Data Editing and Logic: The covering set method from the perspective of logic

Agnes Boskovitz

A thesis submitted for the degree of Doctor of Philosophy at The Australian National University

February 2008
Except where otherwise indicated, this thesis is my own original work.

Agnes Boskovitz
18 February 2008
To Victor.
Acknowledgements

My first and greatest debt is to my parents, Anna and the late Leslie Boskovitz, who encouraged and inspired me in many ways, and in particular gave me a love of learning.

My supervisors, Rajeev Goré, Markus Hegland and John Lloyd, are due my heartfelt thanks: they provided much wise advice and made this work possible. In particular, the chair of my supervisory panel, Rajeev Goré, enthusiastically encouraged me to undertake a PhD, gave generously of his time to explain the ways of academia in general and computer science in particular, and carefully read versions of this thesis. Markus Hegland, the local data mining expert, engaged me in thoughtful discussions, while John Lloyd helped me through many tricky hurdles.

Thank you also to the Australian National University, for granting me scholarships to make my studies feasible and for its wide range of excellent support services.

I am forever grateful to several administrators of the university who ably and efficiently smoothed my way through many administrative problems: Di Kossatz, Michelle Moravec, Rita Murray, Deb Pioch, Ros Taylor and Sue Van Haeften.

I was constantly stimulated and helped by my fellow students Pietro Abate, Asif Ali, Nic Bonnette, Jon Cohen, Dave Davies, Roland Göcke, Evan Greensmith, Charles Gretton, Ed Harrington, Rowan Martin-Hughes, Anthony Mak, Kee Siong Ng, Greg O’Keefe, Cheng Soon Ong, Linda Postniece, Anika Schumann, Tanya Semenova, Sajjad Siddiqi, Kerry Trentelman, Florian Widmann, Paul Wong, and Xiaobing Wu.

Several people other than my supervisors were also kind enough to comment on parts of this thesis. I am exceedingly grateful to Mike Bankier, Keith Farwell, Victor Isaacs, Paulette Lieby, Cheng Soon Ong, Linda Postniece, Peter Radonyi, Ioana Schiopu-Kratina, Anatole Sykley and Kerry Trentelman for their helpful comments.

I am very grateful to Keith Farwell, the editing expert at the Australian Bureau of Statistics, for helping me organise the Editing Meetings at the 55th Session of the International Statistical Institute and explaining many of the practicalities of editing.

I also learnt about editing from John Kovar of Statistics Canada. I thank him for inviting me to present my work at the UNECE Work Session on Statistical Data Editing in Ottawa, where I met and heard from many editing experts.

I was fortunate to be able to have discussions with, and receive feedback from, two of the pioneering researchers in the area: Ton de Waal and Bill Winkler very graciously gave me of their time both in emails and in person.

It is a particular pleasure to recall the assistance of Ioana Schiopu-Kratina. A friendship forged many years ago at McGill University has stood the test of many discussions since on matters mathematical and otherwise.

Most importantly, I could not have done any of this work without the devoted support and encouragement of my darling husband Victor Isaacs.
Errors in collections of data can cause significant problems when those data are used. Therefore the owners of data find themselves spending much time on data cleaning. This thesis is a theoretical work about one part of the broad subject of data cleaning - to be called the covering set method. More specifically, the covering set method deals with data records that have been assessed by the use of edits, which are rules that the data records are supposed to obey. The problem solved by the covering set method is the error localisation problem, which is the problem of determining the erroneous fields within data records that fail the edits. In this thesis I analyse the covering set method from the perspective of propositional logic. I demonstrate that the covering set method has strong parallels with well-known parts of propositional logic. The first aspect of the covering set method that I analyse is the edit generation function, which is the main function used in the covering set method. I demonstrate that the edit generation function can be formalised as a logical deduction function in propositional logic. I also demonstrate that the best-known edit generation function, written here as FH (standing for Fellegi-Holt), is essentially the same as propositional resolution deduction. Since there are many automated implementations of propositional resolution, the equivalence of FH with propositional resolution gives some hope that the covering set method might be implementable with automated logic tools. However, before any implementation, the other main aspect of the covering set method must also be formalised in terms of logic. This other aspect, to be called covering set correctibility, is the property that must be obeyed by the edit generation function if the covering set method is to successfully solve the error localisation problem. In this thesis I demonstrate that covering set correctibility is a strengthening of the well-known logical properties of soundness and refutation completeness. What is more, the proofs of the covering set correctibility of FH and of the soundness / completeness of resolution deduction have strong parallels: while the proof of soundness / completeness depends on the reduction property for counter-examples, the proof of covering set correctibility depends on the related lifting property. In this thesis I also use the lifting property to prove the covering set correctibility of the function $FCF_\omega$, which is based on the Field Code Forest Algorithm. In so doing, I prove that the Field Code Forest Algorithm, whose correctness has been questioned, is indeed correct. The results about edit generation functions and covering set correctibility apply to both categorical edits (edits about discrete data) and arithmetic edits (edits expressible as linear inequalities). Thus this thesis gives the beginnings of a theoretical logical framework for error localisation, which might give new insights to the problem. In addition, the new insights will help develop new tools using automated logic tools. What is more, the strong parallels between the covering set method and aspects of logic are of aesthetic appeal.
Acknowledgements vii

Abstract ix

1 Introduction 1
  1.1 Introduction ................................................................. 1
  1.2 Data errors and data editing ........................................... 1
  1.3 Automation of editing ................................................... 3
  1.4 Detecting incorrect data records ..................................... 4
  1.5 Handling incorrect data records ...................................... 5
  1.6 Error localisation ......................................................... 8
  1.7 The covering set method ................................................ 10
  1.8 Covering set method and logic ......................................... 12
  1.9 Assumed background knowledge ....................................... 13

2 The covering set method 17
  2.1 Introduction ................................................................. 17
  2.2 Error correction and error localisation ............................... 18
  2.3 Solving the error localisation problem using covering sets ........ 22
  2.4 The function FH ............................................................ 29
  2.5 The functions ENFH, MFH and MENFH ................................. 33
  2.6 The Field Code Forest Algorithm ...................................... 40
  2.7 Practicalities of the covering set method ............................ 44
  2.8 Conclusion .................................................................. 46
  2.9 Appendix: Different statements of the FCF Algorithm .......... 49

3 Propositional logic and Boolean algebras 57
  3.1 Introduction ................................................................ 57
  3.2 Propositional logic .......................................................... 57
    3.2.1 Syntax .................................................................. 58
    3.2.2 Semantics .............................................................. 59
    3.2.3 Clauses .................................................................. 63
    3.2.4 Deduction .............................................................. 64
    3.2.5 Relationship between deduction and logical implication .... 66
    3.2.6 The propositional satisfiability problem ....................... 70
  3.3 Boolean algebra ............................................................... 72
    3.3.1 Definition and immediate properties of Boolean algebras.... 72
6.8 Error correction guarantee for $\text{FCF}_\omega$ .......................... 194
6.9 The Field Code Forest Algorithm is correct ............................ 200
6.10 Conclusion ................................................. 200
6.11 Appendix 1: The function $\mathcal{FCF}_\omega$ ......................... 202
6.12 Appendix 2: Winkler’s counter-example to the FCF Algorithm .... 204

7 Arithmetic edits .............................. 207
  7.1 Introduction ........................................... 207
  7.2 Arithmetic edits using sets .................................. 208
  7.3 Logical formalisation of arithmetic edits ....................... 216
  7.4 Covering set correctibility in terms of some natural constructs of logic . 221
  7.5 Arithmetic soundness, completeness and covering set correctibility ... 229
  7.6 Further work .............................................. 234
    7.6.1 The theory Tha ........................................ 234
    7.6.2 The notation $\Box$ ....................................... 235
    7.6.3 A field code forest variation of the function $\mathcal{FM}$ ........ 236
    7.6.4 Extension of the formalisation to more general edits ........... 236
  7.7 Conclusion .............................................. 237

8 Conclusion ............................................. 239

Bibliography ............................................. 243

Index .................................................. 256
Chapter 1

Introduction

1.1 Introduction

This is a theoretical thesis about a very practical topic to be called the covering set method. Two questions arise: what is the covering set method, and why write a theoretical thesis about a very practical topic?

The covering set method is part of the broad subject of data cleaning. More specifically, the covering set method lies within the broad subject of data editing, which is the handling of errors in data after data entry. More specifically still, the covering set method is one method of solving the error localisation problem, which is the problem of determining which are the erroneous fields within an erroneous data record.

In this thesis we will see how the covering set method can be analysed from the point of view of symbolic logic. The thesis demonstrates that the covering set method has strong parallels with well-known aspects of logic.

There are several reasons for doing such an analysis. Firstly, the large collection of sophisticated automated logical tools could potentially be applied to the covering set method. Secondly, logic gives new insights to the covering set method, making it possible to assess error localisation methods and possibly to improve them. Thirdly, the strong parallels with logic are of aesthetic appeal. To quote the theoretical physicist Paul Dirac (1983) from a different but nonetheless relevant context:

one must follow up [the] mathematical idea and see what its consequences are, even though one gets led to a domain which is completely foreign to what one started with.

The sections that follow introduce the topics of data editing, error localisation and the covering set method itself. Precise definitions will be given in Chapter 2.

1.2 Data errors and data editing

Errors in collected data can cause significant problems when those data are used. Such problems include sub-optimal management decisions and poor client service, resulting in wasted expenditure, weak planning, lost customers and a negative public perception. For example, the Data Warehousing Institute estimates that poor quality of customer name and address data costs US businesses $US611 billion per year (Eckerson 2002).
Errors in data are widespread and almost unavoidable. For example, Lorence (2003) presents a survey in which US hospital-based medical records managers reported that an average of 4.6% of records had significant errors resulting in overpayment. As another example, Svanks (1984, pages I-8 to I-9) reports a study of 42,500 Ontario taxation and payout records that, when checked against 300 data constraints, revealed 37,420 data defects.

What do we mean by data errors? There are many things that can go wrong with data, thereby reducing data quality. Wand and Wang (1996) identify four “dimensions” for data quality: data must be complete, unambiguous, meaningful and correct. Greenfield (2006) gives a very similar categorisation: data must be complete, consistent, comprehensible and correct. Other authors have longer lists of dimensions, for example Herzog et al. (2007) give the dimensions as completeness, coherence, relevance, timeliness, accessibility and clarity of results, compatibility, and accuracy. All the lists include a dimension for correctness or accuracy.

In this thesis we focus on one type of data error, namely lack of correctness or accuracy. In particular we focus on “dirty data”, where the values of the data are different from what they purport to represent. There are many other ways in which data can be incorrect, for example through inappropriate formatting, and the violation of the requirement that there be no duplicate records. But we will concentrate on “dirty” or inaccurate data.

There are many approaches to dealing with potentially dirty data. The recommended method is to prevent, through good management practices, the creation of dirty data. However errors inevitably creep in. Another approach is to ignore the issue and treat the data as if they were correct. Such an approach can result in large expenditure when errors are later discovered. The most common approach is to try to find errors after data entry and to do something about them - called data editing.

Data editing is defined in the Glossary of Terms on Statistical Data Editing (United Nations 2000a) as “the activity aimed at detecting and correcting errors in data”, with the implicit assumption that the data are arranged in tables of records. Data editing therefore involves two steps:

1. The first step is to detect incorrect data records, although it may not be possible at this stage to detect which are the incorrect fields. This first step of detecting incorrect records is itself often called data editing.

2. The second step is to handle the incorrect data records. The possibilities include

   (a) doing nothing,

   (b) attempting to correct the data by going back to the original data sources, or

   (c) attempting to correct the data without going back to the original data sources, for example by using the data’s internal redundancies.

In this thesis we focus on step 2(c), of correcting the data without going back to the original sources, and specifically on automating the correction process.
1.3 Automation of editing

Editing is expensive, and manual editing is particularly expensive. For example, the Data Editing Subcommittee of the US Federal Committee on Statistical Methodology (United States 1990, pages 13–14) found that amongst US Federal statistical agencies the cost of editing is typically at least 20% of the total survey cost. Where all the error correction was done manually the editing costs were likely to be over 40% of the total survey cost. Another example was a survey of national statistical agencies in which Luzi, de Waal, and Hulliger (2006) found that about 48% of respondents claimed to spend more than 50% of their overall resources on data editing.

Not only is manual editing expensive, but it can also be inconsistent in the sense that it might not be repeatable. Therefore there is scope for automation.

Automated editing does have its disadvantages. In particular, if there is no revisiting of the original data sources, then at some point an automatic process will have to choose the values of the “corrected” data from a range of possibilities, rather than ascertaining them with certainty from the data suppliers. In addition, automated editing can be inflexible, and might not be able to capture the subtleties that human beings are aware of.

Therefore data that are corrected by purely automated means are not suitable where each individual data record is critical. But they can be suitable when the data are used only in aggregate form. For example, such corrected data can be suitable for survey and census data such as that collected by government statistical agencies. However, the policy in most statistical agencies is to always edit “crucial” records manually (see for example Hoogland (2005), about Statistics Netherlands).

Automated editing has some other disadvantages. Firstly, it might be worthwhile only for large databases. Secondly, there might be a lack of acceptance by staff - Wein (2005) tells of some experiences with introducing new data editing methods.

On the other hand, in addition to being consistent and cheaper than manual editing, automated editing has several side benefits:

1. It allows for the easy production of statistics on data quality - giving the possibility of improvements to data collection methods. Indeed, Granquist and others (e.g. Granquist 1997, 2005, Granquist, Kovar, and Nordbotten 2006) argue that the role of statistical data editing should include such objectives: “learning from the result of editing must be paramount” (Granquist 2005).

2. Statistics on data quality can also give the possibility of improved staff motivation.

3. Apparently high error rates in certain fields might show up incorrect methods of detecting errors.

Many statistical agencies have investigated and implemented automated editing tools. A range of automated editing methods was evaluated by the EUREDIT project (EUREDIT 2004; Charlton 2003). Some of these methods are discussed in the sections to follow.
1.4 Detecting incorrect data records

As explained in Section 1.2, data editing consists of two steps, the first of which is the detection of incorrect data records after data entry.

There are several methods of detecting errors after data entry. The following list divides the methods into four broad categories, the last of which, the use of edits, is the subject of this thesis.

1. Do nothing to detect data errors - cheap in the short term but risky in the longer term.

2. Go back to data sources - can be very expensive, and the inconvenience can cause data suppliers to become less co-operative.

3. Apply statistical methods to the dataset itself. Overviews of various methods are given by Chambers (2004) and Charlton (2002). The methods include statistical modelling and outlier detection (see also Little and Smith (1987), and Ghosh-Dastidar and Schafer (2003)), and neural networks (see also Madsen and Larsen (2000)).

4. Use rules that the data should obey, known from domain knowledge, to find any inconsistencies in the data. Such rules are called edits, which will be defined more precisely in Chapter 2, but the next example gives the idea, albeit with an unlikely case. The example will be developed further later in this chapter.

Example 1.4.1. A data table about individual people includes a data record (to be called v) showing a person who is married, is aged eight, is not eligible to vote, is at preschool, and is a truck driver.

<table>
<thead>
<tr>
<th>Fields:</th>
<th>marital_status</th>
<th>age</th>
<th>vote</th>
<th>school</th>
<th>occupation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record v:</td>
<td>married</td>
<td>8</td>
<td>no</td>
<td>preschool</td>
<td>truck_driver</td>
</tr>
</tbody>
</table>

The following set $E$ of edits is obtained from domain knowledge:

Edit 1. Each married person is aged over 15.

Edit 2. Each person who attends preschool is aged under seven.

Edit 3. Each truck driver is aged over 17.

Edit 4. Each person aged over 17 is eligible to vote.

The data record $v$ fails Edits 1, 2 and 3 and satisfies Edit 4.

The above example is of so-called categorical edits, which deal with discrete data such that it is practical to list all incorrect records. For example, for the above set of edits, the values of the field age can be written as the small finite set $\{0, \ldots, 6\}, \{7, \ldots, 15\}, \{16, 17\}, \{18, \ldots\}$, which can then be used to list the sets of incorrect records. Categorical edits will be defined formally in Chapter 2, Definition 2.2.3.
1.5 Handling incorrect data records

For some categorical edits it is not practical to list all incorrect records. For example, if the table of the above Example 1.4.1 included data about other family members then the edit "mother’s age - child’s age ≥ 12" might be of interest. Although the two "age" fields are discrete, being integers, the set of incorrect records is impractical to list. Hence such edits are usually treated by different means from other categorical edits, and are referred to as “integer edits”.

Edits can also be “arithmetic”, dealing with dense numerical (non-integer) data that have not been discretised. An example of an arithmetic edit is \(-2x_1 + x_2 ≥ 0\), where \(x_1\) and \(x_2\) are rational variables representing the values of fields 1 and 2 respectively.

For most of this thesis, we will deal with categorical edits only. However, Chapter 7 gives an overview of the corresponding results for arithmetic edits. We will not consider integer edits; nor will we consider edits that have both categorical and arithmetic components such as “for each married person, the formula \(-2x_1 + x_2 ≥ 0\) holds”.

The above edits are examples of so-called hard edits (also known as fatal, critical or deterministic edits), because they detect erroneous records with certainty. In contrast, soft edits (also known as statistical, fuzzy or query edits) do not detect errors with certainty, but rather give a probability of error. Another view is that the data records themselves should be seen as probability distributions or as fuzzy sets, and that the edits are also fuzzy (as discussed by Lenz et al. (2006)). This thesis deals with hard edits.

1.5 Handling incorrect data records

After detecting an erroneous record, one needs to decide what to do with it. There are several options: doing nothing, discarding the erroneous record, or correcting the record. Below is some discussion of various methods. The last method, of trying to correct the record via mathematical and statistical techniques, is the direction of this thesis.

1. Do nothing. This may lead to large errors when the data are used. However there are some compromises:

   (a) Do nothing, and seek only consistent query answers, that is, use only those queries whose answers are constant regardless of how the errors are fixed. See for example the review by Bertossi and Chomicki (2004).

   (b) Do nothing, and use measurement error models (or errors-in-variables models) to obtain survey estimates that are free from error. See for example Fuller (1987), and Wansbeek and Meijer (2000).

   (c) Do nothing for selected records. That is, implement selective editing (also known as significance editing) under which nothing is done with records in which the errors are likely to have small effects on estimates. Most statistical agencies use some form of selective (or significance) editing - a summary was prepared by Scarrott (2005). Indeed, Biemer and Lyberg (2003, page 231) comment that “a key lesson from contemporary research on editing is that
not all errors in the data should be fixed, but rather, one should try to
implement selective editing”.

2. Discard the erroneous record. Such an approach is suitable if the errors are large
but their impact is small. But there is a risk of bias. A possibility is to resample
the data source - this is complex to implement correctly and can still lead to bias.

3. Try to correct the erroneous record, with or without going back to the original
data sources:

(a) Go back to the original data sources to obtain corrected data. Although
this method is usually successful and therefore commonly used, it is labour-
intensive and possibly annoying to the data supplier.

(b) Without going back to the original data sources, use mathematical or statistical
techniques, and possibly other data, to replace each erroneous record
by a “corrected” record that is similar to the erroneous record but satisfies
the edits. The corrected record is not guaranteed to be the “true” value
of the record, but a principle of Fellegi and Holt (1973, 1976) requires that
any correction should try to maintain the marginal and preferably the joint
frequency distributions of the variables.

In this thesis we concentrate on the last sub-item in the above list, namely correcting
data without revisiting the data sources. A common approach, first used by Fellegi and
Holt (1973, 1976), is to split the problem into two steps, called the error localisation
problem and the imputation problem:

The error localisation problem is the problem of determining which fields need
to be corrected. A principle of Fellegi and Holt (1973, 1976) states that the
(weighted) number of fields to be changed when making a correction should be
minimised, in which case we have the “smallest weighted error localisation” prob-
lem. (In fact the idea of using “weighted” was added by Garfinkel (1979) and
Liepins (1980b).)

The imputation problem is the problem of estimating correct values for the fields
determined by the error localisation step, while maintaining the frequency distri-
butions. The imputation problem can be addressed in two ways:

Statistical modelling, under which a statistical model is used to predict the
values of the records or fields to be corrected. A challenge for this method
is to satisfy the edits while doing the imputation - for some solutions see

The use of other data as “donors”, which contribute some field values to the
erroneous record. The donors can be either from the current dataset (hot
deck) or from a previous dataset (cold deck).
This thesis deals with the first of the above two steps, namely the error localisation problem. We discuss the error localisation problem in more detail in the next section where we also give an overview of the many solutions available.

But first, we note here that the error correction problem can be seen in a more general framework than that described above. The error correction problem can be set up as the mathematical programming problem of finding the correct records that are “closest” to the erroneous record, together with imputation as required. The main features of any mathematical programming problem are the constraints and the objective function (function to be optimised):

The main constraints are the edits themselves.

The objective function is usually some distance function between the erroneous and the corrected record. The objective is to minimise the distance.

The smallest weighted error localisation problem can be seen as such a mathematical programming problem. The distance function used for the objective function is the (weighted) number of fields at which the erroneous record differs from the corrected record, that is the (weighted) Hamming distance between the two records. Thus the objective is to minimise the (weighted) Hamming distance.

Whether or not the mathematical programming problem is also an error localisation problem, it can in some cases be set up so that its solution is likely to be a unique correct record, in which case no imputation is needed. For example, Thompson et al. (2005), working with particular arithmetic edits called ratio and balance edits, use quadratic programming with a weighted Euclidean distance as an objective function. The disadvantage of the method of Thompson et al. is that most fields are likely to be changed, whereas one would expect that errors occur only in some fields.

Where donors are used, the mathematical programming problem becomes more complex. The donors act as additional constraints, in addition to the edits, because the donors limit the possible imputations. Also, the use of donors can introduce new objective functions related to the donors, such as the distance between the donor and the erroneous record. Any new objective function is in addition to the previously stated objective function of the distance between the correct and the erroneous record. Thus where donors are used, the mathematical programming problem can become a two-stage problem, with two objective functions.

Where the problem is treated as a two-stage mathematical programming problem, the first stage can use the objective function related to the donors, rather than the distance between the erroneous and corrected records. Such a two-stage process is the basis of the “nearest-neighbour imputation methods” or “first donors then fields methods” of two groups: firstly, Bankier et al. (1994–2002), and secondly, Manzari and Reale (2001, 2002), Manzari (2004), Di Zio, Guarnera, Luzi, and Manzari (2005), Bruni, Reale and Torelli (2001, 2002), Bruni and Sassano (2001a), and Bruni (2004a, 2005).

Thus in the more general mathematical programming framework for error correction, the error localisation and imputation phases can be combined into one step. One simultaneously obtains both an error localisation solution and an imputation solution.
However, it is often useful to separate the error localisation step from the imputation step, to allow the imputation step to be analysed more thoroughly.

1.6 Error localisation

As stated in the previous section, the correction of data can be undertaken in two steps, namely error localisation followed by imputation. This thesis concentrates on error localisation, which is the problem of deciding which fields need to be corrected. The next example gives a simple error localisation problem and solution.

**Example 1.6.1.** As in Example 1.4.1, suppose that a database contains the record \( v \), given below, which fails Edits 1 to 3 and satisfies Edit 4, also given below.

<table>
<thead>
<tr>
<th>Fields:</th>
<th>marital_status</th>
<th>age</th>
<th>vote</th>
<th>school</th>
<th>occupation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record ( v ):</td>
<td>married</td>
<td>8</td>
<td>no</td>
<td>preschool</td>
<td>truck driver</td>
</tr>
</tbody>
</table>

**Edit 1.** Each married person is aged over 15.

**Edit 2.** Each person who attends preschool is aged under seven.

**Edit 3.** Each truck driver is aged over 17.

**Edit 4.** Each person aged over 17 is eligible to vote.

Although in practice it might be preferable to discard the record, it can be “corrected”, that is, made to satisfy the edits, by changing the three fields age, vote and school, as in the following possible correction.

<table>
<thead>
<tr>
<th>Fields:</th>
<th>marital_status</th>
<th>age</th>
<th>vote</th>
<th>school</th>
<th>occupation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Possible corrected record:</td>
<td>married</td>
<td>28</td>
<td>yes</td>
<td>none</td>
<td>truck driver</td>
</tr>
</tbody>
</table>

Thus the set \{age, vote, school\} is one possible error localisation solution.

How does one do error localisation? As stated in the previous section, the error localisation problem can be treated as a mathematical programming problem. The constraints are the edits, while the objective function is a distance function between the erroneous and the corrected record. For the smallest weighted error localisation problem the distance function is the weighted Hamming distance between the two records.

The general mathematical programming techniques of the previous section, where the error localisation and imputation steps are combined, automatically include an error localisation solution. Such methods, mentioned on page 7, include the nearest-neighbour imputation methods of Bankier et al. and Manzari et al., and the quadratic programming technique of Thompson et al.
When treating error localisation separately from imputation, there are two related approaches to solving the error localisation problem. The first is to use a pre-existing mathematical programming solver and formalise the error localisation problem in its terms. This can result in a solution method that is not well tuned to the problem. The second method is to formalise the error localisation problem in terms of some particular mathematical programming method and create a solver tuned to the problem.

Several researchers have formalised the error localisation problem in terms suitable for general integer programming or mixed integer programming packages. Therefore all edits are formalised as inequalities or equations. Examples include:

- Schaffer (1987), who uses the Sperry Univac Functional Mathematical Programming System (FMPS);
- Bruni and Sassano (2001a), who use the Volume Algorithm and the branch-and-bound solver Xpress;
- de Waal (2003a, Chapters 3 and 11), who uses the branch-and-bound solver ILOG CPLEX; and
- Bruni (2004a, 2005), who also uses ILOG Cplex.

Rather than using standard solvers, many researchers have focused on specific algorithms or techniques with potential for the error localisation problem. Most of the techniques involve some generation of new constraints and some systematic search through possible solutions. The following are some examples of techniques suitable for mixed integer programming.

- Vertex generation methods based on the Chernikova algorithm (Chernikova 1964, 1965) as developed by Rubin (1975). They have been applied to error localisation by Sande (1978), Schiopu-Kratina and Kovar (1989), Fillion and Schiopu-Kratina (1991), and de Waal (2003b; 2003a, Chapter 5).
- Treat as a dynamic disjunctive facet problem, presented by de Waal (2003a, Chapter 6).
- Use direct branch-and-infer techniques, presented by de Waal et al. (de Waal 2000, 2003a (Chapter 8), 2005; de Waal and Quere 2003).
- Use heuristic methods, where one starts with a feasible (non-minimal) error localisation solution, such as the set of all fields, and then progressively reduces its size. Examples include Riera-Ledesma and Salazar-González (2007b) and Garfinkel, Kunnathur and Liepins (1986a, 1988).

Other researchers have used techniques from logic to solve the error localisation problem:
Barcaroli (1993) formalises edits and data records as formulae of first-order logic. For each erroneous record, he systematically tests new field values with a refutation-by-resolution procedure until he finds no inconsistency. This method also gives a potential corrected record, although without attempting to ensure that statistical distributions are maintained.

Franconi et al. (2001) use disjunctive logic programming with prioritised constraints implemented as DLP\textsuperscript{w}, although there is no decision procedure that correctly determines whether an arbitrary formula expressed in DLP\textsuperscript{w} is valid. The prioritised constraints ensure that the distance function is the Hamming distance between the erroneous and the corrected record. The method also gives a potential corrected record, although without attempting to ensure that statistical distributions are maintained.

Bruni (2004b) uses a modified satisfiability (SAT) solver with any distance function, and works with categorical edits. The method returns a near-optimal solution. The disadvantage of this method is that the underlying SAT solver is designed to find some correct record, of which there are many, rather than the best one.

Rather than formalising the individual procedures, Franconi (2005) suggests formalising the problem statement itself as a logical problem. Bertossi et al. (2003, 2005a, 2005b) have done a similar formalisation for consistent query answering.

The area of logic-based diagnosis (for example Reiter (1987), de Kleer, Mackworth, and Reiter (1990)) might be able to be applied to the error localisation problem, although, in order to apply logic-based diagnosis, one must be able to express “field \(j\) is incorrect” within the logic.

Finally, the error localisation problem can also be solved by the technique of Fellegi and Holt (1973, 1976), which is an example of what I will call the “covering set method”. The covering set method is the main topic of this thesis and is discussed in more detail in the next section.

### 1.7 The covering set method

One method of trying to solve the error localisation problem defined on page 6 is by the covering set method. The examples below will give the idea of how the covering set method works. The first example is a naïve and unsuccessful attempt to use the covering set method, but the examples that follow demonstrate how to use the covering set method correctly.

**Example 1.7.1.** As in Example 1.4.1, suppose that a database contains the record \(v\), given below, which fails Edits 1 to 3 and satisfies Edit 4, also given below.

<table>
<thead>
<tr>
<th>Fields:</th>
<th>marital_status</th>
<th>age</th>
<th>vote</th>
<th>school</th>
<th>occupation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record (v):</td>
<td>married</td>
<td>8</td>
<td>no</td>
<td>preschool</td>
<td>truck_driver</td>
</tr>
</tbody>
</table>
Edit 1. Each married person is aged over 15.

Edit 2. Each person who attends preschool is aged under seven.

Edit 3. Each truck driver is aged over 17.

Edit 4. Each person aged over 17 is eligible to vote.

Since age appears in all three of the failed edits - we will say it “covers” them - it might seem that the record \( v \) can be corrected by changing the age, so that the set \( \{ \text{age} \} \) would be an error localisation solution. However some experimentation shows that the three edits cannot be satisfied by changing only the age, and the set \( \{ \text{age} \} \) is not an error localisation solution.

In the above example, we tried to find an error localisation solution by finding a field (age) that “covers” all the failed edits in the sense that it is involved in each failed edit. However the attempt failed. The reason is that there are additional edits which are implied by the given set of edits and which do not involve age, but which are also failed by the given record.

Example 1.7.2. We continue with Example 1.7.1. The following edits are implied by the given edit set \( E = \{ \text{Edit 1}, \ldots, \text{Edit 4} \} \), are failed by the record \( v \), but do not involve age.

Edit 5. Each truck driver is eligible to vote. (implied by Edits 3 and 4)

Edit 6. No married person is in preschool. (implied by Edits 1 and 2)

Edit 7. No truck driver is in preschool. (implied by Edits 2 and 3)

We will call the set \( \{ \text{Edit 1}, \ldots, \text{Edit 7} \} = G(E) \) the generated set of edits. We find that we need three fields to “cover” the failed edits of \( G(E) \) (Edits 1, 2, 3, 5, 6 and 7 - Edit 4 omitted), in the sense that each failed edit involves at least one of the three fields. For example the field set \( \{ \text{age}, \text{vote}, \text{school} \} \) covers the failed edits, and as seen in Example 1.6.1, it is an error localisation solution for the edit set \( E \) and the record \( v \).

We could have chosen to generate fewer edits, say only Edit 5 (omitting Edits 6 and 7), so that \( G(E) = \{ \text{Edit 1}, \ldots, \text{Edit 5} \} \). In that case the set of failed edits would have been smaller (Edits 1, 2, 3 and 5) and we would have found that the two fields age and vote cover our failed edits. However, the two fields age and vote cannot be changed to correct the record \( v \), and the set \( \{ \text{age}, \text{vote} \} \) is not an error localisation solution. Thus it is vital to compute sufficiently many of the implied edits.

On the other hand, we could also have chosen to include a (silly) non-implied edit in \( G(E) \) such as:

Edit 8. Each truck driver is unmarried.
In this case we would have said that \( G(E) = \{\text{Edit 1}, \ldots, \text{Edit 8}\} \). Edit 8 is failed by the record \( v \). In this case the error localisation solution \( \{\text{age, vote, school}\} \) does not cover the failed edits of \( G(E) \), because neither \text{age} nor \text{vote} nor \text{school} is involved in Edit 8. Thus it is vital to include only implied edits in \( G(E) \).

In the above example, we again tried to find an error localisation solution by using the idea of “covering”. But this time we expanded our set \( E \) of edits by generating some additional edits to obtain the generated set \( G(E) = \{\text{Edit 1}, \ldots, \text{Edit 7}\} \). We found that we need three fields (\text{age, vote, school}) to “cover” all the failed edits of \( G(E) \), in the sense that each of the failed edits involves at least one of the three fields. We also found that these three fields form an error localisation solution.

The above example also demonstrates that it is possible to generate too few or too many new edits. If there are too few generated edits (such as \( G(E) = \{\text{Edit 1}, \ldots, \text{Edit 5}\} \)), then the covering set need not be an error localisation solution. On the other hand, if \( G(E) \) contains wrong edits, such as Edit 8, some error localisation solutions might not be covering sets.

We can summarise the covering set method as follows. Given an edit set \( E \), find a generated edit set \( G(E) \). Then given a record \( v \) that fails \( E \), find a covering set of the edits of \( G(E) \) failed by \( v \).

The above Example 1.7.2 demonstrates that the covering sets output by the covering set method need not be the same as the solutions to the error localisation problem. That is, the covering set method is not necessarily successful at solving the error localisation problem.

However, the covering set method is successful if the function \( G \) is suitably chosen, in which case we say that \( G \) is covering set correctible - to be defined precisely in Chapter 2. For example Fellegi and Holt (1973, 1976) used a function to be called FH - to be defined in Chapter 2 - as the function \( G \): using the edit generation function FH the covering set method is always successful.

Another example of a covering set correctible function is the function \( FCF_\omega \) (standing for field code forest). Its covering set correctibility has been questioned, for example by Winkler (1997). However Chapter 6 contains a proof that it is indeed covering set correctible.

Since the edit generation functions used in the covering set method generate new implied edits from old edits, the covering set method clearly has some logical content. This thesis is about the properties of the covering set method and covering set correctible functions from the perspective of logic.

### 1.8 Covering set method and logic

If it were possible to formalise the covering set method in terms of logic, then it might be possible to use some of the tools of logic to solve the error localisation problem. This thesis gives a rigorous analysis of the logical content of the covering set method.

There are two main steps in the covering set method: the first step is the generation of new edits from old; while the second step is the use of covering sets to find error
localisation solutions. This thesis deals with both steps.

The first step, the generation of new edits from old, can be seen as logical deduction in propositional logic. Chapter 4 gives details for categorical edits, while Chapter 3 gives relevant background on logic.

What is more, the edit generation function FH turns out to be essentially the same as the well-known logical deduction function known as resolution deduction, defined in Chapter 3. Chapter 4 presents the details of the equivalence of FH and resolution deduction.

The covering set method includes a second step, namely the finding of covering sets, which is only useful if the edit generation function is covering set correctible. It turns out that the property of covering set correctibility is a strengthening of the logical property of completeness and soundness. Chapter 5 gives details for categorical edits, while Chapter 3 gives the relevant background on logic.

The above discussion refers only to categorical edits. It turns out, with a suitable formalisation and some further analysis, that equivalent results apply for arithmetic edits. The results for arithmetic edits are presented in Chapter 7.

The proof of the covering set correctibility of FH has a strong similarity to the proof of completeness for resolution deduction. The similarity is not surprising, since FH is essentially the same as resolution deduction, and covering set correctibility is a strengthening of completeness and soundness.

The main property in the proof of the covering set correctibility of FH is called the lifting property, to be defined in Chapter 2. The lifting property is very similar to the main property, known as the reduction property for counter-examples, in the proof of the completeness of resolution deduction. The lifting property is applied elsewhere: in the proof of the covering set correctibility of the function FCFω in Chapter 6, and in the proof of the covering set correctibility of the function FM introduced in Chapter 7.

Why bother with relating the covering set method to logic? Logic is the study of deduction, and most methods of automated error localisation can be seen as a trade-off between search and deduction. A conversion of the error localisation problem to logic could be useful because there are many automated logic tools which could be relevant to error localisation. Conversely, some of the error localisation tools might be useful for the automated reasoning tools. In addition, the logical formalisations are the beginnings of a theoretical logical framework for error localisation, which might give new insights. What is more, the strong parallels between error localisation and logic have aesthetic appeal.

1.9 Assumed background knowledge

I have tried to make the thesis self-contained and accessible to a broad audience, so there are many items which may seem unnecessary to an experienced mathematical reader. In particular parts of Chapter 2 and all of Chapter 3 repeat known knowledge.

The thesis uses a large number of special symbols and terms. The index to the thesis gives the location of the definition of each symbol and term.
This chapter concludes with some additional background knowledge: some notation, and the terms partial order and total order, subfunction and superfunction, superior and inferior, and sequentially generated, as well as some of their properties.

Some notation

**Strings.** We will use notation such as \((x_1, \ldots, x_n)\) both for strings of arbitrary length and tuples of fixed length \(n\). None of the strings we use will have repeated terms, for example we will not have strings such as \((1, 2, 3, 2)\). We will apply to strings the set operations such as \(\setminus\) and \(\cap\) by applying them to the underlying sets. For example, \((1, 2, 3) \setminus (2) = (1, 3)\). The expression \(\sigma \subseteq \tau\) means that every element of \(\sigma\) is in \(\tau\) and the elements of \(\sigma\) are in the same order as in \(\tau\). For example, \((1, 3) \subseteq (1, 2, 3)\), but \((3, 1) \not\subseteq (1, 2, 3)\). Also, the notation \((x_1, \ldots, x_j)\) means the string \((x_k \mid i \leq k \leq j)\); if \(j < i\) then \((x_i, \ldots, x_j)\) means the empty string \(\)\). We will use the symbol \(\circ\) for string concatenation. For example \((x_1, \ldots, x_n) \circ (y_1, \ldots, y_m) = (x_1, \ldots, x_n, y_1, \ldots, y_m)\).

**Composition of functions.** We will also use the symbol \(\circ\) for the composition of functions. We therefore use \(\circ\) for both the composition of functions and for the concatenation of strings, but the context will remove ambiguity. For example, if \(f, g\) and \(h\) are functions and \(x\) is in the domain of \(h\) and the three functions can be composed so that \(f\left(g\left(h(x)\right)\right)\) is well defined, then we write \(f \circ g \circ h(x) = f\left(g\left(h(x)\right)\right)\). We will use both notations.

**Null product of sets.** If \(n \geq k > l \geq 1\), and \(X_1, \ldots, X_n\) and \(Y\) are sets, then 
\[
(\prod_{j=k}^{l} X_j) \times Y = Y \times (\prod_{j=k}^{l} X_j) = Y.
\]

**Power set.** We use the symbol \(\mathcal{P}\) to represent power sets, that is, if \(X\) is a set, then \(\mathcal{P}(X)\) is the set of all subsets of \(X\).

**Empty set.** We use the symbol \(\emptyset\) to represent the empty set.

**Partially and totally ordered sets**

**Definition 1.9.1.** A **partial order** on a set \(X\) is a binary relation on \(X \times X\) that is transitive, reflexive, and anti-symmetric. A **total order** \(\leq\) on \(X\) is a partial order such that if \(x, y \in X\) and \(x \neq y\), then \(x \leq y\) or \(y \leq x\) but not both. Sometimes this definition of a total order is called a **strict total order**.

We have two methods of representing total orderings of a finite set. Firstly, we can use a binary relation symbol, such as \(\leq\). Secondly, we can use the actual sequence representing the order (permutation of the set represented as a string), for example if \(X = \{i_1, \ldots, i_n\}\) and under the total ordering \(i_1 \leq \cdots \leq i_n\), then we will also write the total order as \((i_1, \ldots, i_n)\).
Definition 1.9.2. If \( Z \) is a partially ordered set, then we write \( \text{Min}(Z) \) for the set of minimal elements of \( Z \), and we write \( \text{Max}(Z) \) for the set of maximal elements of \( Z \).

Superfunctions, subfunctions, superior and inferior functions

Definition 1.9.3. Let \( X \) be a set and let \( Y \) be a partially ordered set with the ordering relation \( \leq \). Let \( G, H : X \to Y \). Then the function \( H \) is a subfunction of the function \( G \) if for each element \( x \) of \( X \), we have that \( H(x) \leq G(x) \). The function \( H \) is a superfunction of the function \( G \) if for each element \( x \) of \( X \), we have that \( G(x) \leq H(x) \).

Definition 1.9.4. Let \( Z \) be a partially ordered set with the ordering relation \( \leq \). Suppose that \( X \subseteq Z \) and \( Y \subseteq Z \). We say that the set \( X \) is superior to the set \( Y \) if for each \( y \) in \( Y \) there is an \( x \) in \( X \) such that \( y \leq x \). We say that the set \( X \) is inferior to the set \( Y \) if for each \( y \) in \( Y \) there is an \( x \) in \( X \) such that \( x \leq y \).

Let \( V \) be a set and suppose we have the functions \( G, H : V \to Z \). We say that \( G \) is a superior function to \( H \) if for each subset \( U \) of \( V \), the set \( G(U) \) is superior to the set \( H(U) \). We say that \( G \) is an inferior function to \( H \) if for each subset \( U \) of \( V \), the set \( G(U) \) is inferior to the set \( H(U) \).

Note: inferior is also called “dense” in the case when \( X \subseteq Y \), see for example Planetmath (2002).

Note that the relations “inferior” and “superior” are not partial orders because they are not anti-symmetric. For example, two sets can be mutually inferior without being equal. However mutually inferior sets do have the same minimal elements as per part 1 of the next lemma, which also states some simple results about the connection between maximal and minimal sets and superiority and inferiority.

Lemma 1.9.5.

1. Let \( Z \) be a partially ordered set, with ordering relation \( \leq \). Suppose that \( X \subseteq Z \) and \( Y \subseteq Z \), and that \( X \) is inferior to \( Y \) and that \( Y \) is inferior to \( X \). Then \( \text{Min}(X) = \text{Min}(Y) \).

2. Suppose that \( V \) is a set. Let \( G : \mathcal{P}(V) \to \mathcal{P}(V) \), and let \( \text{Max}G : \mathcal{P}(V) \to \mathcal{P}(V) \) be defined for each subset \( U \) of \( V \) by

\[
\text{Max}G(U) = \text{Max}(G(U)).
\]

Then \( \text{Max}G \) is a superior function to \( G \).

Let \( \text{Min}G : \mathcal{P}(V) \to \mathcal{P}(V) \) be defined for each subset \( U \) of \( V \) by

\[
\text{Min}G(U) = \text{Min}(G(U)).
\]

Then \( \text{Min}G \) is an inferior function to \( G \).
Proof.

1. We show that $\text{Min}(X) \subseteq \text{Min}(Y)$. Let $x \in \text{Min}(X)$. We will first show that $x \in Y$, and then that $x \in \text{Min}(Y)$.

To show that $x \in Y$: Since $Y$ is inferior to $X$, there is a $y$ in $Y$ such that $y \leq x$. Since $X$ is inferior to $Y$, there is an $x'$ in $X$ such that $x' \leq y$. Hence $x' \leq x$, by the transitivity of $\leq$. But $x$ is minimal in $X$; hence $x' = x$. Hence $x \leq y \leq x$, and so $y = x$ by the anti-symmetry of $\leq$. Hence $x \in Y$.

To show that $x \in \text{Min}(Y)$: Suppose that $y^* \in Y$ and $y^* \leq x$. Then since $X$ is inferior to $Y$, there is an $x^*$ in $X$ such that $x^* \leq y^*$. Hence $x^* \leq x$, by the transitivity of $\leq$. But $x$ is minimal in $X$; hence $x^* = x$. Hence $x \leq y^* \leq x$, and so $y^* = x$ by the anti-symmetry of $\leq$. Hence $x$ is minimal in $Y$, and $x \in \text{Min}(Y)$, as required.

Similarly, by symmetry, $\text{Min}(Y) \subseteq \text{Min}(X)$, and hence $\text{Min}(X) = \text{Min}(Y)$.

2. Follows from the definitions of maximal and minimal. \hfill \square

Sequentially generated sets

Definition 1.9.6. Suppose that $S$ is a set, that $B \subseteq S$, and that $R \subseteq P(S) \times S$. Then the set $B_*$ is sequentially generated by the set $B$ and the relation $R$ if, for each $b$ in $B_*$, there is a sequence $b_1, \ldots, b_n = b$ of elements of $S$ such that for each $j = 1, \ldots, n$

1. $b_j \in B$, or

2. there is a subset $X$ of $\{b_1, \ldots, b_{j-1}\}$ such that $(X, b_j) \in R$.

Sequentially generated sets are the same as defined inductively sets:

Theorem 1.9.7. Suppose that $S$ is a set, that $B \subseteq S$, and that $R \subseteq P(S) \times S$, and that $B_*$ is sequentially generated by $B$ and $R$. Suppose also that that $B^*$ is inductively defined as follows:

1. $B \subseteq B^*$, and

2. if $Y \subseteq B^*$ and $(Y, b) \in R$, then $b \in B^*$.

Then $B_* = B^*$.

Proof. See the book of Rasiowa and Sikorski (1968, pages 183–186), which deals with the specific case where $S$ is a set of logical formulae. The book of Segerberg (1982, pages 5–9) also gives a proof, although it gives a different but equivalent definition of $B_*$. \hfill \square
Chapter 2

The covering set method

2.1 Introduction

In Chapter 1 we observed the idea of using “covering sets” to solve the error localisation problem. In this chapter we formally define the technique of using covering sets, called the “covering set method” (Definition 2.3.12), supported by the underpinning definitions and some basic properties.

Since the covering set method is used to solve the error localisation problem, we first define the error localisation problem, as well as the broader error correction problem. Both the error correction problem and the error localisation problem are defined in the next section (in Definitions 2.2.6 and 2.2.8 respectively). Having defined the error localisation problem, we will be ready to formally define the covering set method, which we do in the subsequent section, Section 2.3.

As seen in Chapter 1, the method of using covering sets does not always work in the sense that it does not always succeed in solving the error localisation problem. But it does work when certain “edit generation” functions (Definition 2.3.13) are used within the method. The method works when the relevant edit generation function has a property called “covering set correctibility” (Definition 2.3.14).

I shall present several functions that are covering set correctible, in Sections 2.4 to 2.6. I start, in Section 2.4, with the function FH, originally defined by Fellegi and Holt (1973, 1976). The functions presented in the subsequent two sections (Sections 2.5 and 2.6) are all subfunctions of FH. They are called ENFH, MFH, MENFH and FCF$\omega$. I also present the proofs of the covering set correctibility of the various functions, although the proof for FCF$\omega$ is complicated and deferred until Chapter 6.

There are also some techniques to speed up the covering set method and to reduce its memory usage. Some of the techniques tinker with the edit generation functions, while others are compromises to the method. One technique that has not been much explored is the use of logic, the topic of this thesis. In Section 2.7 I discuss some of the practicalities of using the covering set method.

This chapter is mainly a summary of material previously published by other authors. However there are some aspects that are new, or published in some of my previous papers (Boskovitz et al.2005; Boskovitz and Goré 2005).

The first new aspect is the property of covering set correctibility. Other authors
have implicitly used a component of covering set correctibility, which I call the error correction guarantee (Definition 2.3.16). Garfinkel et al. (1986b) have used a property called sufficiency, which is similar to the error correction guarantee. But I am not aware of any authors using the other component of covering set correctibility, which I call error correction totality (Definition 2.3.16).

The second new aspect concerns the proofs of covering set correctibility for the various functions. The proofs of one component, the error correction totality, are new, although Fellegi and Holt (1973, 1976) did prove the main result needed to prove error correction totality. As to the other component, the error correction guarantee, the situation is more complicated. For the functions other than $\text{FCF}_\omega$, the proofs have been previously published in full or in essence. However some of the results are special cases of more general results about superior functions and subfunctions, and the proofs presented here are in those terms.

This chapter focuses on categorical edits, rather than arithmetic edits. The corresponding definitions and properties for arithmetic edits are presented in Chapter 7.

We start with the background needed to define the covering set method, namely the definition of the error localisation problem.

### 2.2 Error correction and error localisation

In this section, we define the problem solved by the covering set method. We also define the various components of the problem. The covering set method solves the error localisation problem, to be defined in Definition 2.2.8, which is part of a technique to solve the error correction problem, to be defined in Definition 2.2.6. Hence we define the error correction problem and its components.

We assume that we are dealing with data arranged in records, which are $N$-tuples, where $N$ is fixed for all records under consideration.

**Definition 2.2.1.** A record $v$ is an $N$-tuple $(v_1, \ldots, v_N)$ where $N$ is a positive integer; $j = 1, \ldots, N$ are called fields; and $v_j \in A_j$ where each $A_j$ is a set containing at least two elements, called the $j$th field domain. We assume that $N$ is the same for all records. The set of all possible records is the data domain $D = A_1 \times \cdots \times A_N$ (also called the domain).

**Example 2.2.2.** A data table about school students includes a data record showing a person who is aged 6, has a driver’s licence and is in Grade 8 of school. The data table includes three fields, with field domains listed below:

- $A_1$ (relevant ages), which we relabel $A_{\text{age}} = \{5, 6, \ldots, 20\}$;
- $A_2$ (whether someone has a driver’s licence), which we relabel $A_{\text{driver}} = \{\text{yes, no}\}$;
- $A_3$ (school grade levels), which we relabel $A_{\text{grade}} = \{1, 2, \ldots, 12\}$.

The data domain $D$ is $A_{\text{age}} \times A_{\text{driver}} \times A_{\text{grade}}$.

The record in this example is $v = (6, \text{yes}, 8)$. 
Edits (defined precisely below) are used to specify potential errors in each record. Edits apply to one record at a time. We define categorical edits below, while we leave the definition of arithmetic edits to Chapter 7. Each categorical edit specifies a rejection region, that is, a set of unacceptable records. More formally we have:

**Definition 2.2.3.** A categorical edit is a subset of \( D = A_1 \times \cdots \times A_N \).

A record \( v \) is incorrect with respect to the categorical edit \( e \) if \( v \in e \). We can also say that \( v \) fails \( e \). If \( v \notin e \), we say that \( v \) is correct with respect to \( e \), or \( v \) satisfies \( e \). The record \( v \) fails a set \( E \) of edits if \( v \) fails some edit in \( E \). The record \( v \) satisfies a set \( E \) of edits if \( v \) satisfies all edits in \( E \).

Note: The set \( D \) is an edit, although it is failed by every record.

**Example 2.2.4.** Suppose that the following requirements apply to the data table of Example 2.2.2:

Requirement 1: A person with a driver’s licence must be in at least Grade 11.
Requirement 2: A person in Grade 7 or higher must be at least 10 years old.

The edits \( e_1 \) and \( e_2 \) corresponding to the above two requirements are the subsets of \( D \) that fail each respective requirement:

\[
\begin{align*}
e_1 & = A_{age} \times \{yes\} \times \{1, 2, \ldots, 10\}, \\
e_2 & = \{5, 6, \ldots, 9\} \times A_{driver} \times \{7, 8, \ldots, 12\}.
\end{align*}
\]

We will write \( EE = \{e_1, e_2\} \).

Consider the record \( vv = (6, yes, 8) \). Since \( vv \in e_1 \) and \( vv \in e_2 \), we say that \( vv \) fails both edits.

We will use the term “dominate” to describe the strict superset relationship between edits:

**Definition 2.2.5.** (Kunnathur 1982) Edit \( e \) dominates edit \( e' \) if \( e \supset e' \).

Having defined edits and records, we can now formally define the data correction problem:

**Definition 2.2.6.** Given a record \( v \) and a set \( E \) of edits, the error correction problem is the problem of choosing a suitable record \( w \) that satisfies each edit in \( E \). We say that the record \( w \) corrects \( v \) with respect to \( E \).

**Example 2.2.7.** As in Examples 2.2.2 and 2.2.4, suppose that

\[
\begin{align*}
D & = A_{age} \times A_{driver} \times A_{grade} \\
EE & = \{e_1, e_2\}, \text{ where} \\
e_1 & = A_{age} \times \{yes\} \times \{1, 2, \ldots, 10\}, \\
e_2 & = \{5, 6, \ldots, 9\} \times A_{driver} \times \{7, 8, \ldots, 12\}, \text{ and} \\
vv & = (6, yes, 8), \text{ which fails both edits } e_1 \text{ and } e_2.
\end{align*}
\]
Since the record $ww = (6, \text{no}, 1)$ satisfies both edits, we say that $ww$ corrects the record $vv$ with respect to the edit set $EE$.

How to choose a “suitable” record $w$? This thesis deals with one way: use the covering set method to decide which fields should be changed (error localisation), and then impute on those fields. Hence we give a formal definition of the error localisation problem:

**Definition 2.2.8.** Let $v = (v_1, \ldots, v_N)$ be a record and $E$ be a set of edits. The error localisation problem is the problem of deciding which sets of fields can be changed to correct $v$ with respect to $E$. That is, it is the following problem: given a set $C$ of fields and a record $v$, decide whether there is a record $w = (w_1, \ldots, w_N)$ such that

a. $C \supseteq \{ j \mid w_j \neq v_j \}$, and

b. $w$ satisfies all edits in $E$.

If there exists a $w$ such that $a$ and $b$ are satisfied, then we say that the set $C$ yields a correction $w$ to $v$ with respect to $E$ and we shall also say that $C$ is a solution to the error localisation problem for $v$ and $E$.

The set of solutions to the error localisation problem for the record $v$ and the edit set $E$ will be written $EL(E, v)$. This means that $C$ yields a correction to $v$ with respect to $E$ if and only if $C \in EL(E, v)$.

**Note 1:** $v$ satisfies $E$ if and only if $\emptyset \in EL(E, v)$.
**Note 2:** If $D \in E$ then $EL(E, v) = \emptyset$.

**Example 2.2.9.** In Example 2.2.7, the correct record $ww$ differs from the incorrect record $vv$ on the fields driver and grade. We say that the field set $\{\text{driver, grade}\}$ yields the correction $ww$, among others, to $vv$ with respect to $EE$. It is impossible to correct $vv$ by changing just one field, but any pair of fields is an error localisation solution for $vv$ with respect to $EE$, as is the set of all three fields. Thus

$$EL(EE, vv) = \{ \{\text{age, driver}\}, \{\text{age, grade}\}, \{\text{driver, grade}\}, \{\text{age, driver, grade}\} \}.$$

In practice we want to solve a tighter problem than the above error localisation problem. We are only interested in the “best” field sets $C$ which we choose according to some criterion. A suitable criterion is to seek a smallest set of weighted fields that can be changed to correct $v$, as used by Garfinkel et al. (1986b). We call this problem the “smallest weighted error localisation” problem and define it as follows.

**Definition 2.2.10.** The smallest weighted error localisation problem is the following problem:

Given an edit set $E$, a record $v = (v_1, \ldots, v_N)$, and a vector $b = (b_1, \ldots, b_N)$ of $N$ positive numbers (weights), decide whether the field set $C$ minimises $\sum_{i \in C} b_i$ such that the set $C$ yields a correction to $v$ with respect to $E$.

The set of solutions to the smallest weighted error localisation problem for record $v$ with respect to edit set $E$ and weights vector $b$ will be written $SEL(E, v, b)$. 
Note 1: Each solution to the smallest weighted error localisation problem is a weighted example of a “cardinality-oriented repair” defined by Bertossi et al. (2003, Definition 2(c)).

Note 2: Another criterion for choosing the “best” field set $C$ is to choose a set $C$ that yields a correction and that is minimal under set inclusion. Bertossi et al. (2003, Definition 2(b)) give the name “subset-oriented repair” to any correction obtained by such a criterion.

**Example 2.2.11.** As in the previous examples, suppose that

$$D = A_{age} \times A_{driver} \times A_{grade},$$

$$EE = \{e_1, e_2\},$$

where

$$e_1 = A_{age} \times \{yes\} \times \{1, 2, \ldots, 10\},$$

$$e_2 = \{5, 6, \ldots, 9\} \times A_{driver} \times \{7, 8, \ldots, 12\},$$

and

$$vv = (6, yes, 8),$$

which fails both edits $e_1$ and $e_2$.

As seen in Example 2.2.9, the record $vv$ can be corrected by suitable changes to any pair of fields, but cannot be corrected by changing only one field. If each field is weighted equally, that is $b = (1, \ldots, 1)$, then $\mathcal{SEL}(EE, vv, (1, \ldots, 1))$ is the set of all pairs of fields:

$$\mathcal{SEL}(EE, vv, (1, \ldots, 1)) = \{\{age, driver\}, \{age, grade\}, \{driver, grade\}\}.$$

It turns out that the error localisation problem is more easily addressed if all the edits are of normal form, defined next.

**Definition 2.2.12.** A non-empty categorical edit $e$ is of **normal form** if $e = A_1^e \times \cdots \times A_N^e$ where for each $j = 1, \ldots, N$, the set $A_j^e \subseteq A_j$.

The **normal form of the empty edit** is $\emptyset \times \cdots \times \emptyset$, with $\emptyset$ repeated $N$ times. The other representations of the empty edit as products are not considered to be of normal form.

Instead of the term “an edit of normal form” we also use the term **normal edit**.

If $X \subseteq D$ then we write $\mathcal{N}(X)$ to mean the set of normal edits that are subsets of $X$. The set $\mathcal{N}(X)$ is a subset of the power set $\mathcal{P}(X)$.

**Notation.** For a normal edit $e$, we will use the notation of Definition 2.2.12. That is

$e = A_1^e \times \cdots \times A_N^e$ where for each $j = 1, \ldots, N$, the set $A_j^e \subseteq A_j$.

Note: Later on, we will see that the most useful elements of $\mathcal{N}(X)$ are the maximal normal edits contained in $X$.

**Example 2.2.13.** The edits $e_1$ and $e_2$ are presented in normal form in Example 2.2.11. An example of a non-normal edit is $\{(6, yes, 8), (7, no, 12)\}$, which cannot be written as a product of subsets of the fields.

In general we will assume that all categorical edits are in normal form, because the next proposition tells us that each categorical edit can be broken down as a union of a set of normal edits.
**The covering set method**

**Proposition 2.2.14.** If \( E \) is a set of edits (some of which might not be normal) then there exists a set \( E' \) of normal edits such that, for each record \( v \), the record \( v \) fails \( E \) if and only if \( v \) fails \( E' \).

**Proof.** At worst \( E' = \{ \{a_1\} \times \cdots \times \{a_N\} \mid (a_1, \ldots, a_N) \text{ is an element of some edit in } E \} \).

We will occasionally use the following property of normal edits.

**Lemma 2.2.15.** If \( e_1 \) and \( e_2 \) are normal edits with \( e_1 \subseteq e_2 \), then for each field \( j \), we have that \( A^{e_1}_j \subseteq A^{e_2}_j \).

**Proof.** If \( e_1 \neq \emptyset \), then the result follows from a property of sets. If \( e_1 = \emptyset \), then the result follows from the definition of the normal form for the empty edit as the product of \( N \) empty sets.

---

**2.3 Solving the error localisation problem using covering sets**

This section gives a formal definition of covering sets (Definition 2.3.3) and of the covering set method (Definition 2.3.12). The covering set method depends on the definition of edit generation functions (Definition 2.3.13) which can have various properties: covering set correctibility (Definition 2.3.14), error correction guarantee (Definition 2.3.16), error correction totality (Definition 2.3.16), and smallest weighted covering set correctibility (Definition 2.3.19).

The covering set method depends on the following observations.

**Observation 1.** Suppose that the record \( v \) fails the normal edit \( e \). Then any correction to \( v \) with respect to \( \{e\} \) must change at least one field \( j \) for which \( A^e_j \) is a strict subset of \( A_j \), because otherwise the change in field \( j \) cannot change \( v \)'s correctness with respect to \( e \).

**Observation 2.** Hence if \( v \) fails a set \( E \) of normal edits, then any correction to \( v \) changes a set \( C \) of fields where to each failed edit \( e \) in \( E \) there is a field \( j \) in \( C \) such that \( A^e_j \) is a strict subset of \( A_j \). Such a set \( C \) is a solution to the error localisation problem for \( v \) and \( E \).

**Example 2.3.1.** As in the previous examples, suppose that

\[
D = A_{\text{age}} \times A_{\text{driver}} \times A_{\text{grade}},
\]

\[
EE = \{e_1, e_2\}, \text{ where}
\]

\[
e_1 = A_{\text{age}} \times \{\text{yes}\} \times \{1, 2, \ldots, 10\},
\]

\[
e_2 = \{5, 6, \ldots, 9\} \times A_{\text{driver}} \times \{7, 8, \ldots, 12\},
\]

\[
vv = (6, \text{yes}, 8), \text{ which fails both edits } e_1 \text{ and } e_2, \text{ and}
\]

\[
ww = (6, \text{no}, 1), \text{ which corrects } vv \text{ with respect to the edit set } EE.
\]
The correction \(ww\) changes the values of the fields of \(vv\) in the set \{driver, grade\}. Thus Observation 2 applies to \(EE, vv\) and \(ww\) because:

for edit \(e_1\) of \(EE\), the sets \(A^{e_1}_{\text{driver}} = \{\text{yes}\}\) and \(A^{e_1}_{\text{grade}} = \{1, 2, \ldots, 10\}\) are strict subsets of \(A_{\text{driver}}\) and \(A_{\text{grade}}\) respectively; and

for edit \(e_2\) of \(EE\), the set \(A^{e_2}_{\text{grade}} = \{7, 8, \ldots, 12\}\) is a strict subset of \(A_{\text{grade}}\).

We call the set \(C\) of Observation 2 a “covering set of the edits of \(E\) failed by \(v\)” and we say that a field \(j\) with the property that \(A^e_j\) is a strict subset of \(A_j\) is “involved in \(e\)”. More formally:

**Definition 2.3.2.** The field \(j\) is involved or entering in edit \(e = A^1_i \times \cdots \times A^N_N\) if \(A^e_j \neq A_j\). An uninvolv\(e\)d or non-entering field \(j\) has \(A^e_j = A_j\) and is also referred to as an essential field.

Note: Every field is involved in the empty edit.

**Definition 2.3.3.** A covering set \(C\) of the set \(E\) of edits is a set of fields such that, for each edit \(e\) in \(E\), there exists a field in \(C\) that is involved in \(e\). The set of all covering sets of \(E\) is written \(C(E)\):

\[
C(E) = \{C \subseteq \{1, \ldots, N\} \mid C \text{ is a covering set of } E\}.
\]

That is, a covering set of the edit set \(E\) is a set of fields that includes an involved field of each edit in \(E\).

Note 1: If \(C \in C(E)\) then every superset of \(C\) is also in \(C(E)\).

Note 2: \(E = \emptyset\) if and only if \(\emptyset \in C(E)\). Hence, by Note 1, any set of fields covers the edit set \(\emptyset\).

Note 3: \(D \in E\) if and only if \(C(E) = \emptyset\).

**Example 2.3.4.** In the previous examples,

the edit \(e_1\) involves the fields \(\text{driver}\) and \(\text{grade}\);

the edit \(e_2\) involves the fields \(\text{age}\) and \(\text{grade}\).

The two edits are covered by the singleton field set \{\text{grade}\} and also by any field set that includes \text{grade}; they are also covered by the doubleton field set \{\text{age}, \text{driver}\}.

We are especially interested in finding a covering set of the edits of \(E\) failed by \(v\), and so the following definition will be useful.

**Definition 2.3.5.** Define the set \(X(E, v)\) to be the set of edits in the set \(E\) that are failed by \(v\). That is, \(X(E, v) = \{e \in E \mid v \in e\}\).

By Definitions 2.3.3 and 2.3.5, the set of covering sets of the edits of \(E\) failed by \(v\) is \(C(X(E, v))\). Since we will be using the combination of \(C\) and \(X\) often, we will simplify notation a little, as follows.

**Definition 2.3.6.** Define \(CX(E, v)\) to be the set of covering sets of the edits in the set \(E\) that are failed by the record \(v\). That is \(CX(E, v) = C(X(E, v))\).
Example 2.3.7. In the previous examples, we found that $vv$ fails both edits of $EE$. Hence $\mathcal{X}(EE, vv) = EE$. From Example 2.3.4, we have that

$$\mathcal{C}X(EE, vv) = \{ \{ \text{grade} \}, \{ \text{age}, \text{driver} \}, \{ \text{age}, \text{grade} \}, \{ \text{driver}, \text{grade} \}, \{ \text{age}, \text{driver}, \text{grade} \} \}.$$ 

Although every error localisation solution is a covering set of the failed edits, we saw in Chapter 1 that the converse need not apply. That is, there is no guarantee that every covering set of the failed edits is an error localisation solution, and so the covering sets do not solve the error localisation problem. In terms of notation:

$$\text{although } \mathcal{E}L(E, v) \subseteq \mathcal{C}X(E, v), \quad (2.3.1)$$

there is no guarantee that $\mathcal{E}L(E, v) = \mathcal{C}X(E, v)$. \quad (2.3.2)

Example 2.3.8. Defining the edit set $EE$ and the record $vv$ as in the previous examples, we see from Examples 2.2.9 and 2.3.7 that $\mathcal{C}X(EE, vv) \neq \mathcal{E}L(EE, vv)$. In particular, $\{ \text{grade} \} \in \mathcal{C}X(EE, vv)$, but $\{ \text{grade} \} \notin \mathcal{E}L(EE, vv)$.

As observed in Chapter 1, the reason that $\mathcal{E}L(E, v)$ need not equal $\mathcal{C}X(E, v)$ is that there can be additional edits implied by $E$ which are not covered by every element of $\mathcal{C}X(E, v)$ but which are failed by $v$.

Example 2.3.9. As seen in Example 2.2.4, the edits

$$e_1 = A_{\text{age}} \times \{ \text{yes} \} \times \{ 1, 2, \ldots, 10 \} \quad \text{and}$$

$$e_2 = \{ 5, 6, \ldots, 9 \} \times A_{\text{driver}} \times \{ 7, 8, \ldots, 12 \}$$

correspond to the requirements

Requirement 1: A person with a driver’s licence must be in at least Grade 11; and

Requirement 2: A person in Grade 7 or higher must be at least 10 years old,

which imply another requirement:

Requirement 3: A person with a driver’s licence must be at least 10 years old. (Of course, in reality, something much stronger than this holds, but this requirement is a logical implication of the first two requirements.)

Requirement 3 corresponds to the edit

$$e_3 = \{ 5, 6, \ldots, 9 \} \times \{ \text{yes} \} \times A_{\text{grade}}.$$ 

Although the field set $\{ \text{grade} \}$ covers the edit set $\{ e_1, e_2 \}$, it does not cover the implied edit set $\{ e_1, e_2, e_3 \}$.

Although Statement 2.3.2 tells us that in general $\mathcal{E}L(E, v) \neq \mathcal{C}X(E, v)$, Fellegi and Holt (1973, 1976) used the idea of implied edits to find a similar, but correct,
relationship. They constructed, for each edit set $E$, an edit set $FH(E)$ such that the covering sets of the failed edits of the new set $FH(E)$ are exactly the error localisation solutions. In other words, the incorrect equation (in Statement 2.3.2) can be modified to give the correct equation

$$\mathcal{EL}(E, v) = \mathcal{CX}(FH(E), v).$$

Fellegi and Holt’s construction for $FH(E)$ will be given in the next section, but the following example gives $FH(EE)$ for the edit set $EE$ of the examples.

**Example 2.3.10.** Suppose that the edit set $EE = \{e_1, e_2\}$ is as defined in the previous examples. Then $FH(EE) = \{e_1, e_2, \ldots, e_8\}$, where $e_1$, $e_2$, and $e_3$ are as given in Example 2.3.9, and

- $e_4 = \{age\} \times \{yes\} \times \{7, 8, 9, 10\},$
- $e_5 = \{5, 6, \ldots, 9\} \times \{yes\} \times \{7, 8, 9, 10\},$
- $e_6 = \{5, 6, \ldots, 9\} \times A_{driver} \times \{7, 8, 9, 10\},$
- $e_7 = \{5, 6, \ldots, 9\} \times \{yes\} \times \{1, 2, \ldots, 10\},$
- $e_8 = \{5, 6, \ldots, 9\} \times \{yes\} \times \{7, 8, \ldots, 12\}.$

Then $\mathcal{CX}(FH(EE), vv) = \mathcal{EL}(EE, vv) = \{\{age, driver\}, \{age, grade\}, \{driver, grade\}, \{age, driver, grade\}\}.$

In the above example, we started with the edits $e_1$ and $e_2$, from which we derived the remaining edits $e_3, \ldots, e_8$. We will call the starting set of edits “explicit edits”:

**Definition 2.3.11.** The starting set of edits, obtained from domain knowledge, is called the set of explicit edits, to distinguish them from implied edits.

We will use the term “covering set method” to refer to the technique of using covering sets for error localisation, and define it precisely below.

**Definition 2.3.12.** The covering set method is a method of trying to solve the error localisation problem. Given a normal edit set $E$, the method consists of the following steps:

1. Calculate a normal edit set $G(E)$ derived from the set $E$, where $G$ is some predefined function.

2. For any record $v$ that fails $E$, calculate the field set $\mathcal{CX}(G(E), v)$ of the covering sets of the edits of $G(E)$ failed by $v$.

Note that the covering set method does not necessarily work, that is, the calculated field set $\mathcal{CX}(G(E), v)$ need not be equal to the set of error localisation solutions. For example if $G(E) = E$ then, as seen in Example 2.3.8, the covering set method need not work. On the other hand, if $G(E) = FH(E)$ then we will see in the next section that the covering set method does work, and that the set $\mathcal{CX}(FH(E), v)$ is indeed the set of error localisation solutions.
The covering set method is often referred to as the Fellegi-Holt method, although sometimes the Fellegi-Holt method refers explicitly to using the covering set method with the particular function \( FH \). Occasionally the term “Fellegi-Holt method” is also used for some or all of the criteria specified by Fellegi and Holt (1973, 1976), some of which have been alluded to previously:

1. The (weighted) minimum number of fields should be changed (where the word “weighted” was added by Garfinkel (1979) and Liepins (1980b)) - see also page 6 of Chapter 1.

2. Imputation rules should derive automatically from the edit rules.

3. Imputation should try to maintain the marginal and preferably the joint frequency distributions of variables - see also page 6 of Chapter 1.

The terms “Fellegi-Holt paradigm” and “Fellegi-Holt principle” are also used for one or more of the above criteria.

The function \( FH \), and the function \( G \) used in Definition 2.3.12, are described as “normal edit generation functions”, defined next.

**Definition 2.3.13.** An edit generation function is a function from edit sets to edit sets, that is, a function \( G \) where

\[
G : \mathcal{P}(\mathcal{P}(D)) \rightarrow \mathcal{P}(\mathcal{P}(D)).
\]

A normal edit generation function is a function from normal edit sets to normal edit sets, that is, a function \( G \) where

\[
G : \mathcal{P}(\mathcal{N}(D)) \rightarrow \mathcal{P}(\mathcal{N}(D)),
\]

and \( \mathcal{N}(D) \) is the set of normal edits in \( D \), as defined in Definition 2.2.12.

In general we will assume that all categorical edit generation functions are normal unless specifically stated otherwise.

Instead of saying the long expression that “the covering sets of the failed edits of \( FH(E) \) are the error localisation solutions”, we will say that the function \( FH \) is “covering set correctible”, defined precisely by:

**Definition 2.3.14.** The normal edit generation function \( G \) is covering set correctible if for each edit set \( E \) and for each record \( v \), we have that \( CX(G(E), v) = EL(E, v) \).

**Example 2.3.15.** In Example 2.3.10, we saw a demonstration of the covering set correctibility of \( FH \), because \( CX(FH(EE), vv) = EL(EE, vv) \).

There are edit generation functions other than \( FH \) that are covering set correctible. In Sections 2.5 and 2.6, we will consider four such functions, which we call ENFH, MFH, MENFH and FCF\(_{\omega} \).

For convenience we will split covering set correctibility into two components, error correction guarantee and error correction totality, as in the next definition:
Definition 2.3.16. The edit generation function $G$ has the error correction guarantee if, for each edit set $E$ and each record $v$, we have that $\mathcal{CL}(G(E), v) \subseteq \mathcal{EL}(E, v)$.

The edit generation function $G$ has error correction totality if, for each edit set $E$ and each record $v$, we have that $\mathcal{CL}(G(E), v) \supseteq \mathcal{EL}(E, v)$.

In practice one seeks, not any covering set, but a “best” covering set, and so we define “smallest weighted covering set”:

Definition 2.3.17. A smallest weighted covering set $C$ of the set $E$ of normal edits, with a positive $N$-tuple $(b_1, \ldots, b_N)$ as weights, is a covering set of $E$ such that:

if $C'$ is also a covering set of $E$ then $\sum_{i \in C'} b_i \geq \sum_{i \in C} b_i$.

As we did for covering sets, we will use a notation for smallest weighted covering sets:

Definition 2.3.18. The set of smallest weighted covering sets of the set $E$, using the $N$-tuple of weights $(b_1, \ldots, b_N) = b$, will be written $\mathcal{SC}(E, b)$.

The set of the smallest weighted covering sets of the edits of $E$ failed by $v$, using the weights $(b_1, \ldots, b_N) = b$, will be written $\mathcal{SCX}(E, v, b)$. That is

$\mathcal{SCX}(E, v, b) = \mathcal{SC}(\mathcal{X}(E, v, b))$.

Hence if $C \in \mathcal{SC}(E, b)$, where $b = (b_1, \ldots, b_N)$, and $C' \in C(E)$, then $\sum_{i \in C'} b_i \geq \sum_{i \in C} b_i$.

We would like the smallest weighted covering sets to be exactly the smallest weighted error localisation solutions. Hence we define the notion of “smallest covering set correctible”, which is related to the notion of “covering set correctible”.

Definition 2.3.19. The edit generation function $G$ is smallest weighted covering set correctible (for the weights $b$) if for any edit set $E$ and record $v$, we have that $\mathcal{SCX}(G(E), v, b) = \mathcal{SEL}(E, v, b)$, where $\mathcal{SEL}(E, v, b)$ is the set of solutions to the smallest weighted error localisation problem for the record $v$ with respect to the edit set $E$ and weights vector $b$, as defined in Definition 2.2.10.

Smallest weighted covering set correctibility for all possible weights is equivalent to covering set correctibility. The proof is given below in Proposition 2.3.20.

However, if an edit generation function is covering set correctible for some but not all weights, then it need not be covering set correctible. We will give an example later, in Section 2.5, Example 2.5.16, after having defined the function $FH$ and related functions.

Proposition 2.3.20. The edit generation function $G$ is covering set correctible if and only if, for each positive $N$-tuple $b$ of weights, the function $G$ is smallest weighted covering set correctible for $b$. 

Proof.
Forward direction: immediate.
Backward direction: Suppose that, for each sequence \( b \) of \( N \) weights, the function \( G \) is smallest weighted covering set correctible for \( b \). That is, given a record \( v \), an edit set \( E \), and a positive \( N \)-tuple \( b \), then \( SCX(G(E), v, b) = E\mathcal{L}(E, v, b) \). We will show that \( C\mathcal{X}(G(E), v) = \mathcal{E}\mathcal{L}(E, v) \).

Firstly, we show that \( C\mathcal{X}(G(E), v) \subseteq \mathcal{E}\mathcal{L}(E, v) \).
Let \( C \in C\mathcal{X}(G(E), v) \). We will show that \( C \in \mathcal{E}\mathcal{L}(E, v) \).
Define the \( N \)-tuple \( d = (d_1, \ldots, d_N) \) by:

\[
d_i = \begin{cases} 1 & i \in C \\ N & i \notin C \end{cases}
\]

Let \( C' \in SCX(G(E), v, d) \).
Then \( C' \in S\mathcal{E}\mathcal{L}(E, v, d) \), since \( G \) is smallest weighted covering set correctible.
Then \( C' \in \mathcal{E}\mathcal{L}(E, v) \), by the definition of “smallest weighted covering set correctible” and “covering set correctible”.
Also \( C' \subseteq C \), because, if not

\[
\sum_{i \in C'} d_i = |C \cap C'| + N|C' \setminus C|, \quad \text{by the definition of } d
\]

\[
\geq N, \quad \text{since } C' \nsubseteq C \Rightarrow C' \setminus C \neq \emptyset
\]

\[
> |C|, \quad \text{since } C' \setminus C \neq \emptyset \Rightarrow |C| < N
\]

\[
= \sum_{i \in C} d_i, \quad \text{by the definition of } d,
\]

which contradicts the fact that \( C' \in SCX(G(E), v, d) \).

Hence \( C' \subseteq C \).
Hence \( C \in \mathcal{E}\mathcal{L}(E, v) \), since \( C' \in \mathcal{E}\mathcal{L}(E, v) \) and \( C' \subseteq C \).
Hence \( C\mathcal{X}(G(E), v) \subseteq \mathcal{E}\mathcal{L}(E, v) \).

Secondly, to show that \( \mathcal{E}\mathcal{L}(E, v) \subseteq C\mathcal{X}(G(E), v) \), use the same argument as above except with \( \mathcal{E}\mathcal{L}(E, v) \) swapped with \( C\mathcal{X}(G(E), v) \), and \( S\mathcal{E}\mathcal{L}(E, v, d) \) swapped with \( SCX(G(E), v, d) \).

This section has outlined the covering set method and relevant properties for categorical edits. A similar method and properties apply to arithmetic edits expressed as linear inequalities. We leave the details to Chapter 7, but note here that the ideas were introduced by Fellegi and Holt (1976), and have been applied by Greenberg (1981), Greenberg and Surdi (1984), Winkler and Draper (1996), and Garcia and Goodwin (2002).
Before concluding this section, we summarise its main points. One method of error correction is to first do error localisation by the use of covering sets. For a given edit set \( E \), a given record \( v \), and a chosen edit generation function \( G \), we find the covering sets of the edits of \( G(E) \) failed by \( v \) - that is, we find the covering sets \( \mathcal{C}\mathcal{X}(G(E), v) \). If, for each \( E \) and \( v \), the covering sets \( \mathcal{C}\mathcal{X}(G(E), v) \) are exactly the same as the error localisation solutions \( \mathcal{E}\mathcal{L}(E, v) \), then we say that \( G \) is covering set correctible. Indeed, we would prefer that \( G \) is smallest weighted covering set correctible: it turns out that smallest weighted covering set correctibility for all weights is equivalent to covering set correctibility. The challenge is to find functions \( G \) that are covering set correctible.

One such function is \( \text{FH} \), which we will define in the next section, where we also prove its covering set correctibility. Other such functions are \( \text{ENFH}, \text{MFH}, \text{MENFH} \) and \( \text{FCF}_\omega \), to be considered in the subsequent two sections.

### 2.4 The function FH

In this section, we define the edit generation function \( \text{FH} \) and prove that it is covering set correctible. In later sections, we will use \( \text{FH} \) to define other covering set correctible functions.

The function \( \text{FH} \) depends on another function, \( \text{FHG} \), of edit sets.

**Definition 2.4.1.** (Fellegi and Holt 1973, 1976) Let \( E \) be a set of normal edits, where each edit \( e \) in \( E \) is written \( A_1^e \times \cdots \times A_N^e \). Then the \( \text{FH-generated edit} \) on \( E \) with generating field \( i \) is

\[
\text{FHG}(i, E) = \prod_{j=1}^{i-1} \bigcap_{e \in E} A_j^e \times \bigcup_{e \in E} A_i^e \times \prod_{j=i+1}^{N} \bigcap_{e \in E} A_j^e, \tag{2.4.1}
\]

and the function \( \text{FHG} : \{1, \ldots, N\} \times \mathcal{P}(\mathcal{N}(D)) \longrightarrow \mathcal{P}(D) \), where \( \mathcal{N}(D) \) is defined in Definition 2.2.12 to be the set of normal edits in \( D \).

Note 1: The above expression is the normal form for \( \text{FHG}(i, E) \), except when \( \text{FHG}(i, E) \) is empty. When \( \text{FHG}(i, E) \) is empty, all components of its normal form are defined by Definition 2.2.12 to be empty, that is for all \( j \), the set \( A_j^{\text{FHG}(i,E)} = \emptyset \), which is not necessarily the component calculated in the above expression.

Note 2: \( \text{FHG}(i, \emptyset) = \emptyset \).

Note 3: If \( e \) is a normal edit then \( \text{FHG}(i, \{e\}) = e \).

**Example 2.4.2.** Suppose, as in the previous examples, that

\[
\begin{align*}
D &= A_{\text{age}} \times A_{\text{driver}} \times A_{\text{grade}}, \\
\epsilon_1 &= A_{\text{age}} \times \{\text{yes}\} \times \{1, 2, \ldots, 10\}, \\
\epsilon_2 &= \{5, 6, \ldots, 9\} \times A_{\text{driver}} \times \{7, 8, \ldots, 12\}.
\end{align*}
\]
Then the edits $e_3, \ldots, e_8$ introduced in Examples 2.3.9 and 2.3.10 are constructed as follows:

\[
\begin{align*}
e_3 &= \text{FHG}(\text{grade}, \{e_1, e_2\}) = \{5,6,\ldots,9\} \times \{\text{yes}\} \times A_{\text{grade}}, \\
e_4 &= \text{FHG}(\text{age}, \{e_1, e_2\}) = A_{\text{age}} \times \{\text{yes}\} \times \{7,8,9,10\}, \\
e_5 &= \text{FHG}(\text{driver}, \{e_1, e_2\}) = \{5,6,\ldots,9\} \times A_{\text{driver}} \times \{7,8,9,10\}, \\
e_6 &= \text{FHG}(\text{grade}, \{e_4, e_5\}) = \{5,6,\ldots,9\} \times \{\text{yes}\} \times \{7,8,\ldots,12\}, \\
e_7 &= \text{FHG}(\text{driver}, \{e_1, e_3\}) = \{5,6,\ldots,9\} \times \{\text{yes}\} \times \{1,2,\ldots,10\}, \\
e_8 &= \text{FHG}(\text{age}, \{e_2, e_3\}) = \{5,6,\ldots,9\} \times \{\text{yes}\} \times \{7,8,\ldots,12\}.
\end{align*}
\]

We now give the definition of Fellegi and Holt (1973, 1976) for the edit generation function $\text{FH} : \mathcal{P} \left( \mathcal{N}(D) \right) \rightarrow \mathcal{P} \left( \mathcal{N}(D) \right)$, where $\mathcal{N}(D)$ is defined in Definition 2.2.12 to be the set of normal edits in $D$. Given a normal edit set $E$, the edit set $\text{FH}(E)$ is defined by the following process. A field $i$ and a subset $X$ of $E$ are chosen such that $\text{FHG}(i, X)$ is not already in $E$. The edit $\text{FHG}(i, X)$ is then added to $E$ and the process repeated until no new edits can be generated. The process will eventually terminate because the domain is finite. The set $\text{FH}(E)$ is the edit set generated by the process.

A more formal inductive definition is:

**Definition 2.4.3.** ( Fellegi and Holt 1973, 1976) The FH edit generation function

\[
\text{FH} : \mathcal{P} \left( \mathcal{N}(D) \right) \rightarrow \mathcal{P} \left( \mathcal{N}(D) \right)
\]

is defined inductively for each set $E$ of normal edits as follows:

1. $E \subseteq \text{FH}(E)$;
2. if $X \subseteq \text{FH}(E)$ and $i$ is a field, then $\text{FHG}(i, X) \in \text{FH}(E)$.

Note: $\text{FH}(\emptyset) = \{\emptyset\}$.

An equivalent definition of the function $\text{FH}$ is in terms of sequences of normal edits, as stated in the next proposition.

**Proposition 2.4.4.** Let $E$ be a set of normal edits and let $e$ be a normal edit. Then $e \in \text{FH}(E)$ if and only if there exists a finite sequence of normal edits $e_1, \ldots, e_n = e$ such that for each $j = 1, \ldots, n$

1. $e_j \in E$, or
2. there is a subset $X$ of $\{e_1, \ldots, e_{j-1}\}$ and a field $i$ such that $e_j = \text{FHG}(i, X)$.

**Proof.** Apply Theorem 1.9.7 with:

1. $S = \text{the set of normal edits } \mathcal{N}(D)$;
2. $B = E$; and
3. the relation $R$ on $\mathcal{P} \left( \mathcal{N}(D) \right) \times \mathcal{N}(D)$ defined as follows: $(X, e) \in R$ if and only if there is a field $i$ such that $e = \text{FHG}(i, X)$. 

\[\square\]
We now turn to the proof that FH is covering set correctible. We will deal separately with the two components, error correction guarantee and error correction totality. We deal first with the more difficult component, the error correction guarantee, in Theorem 2.4.6 below, due to Fellegi and Holt (1973, 1976).

The main idea of the proof of the error correction guarantee of FH is the construction of a sequence of edit sets with a property called the “lifting property”, which is the same as the “lifting principle” of de Waal (2003a, page 41).

**Definition 2.4.5.** A sequence of edit sets \((X_j)_{j=a}^{b}\), where \(0 \leq a < b \leq N\), has the lifting property for fields \(i_a, \ldots, i_b\) if the following holds:

- If \(v\) is a record and \(a + 1 \leq k \leq b\) and \(v\) satisfies all edits in \(X_k\) then \(\{i_k\} \in \mathcal{EL}(X_{k-1}, v)\).

(That is, if \(v\) satisfies all edits in \(X_k\) then the field \(k\) of \(v\) can be changed so that the new record satisfies all edits in \(X_{k-1}\).)

**Theorem 2.4.6 (Error correction guarantee of FH).** The edit generation function FH has the error correction guarantee. (Fellegi and Holt (1973, 1976, Theorem 1, Corollaries 1, 2) and Liepins (1980b, Lemma 4, Corollaries 1, 2).)

**Outline of proof.** For a given set \(E\) of explicit edits, Fellegi and Holt’s proof depends on a sequence of edit sets \(\Omega_N, \ldots, \Omega_0 = FH(E)\), where each set \(\Omega_k\) contains only those edits that are in \(FH(E)\) and in which the fields \(1, \ldots, k\) are uninvolved\(^1\). That is

\[ \Omega_k = \{ e \in FH(E) \mid \text{fields } 1, \ldots, k \text{ are uninvolved in } e \}. \]

The proof is in three steps: the Lifting Property Theorem for \((\Omega_k)_{k=0}^{N}\) (F & H Theorem 1), the Repeated Lifting Corollary (F & H Corollary 1) and, finally, the Error Correction Guarantee Corollary (F & H Corollary 2). The details are below but I first give the ideas. The Lifting Property Theorem for \((\Omega_k)_{k=0}^{N}\) ensures that, so long as the record satisfies \(\Omega_k\) for some \(k\), it can be “improved” by changing the field \(k\) so as to satisfy \(\Omega_{k-1}\). Under the Repeated Lifting Corollary, if the record satisfies \(\Omega_k\) for some \(k\) then it can be successively “improved” to satisfy \(\Omega_k, \ldots, \Omega_0\), which is the set \(FH(E)\) of all edits generated by the Fellegi-Holt edit generation process. The proof of the Error Correction Guarantee Corollary finds an appropriate renumbering of the fields so that the record satisfies some \(\Omega_k\). Then, by the Repeated Lifting Corollary, the record can be successively improved to satisfy \(FH(E)\).

The proof of F & H Theorem 1 (Lifting Property Theorem) works by contradiction. First assume \((\Omega_k)_{k=0}^{N}\) does not have the lifting property. That is, assume that some record \(v\) satisfies all the edits in some \(\Omega_k\) but \(\{k\} \notin \mathcal{EL}(\Omega_{k-1}, v)\). This means that, no matter how the field \(k\) of \(v\) is changed, the new record will fail some edit in \(\Omega_{k-1}\). Hence, if the field \(k\) of record \(v\) is changed to the value \(a\) of field domain \(A_k\), then the new record \(v^a\) will fail some edit \(e^a\) in \(\Omega_{k-1}\). The set \(S = \{e^a \mid a \in A_k\}\) of such edits is

---

\(^1\)Note that I have switched around the definition of \(\Omega_k\) from that used by Fellegi and Holt, with a correspondingly changed result. Whereas I define the edits of \(\Omega_k\) to have fields \(1, \ldots, k\) uninvolved, Fellegi and Holt define the edits of \(\Omega_k\) to have fields \(k+1, \ldots, N\) uninvolved.
then used to generate a new edit $\gamma = \text{FHG}(k, S)$. A calculation of the new edit $\gamma$ shows that $\gamma \in \Omega_k$ and hence $\gamma$ is satisfied by $v$, since we assumed that $v$ satisfies everything in $\Omega_k$. Yet the calculation of $\gamma$ also shows that $\gamma$ is failed by $v$. So $\{k\} \in \mathcal{EL}(\Omega_{k-1}, v)$, after all.

F & H Corollary 1 (Repeated Lifting Corollary) states that if $v$ satisfies $\Omega_k$ then $\{1, \ldots, k\} \in \mathcal{EL}(\text{FH}(E), v)$. The proof is by repeated application of the Lifting Property Theorem, noting that $\Omega_0 = \text{FH}(E)$.

F & H Corollary 2 (Error Correction Guarantee Corollary) states that FH has the error correction guarantee. To prove it, suppose that $C \in \mathcal{C}(\text{FH}(E), v)$. Renumber the fields so that $C = \{1, \ldots, k\}$ where $k \geq 0$. Then $v$ satisfies $\Omega_k$ and, by the Repeated Lifting Corollary, $C \in \mathcal{EL}(\text{FH}(E), v)$. Since $\text{FH}(E) \supseteq E$, we have that $C \in \mathcal{EL}(E, v)$.

The error correction totality of FH is given below in Proposition 2.4.9. Although not explicitly proved by Fellegi and Holt, the error correction totality of FH does follow quickly from the following lemma of Fellegi and Holt, and its corollary.

**Lemma 2.4.7.** (Fellegi and Holt 1973, 1976) Let $E$ be a set of normal edits, $i$ be a field, and $v$ be a record that satisfies all edits in $E$. Then $v$ satisfies the edit $\text{FHG}(i, E)$.

**Outline of proof.** Prove the contrapositive, as follows. Given an edit set $E$ and a record $v$, suppose that $v \in \text{FHG}(i, E)$. Then by the construction of $\text{FHG}(i, E)$, the record $v$ is in some edit $e$ of $E$.

The next corollary directly extends the above lemma to the function FH. In the later logical formalisation of Chapter 3, the following property will be known as “soundness”.

**Corollary 2.4.8 ("Soundness" of FH).** Let $E$ be a set of normal edits, and $v$ be a record that satisfies all edits in $E$. Then $v$ satisfies all edits in $\text{FH}(E)$.

We can now prove that FH has error correction totality.

**Proposition 2.4.9 (Error correction totality of FH).** The edit generation function FH has error correction totality.

**Proof.** Let $E$ be an edit set and let $v = (v_1, \ldots, v_N)$ be a record. We will show that $\mathcal{EL}(E, v) \subseteq \mathcal{C}(\text{FH}(E), v)$, by supposing that $C \in \mathcal{EL}(E, \mathcal{v})$ and showing that $C \in \mathcal{C}(\text{FH}(E), v)$. We will do this by showing that for each edit $e$ in $\mathcal{C}(\text{FH}(E), v)$ there is a field in $C$ that is involved in $e$.

We first find a field $k$ that is involved in $e$, as follows. Since $C \in \mathcal{EL}(E, v)$, then by Definition 2.2.8, there is a record $w = (w_1, \ldots, w_N)$ that satisfies $E$ and such that $C \supseteq \{j \mid w_j \neq v_j\}$. Then by Corollary 2.4.8, the record $w$ satisfies all edits in $\text{FH}(E)$, including the edit $e$, ie $w \notin e$. Hence there is a field $k$ such that $w_k \notin A_k$, so that $A_k \neq A_k$. Hence $k$ is involved in $e$.

We now show that the field $k$ is in $C$, as follows. Since $e \in \mathcal{C}(\text{FH}(E), v)$ we have that $v \in e$, and hence $v_k \in A_k$. Since $w_k \notin A_k$, we have that $w_k \neq v_k$, ie $k \in \{j \mid w_j \neq v_j\} \subseteq C$, from the properties of $w$ given above.
§2.5  The functions ENFH, MFH and MENFH

This section introduces three more edit generation functions, ENFH, MFH, and MENFH, each of which is covering set correctible. Each of the three functions is a different sub-function of FH, where we defined subfunction in Chapter 1, Definition 1.9.3.

We start with the definitions of the three functions. Firstly, the function ENFH depends on a special type of FH-generated edit called an essentially new FH-generated edit:

**Definition 2.5.1.** *(based on Fellegi and Holt 1973, 1976.)* Let $i$ be a field, and let $E$ be a set of normal edits, where each edit $e$ in $E$ is written $A^e_1 \times \cdots \times A^e_N$. Then
The covering set method

FHG\((i, E)\) is called an **essentially new FH-generated edit** if \(A^\text{FHG}(i, E) = A_i\). We also use the shorter term **essentially new edit**.

Note 1: This means that each essentially new edit is non-empty.
Note 2: I have simplified Fellegi and Holt’s definition of “essentially new implied edits”, which included an additional property. The additional property was in terms of dominated edits, defined in Definition 2.2.5. In their definition of essentially new implied edits, Fellegi and Holt required, for each edit \(e\) of \(E\), that no generating edits dominate the generated edit. That is, in Definition 2.5.1, they also required, for each edit \(e\) of \(E\), that \(A^e_i \subset A_i\). In this thesis, I do not impose this requirement on essentially new edits.

The function \(\text{ENFH}\) is defined inductively in the same way as \(\text{FH}\), except that it depends on essentially new edits.

**Definition 2.5.2.** The **essentially new FH edit generation function**, written \(\text{ENFH}\), is the normal edit generation function

\[
\text{ENFH} : \mathcal{P}(\mathcal{N}(D)) \rightarrow \mathcal{P}(\mathcal{N}(D))
\]
defined inductively for each set \(E\) of normal edits by:

1. \(E \subseteq \text{ENFH}(E)\);
2. if \(X \subseteq \text{ENFH}(E)\), and \(i\) is a field, and \(\text{FHG}(i, X)\) is essentially new, then \(\text{FHG}(i, X) \in \text{ENFH}(E)\).

**Example 2.5.3.** Suppose, as in the previous examples, that

\[
D = A_{\text{age}} \times A_{\text{driver}} \times A_{\text{grade}},
\]

\[
EE = \{e_1, e_2\}, \text{ where } e_1 = A_{\text{age}} \times \{\text{yes}\} \times \{1, 2, \ldots, 10\}, \text{ and } e_2 = \{5, 6, \ldots, 9\} \times A_{\text{driver}} \times \{7, 8, \ldots, 12\}.
\]

Then \(\text{ENFH}(EE) = \{e_1, e_2, e_3, e_4, e_5\}\), where \(e_3, e_4\) and \(e_5\) were constructed in Example 2.4.2 as:

\[
e_3 = \text{FHG}(\text{grade}, EE) = \{5, 6, \ldots, 9\} \times \{\text{yes}\} \times A_{\text{grade}},
\]

\[
e_4 = \text{FHG}(\text{age}, EE) = A_{\text{age}} \times \{\text{yes}\} \times \{7, 8, 9, 10\},
\]

\[
e_5 = \text{FHG}(\text{driver}, EE) = \{5, 6, \ldots, 9\} \times A_{\text{driver}} \times \{7, 8, 9, 10\}.
\]

The other two functions to be introduced in this section are \(\text{MFH}\) and \(\text{MENFH}\), defined below. The two functions depend on the idea, introduced by Liepins (1980b, Corollary 2, page 15), of considering edit generation functions that return only maximal edits. The two functions are defined as follows.
Definition 2.5.4. The maximal FH edit generation function, written MFH, is the normal edit generation function defined as follows. Given an edit set \( E \),

\[
\text{MFH}(E) = \text{Max}(\text{FH}(E)),
\]

where “Max” means taking the maximal elements with respect to subset inclusion, as in Definition 1.9.2.

Definition 2.5.5. The maximal essentially new FH edit generation function, written MENFH, is the normal edit generation function defined as follows. Given an edit set \( E \),

\[
\text{MENFH}(E) = \text{Max}(\text{ENFH}(E)),
\]

where “Max” means taking the maximal elements with respect to subset inclusion, as in Definition 1.9.2.

Note: Fellegi and Holt had already excluded some dominated edits in their definition of “essentially new”, but did not restrict the function to just maximal edits. Their definition of “essentially new” prevents a generated essentially new edit from being dominated by any of its generating edits.

Example 2.5.6. As in the previous examples, suppose that

\[
D = A_{\text{age}} \times A_{\text{driver}} \times A_{\text{grade}},
\]

\[
EE = \{e_1, e_2\}, \text{ where }
\]

\[
e_1 = A_{\text{age}} \times \{\text{yes}\} \times \{1, 2, \ldots, 10\}, \text{ and }
\]

\[
e_2 = \{5, 6, \ldots, 9\} \times A_{\text{driver}} \times \{7, 8, \ldots, 12\}.
\]

As given in Example 2.3.10, we have that \( \text{FH}(EE) = \{e_1, \ldots, e_8\} \), where

\[
e_3 = \{5, 6, \ldots, 9\} \times \{\text{yes}\} \times A_{\text{grade}},
\]

\[
e_4 = A_{\text{age}} \times \{\text{yes}\} \times \{7, 8, 9, 10\},
\]

\[
e_5 = \{5, 6, \ldots, 9\} \times A_{\text{driver}} \times \{7, 8, 9, 10\},
\]

\[
e_6 = \{5, 6, \ldots, 9\} \times \{\text{yes}\} \times \{7, 8, 9, 10\},
\]

\[
e_7 = \{5, 6, \ldots, 9\} \times \{\text{yes}\} \times \{1, 2, \ldots, 10\},
\]

\[
e_8 = \{5, 6, \ldots, 9\} \times \{\text{yes}\} \times \{7, 8, \ldots, 12\}.
\]

We then have that \( \text{MFH}(EE) = \text{MENFH}(EE) = \{e_1, e_2, e_3\} \).

Having defined the functions ENFH, MFH and MENFH, we now consider their covering set correctibility. We first consider their error correction totality, which follows from the next lemma and corollary. The lemma is about the connections between covering sets, the function \( X \), and subsets of edit sets; while its corollary is about the connection between subfunctions and error correction totality.

Lemma 2.5.7. Let \( E, E_1 \) and \( E_2 \) be sets of normal clauses and let \( v \) be a record. Then the following hold:
1. If $E_1 \subseteq E_2$, then $C(E_1) \supseteq C(E_2)$.

2. If $E_1 \subseteq E_2$, then $X(E_1, v) \subseteq X(E_2, v)$.

3. If $H$ is a subfunction of the edit generation function $G$, then $C(H(E), v) \supseteq C(G(E), v)$.

Proof.

1. Let $C \in C(E_2)$, and let $e \in E_1$. Since $E_1 \subseteq E_2$, we have that $e \in E_2$, and hence there is a field $i$ in $C$ that is involved in $e$. Hence $C$ covers $E_1$, as required.

2. Let $e \in X(E_1, v)$. Then $e \in E_1$ and hence $e \in E_2$. Also $v$ fails the edit $e$. Hence $e \in X(E_2, v)$.

3. Since $H$ is a subfunction of $G$, we have that $H(E) \subseteq G(E)$. Then by part 2 of this lemma, $X(H(E), v) \subseteq X(G(E), v)$, and hence the result follows from part 1 of this lemma.

Corollary 2.5.8. Each subfunction of an edit generation function with error correction totality also has error correction totality.

Proof. Let $G$ be an edit generation function with error correction totality. Let $H$ be a subfunction of $G$. Let $E$ be an edit set and let $v$ be a record. Since $H$ is a subfunction of $G$, then by Lemma 2.5.7 we have that $C(X(H(E), v) \supseteq C(X(G(E), v))$.

But since $G$ has error correction totality, we have that $C(X(G(E), v) \supseteq E(L(E), v)$, and thus $C(X(H(E), v) \supseteq E(L(E), v)$. Hence $H$ has error correction totality.

Corollary 2.5.9. The functions ENFH, MFH and MENFH have error correction totality.

Proof. The functions ENFH, MFH and MENFH are subfunctions of FH, which has error correction totality (by Proposition 2.4.9).

Having dealt with error correction totality, we are now ready to consider the second component of covering set correctness, namely the error correction guarantee.

But first, we note that Corollary 2.5.8, about subfunctions and error correction totality, has a dual, presented below as Corollary 2.5.10, about superfunctions and error correction guarantee. Although not relevant in this chapter, the corollary will be relevant in Chapter 6.

Corollary 2.5.10. Each superfunction of an edit generation function with the error correction guarantee also has the error correction guarantee.

Proof. Let $H$ be an edit generation function with the error correction guarantee. Let $G$ be a superfunction of $H$. Let $E$ be an edit set and let $v$ be a record. Since $H$ is a subfunction of $G$, then by Lemma 2.5.7 we have that $C(X(H(E), v) \supseteq C(X(G(E), v))$.

But since $H$ has the error correction guarantee, we have that $E(L(E), v) \supseteq E(L(H(E), v)$, and thus $E(L(E), v) \supseteq C(X(G(E), v))$. Hence $G$ has the error correction guarantee.
We now turn to the error correction guarantee of the three functions ENFH, MFH, and MENFH. We first consider the function ENFH. Although Fellegi and Holt did not explicitly prove that ENFH has the error correction guarantee, they did imply its proof because of its similarity with the proof of the error correction guarantee of FH.

**Proposition 2.5.11.** ENFH has the error correction guarantee.

*Proof.* Exactly the same proof as for FH, noting that all edits generated in the proof are essentially new. \(\square\)

We now consider the error correction guarantee of the two functions MFH and MENFH. Their error correction guarantee follows from the next lemma and corollary, which are duals of the above Lemma 2.5.7 and Corollary 2.5.8. The lemma is about the connections between covering sets, the function \(\mathcal{X}\), and superior sets (Definition 1.9.4 of Chapter 1); while its corollary is about superior functions (also defined in Definition 1.9.4).

**Lemma 2.5.12.** Let \(E, E_1\) and \(E_2\) be sets of normal clauses and let \(v\) be a record. Then the following hold:

1. If \(E_1\) is superior to \(E_2\), then \(C(E_1) \subseteq C(E_2)\).
2. If \(E_1\) is superior to \(E_2\), then \(\mathcal{X}(E_1, v)\) is superior to \(\mathcal{X}(E_2, v)\).
3. If \(H\) is a superior function to the edit generation function \(G\), then \(C\mathcal{X}(H(E), v) \subseteq C\mathcal{X}(G(E), v)\).

*Proof.*

1. Let \(C \in C(E_1)\), and let \(e_2 \in E_2\). Since \(E_1\) is superior to \(E_2\), there is an edit \(e_2\) in \(E_1\) such that \(e_1 \supseteq e_2\). Since \(C \in C(E_1)\), there is a field \(i\) in \(C\) that is involved in \(e_1\), i.e., \(A_i^{e_1} \neq A_i\). Since \(e_1 \supseteq e_2\), we have that \(A_i^{e_1} \supseteq A_i^{e_2}\) and so \(A_i^{e_2} \neq A_i\). That is, the field \(i\) is involved in \(e_2\). Hence \(C \in C(E_2)\).

2. Let \(e_2 \in \mathcal{X}(E_2, v)\). Then \(e_2 \in E_2\), and since \(E_1\) is superior to \(E_2\), there is an edit \(e_1\) in \(E_1\) such that \(e_1 \supseteq e_2\). Also, since \(e_2 \in \mathcal{X}(E_2, v)\), we have that \(v \in e_2\), and hence also \(v \in e_1\). Hence \(e_1 \in \mathcal{X}(E_1, v)\). Hence \(\mathcal{X}(E_1, v)\) is superior to \(\mathcal{X}(E_2, v)\), as required.

3. Since \(H\) is a superior function to \(G\), we have that \(H(E)\) is superior to \(G(E)\). Then by part 2 of this lemma, \(\mathcal{X}(H(E), v)\) is superior to \(\mathcal{X}(G(E), v)\), and hence the result follows from part 1 of this lemma. \(\square\)

**Corollary 2.5.13.** If the edit generation function \(G\) has the error correction guarantee, and \(H\) is a superior function to \(G\), then \(H\) has the error correction guarantee.
The covering set method

Proof. Let \( v \) be a record, and let \( E \) be an edit set. Since the set \( H(E) \) is superior to the set \( G(E) \), then by Lemma 2.5.12 we have that \( \mathcal{C}(H(E), v) \subseteq \mathcal{C}(G(E), v) \). But since \( G \) has the error correction guarantee we have that \( \mathcal{C}(G(E), v) \subseteq \mathcal{E}(E, v) \). Hence \( \mathcal{C}(H(E), v) \subseteq \mathcal{E}(E, v) \), and thus \( H \) has the error correction guarantee. ⊣

Corollary 2.5.14. The functions \( MFH \) and \( MENFH \) have the error correction guarantee.

Proof. By Lemma 1.9.5 part 2, the functions \( MFH \) and \( MENFH \) are superior functions to \( FH \) and \( ENFH \) respectively, and both \( FH \) and \( ENFH \) have the error correction guarantee. ⊣

Note: Kunnathur (1982, Lemma 3.1, page 38) gave the essence of the proof that dominated edits can be removed in the edit generation process without affecting the error correction guarantee.

We have now shown that the three functions \( ENFH \), \( MFH \) and \( MENFH \) have both error correction totality and the error correction guarantee. Therefore we have completed the main purpose of this section, which is to show that the three functions are covering set correctible.

Before concluding this section, I present two more items. The first item is to note that Corollary 2.5.13, about superior functions and error correction guarantee, has a dual, presented below as Corollary 2.5.15, about error correction totality. Although not relevant in this chapter, the corollary will be relevant in Chapter 6.

Corollary 2.5.15. If the edit generation function \( H \) has error correction totality, and \( H \) is a superior function to \( G \), then \( G \) has error correction totality.

Proof. Let \( v \) be a record, and let \( E \) be an edit set. Since the set \( H(E) \) is superior to the set \( G(E) \), then by Lemma 2.5.12 we have that \( \mathcal{C}(H(E), v) \subseteq \mathcal{C}(G(E), v) \). But since \( H \) has error correction totality we have that \( \mathcal{E}(E, v) \subseteq \mathcal{C}(H(E), v) \). Hence \( \mathcal{E}(E, v) \subseteq \mathcal{C}(G(E), v) \), and thus \( G \) has error correction totality. ⊣

The second item that I present before concluding this section is the example foreshadowed in Section 2.3, page 27, which was deferred until now because it depends on the function \( ENFH \). The example demonstrates that a function that is smallest weighted covering set correctible for a given set of weights need not be covering set correctible.

Example 2.5.16. Suppose that \( N = 2 \), that \( A_1 = \{1, 2, 3, 4\} \), that \( A_2 = \{1, 2\} \), that \( b = (1, 2) \), that the edit set \( X = \{\{2, 3\} \times \{1\}, \{3, 4\} \times \{2\}\} \), and that \( G(E) \) is defined for each edit set \( E \) by

\[
G(E) = \begin{cases} 
\text{ENFH}(E), & \text{if } E \neq X \\
E, & \text{if } E = X.
\end{cases}
\]

We will show that \( G \) is smallest weighted covering set correctible for the weights \((1, 2)\) but is not covering set correctible.
Firstly, to confirm that $G$ is smallest weighted covering set correctible for the weights $(1, 2)$, we show, for each edit set $E$ and each record $v$, that

$$SCX(G(E), v, (1, 2)) = SEL(E, v, (1, 2)),$$

by considering three cases:

Case 1: $E \neq X$. We can use the fact that ENFH is covering set correctible, and hence is smallest weighted covering set correctible for any set of weights:

$$SCX(G(E), v, (1, 2)) = SCX(ENFH(E), v, (1, 2)),$$

because ENFH is smallest weighted covering set correctible for any set of weights.

Case 2: $E = X$ and $v$ satisfies $X$. Then

$$SCX(G(E), v, (1, 2)) = SCX(E, v, (1, 2)),$$

by the definition of $G$

$$= SC(\emptyset), \text{ since } v \text{ satisfies } E$$

$$= \emptyset$$

$$= SEL(E, v, (1, 2)), \text{ since } v \text{ satisfies } E.$$ 

Case 3: $E = X$ and $v$ fails $X$. Then

$$SCX(G(E), v, (1, 2)) = SCX(E, v, (1, 2)),$$

by the definition of $G$

$$= \{\text{field 1}\}, \text{ since field 1 is involved in both edits of } X, \text{ and } \{\text{field 1}\} \text{ is the smallest weighted non-empty set of fields}$$

$$= SEL(E, v, (1, 2)), \text{ since any record whose first field equals 1 satisfies all of } E \text{ and } \{\text{field 1}\} \text{ is the smallest weighted non-empty set of fields}.$$ 

Having confirmed that $G$ is smallest weighted covering set correctible for the weights $(1, 2)$, we now confirm that $G$ is not covering set correctible. Consider the record $v = (3, 1)$, which fails the first edit in the set $X$ and satisfies the second. Then $\{\text{field 2}\} \in CX(G(X), v)$, because $G(X) = X$ and field 2 is uninvolved in the first edit of $X$. But $\{\text{field 2}\} \notin E(X, v)$, because the only possible correction using only field 2, namely $(3, 2)$, fails the second edit of $X$. Hence $CX(G(X), v) \neq E(X, v)$ and so $G$ is not covering set correctible.

Thus $G$ is smallest weighted covering set correctible for $(1, 2)$, but is not covering set correctible.

This section has introduced the three functions ENFH, MFH and MENFH, which are subfunctions of FH. They are all covering set correctible. One direction of their covering set correctness, namely their error correction totality, follows from their being
The covering set method subfunctions of FH. As for the other direction, namely the error correction guarantee, the proof for ENFH is different from that for MFH and MENFH. The proof of error correction guarantee of ENFH is essentially the same as that for FH, while the error correction guarantee of MFH and MENFH follows from their being superior functions of FH and ENFH respectively.

There is one more covering set correctible function to be considered: it is the function $FCF_\omega$. Its definition is more complex, and is left to the next section.

2.6 The Field Code Forest Algorithm

In this section, we define the edit generation function $FCF_\omega$, or rather, the set of such functions: there is one function of form $FCF_\omega$ for each permutation $\omega$ of the fields. As to the covering set correctibility of the functions $FCF_\omega$, its proof is significantly more complex than the other proofs of covering set correctibility and is left to Chapter 6.

The letters FCF stand for “Field Code Forest”, because the function $FCF_\omega$ is specified by the so-called Field Code Forest Algorithm, abbreviated as FCF Algorithm. The FCF Algorithm was developed by Liepins, Kunnathur and Garfinkel (Liepins 1980a, 1984; Kunnathur 1982; Garfinkel, Kunnathur, and Liepins 1984, 1986b).

There have been questions about whether the FCF Algorithm is correct, in the sense of whether the corresponding function $FCF_\omega$ has covering set correctibility (Winkler 1997). Although Garfinkel et al. (1986b) presented a proof of what they called the “sufficiency” of the FCF Algorithm, we will see in Chapter 6 that their proof proves something other than the correctness of the FCF Algorithm.

Each function $FCF_\omega$ is a subfunction of ENFH. The algorithm gives a systematic way of choosing generating fields, and considerably reduces the size of the search space of generating fields.

The FCF Algorithm finds implied edits as it traverses a field code forest, which is in fact a tree rather than a forest. Its nodes are labelled with subsets of the set $\{1, \ldots, N\}$ such that each subset of $\{1, \ldots, N\}$ labels exactly one node of the tree. An example, for $N = 4$, is given in Figure 2.1, where each subset of $\{1, \ldots, N\}$ is represented by a corresponding string.

The more detailed definition of a field code forest is:

**Definition 2.6.1.** A field code forest $F(\omega)$ is defined for each set $\{1, \ldots, N\}$ and each total ordering (or permutation) $\omega = (o_1, \ldots, o_N)$ of $\{1, \ldots, N\}$. Here the $o_k$’s are distinct elements of $\{1, \ldots, N\}$. The Field Code Forest is a labelled tree, labelled by strings over the alphabet $\{1, \ldots, N\}$.

1. The root node is labelled with the empty string ($\lambda$).

2. If a node $n$ is labelled with the string $S$ then:
   - if $o_N \in S$ then $n$ has no children, and
   - if $o_N \notin S$ and $K = \text{Max}(\{0\} \cup \{k \mid o_k \in S\})$ then $n$ has $N - K$ children, labelled with $S \circ (o_j)$ for $j \in \{K + 1, \ldots, N\}$, where $\circ$ means concatenation of strings. (Note that $K < N$.)
§2.6  The Field Code Forest Algorithm

Figure 2.1: The field code forest F((1, 2, 3, 4)).

We will use the same symbol to represent both the node’s string and the corresponding subset, whenever there is no ambiguity.

Some simple consequences of the definition are:

- Each subset of \(\{1, \ldots, N\}\) appears exactly once in the field code forest.
- Hence the node’s label (string or subset) can be used to name the node.
- The order in which fields appear in each label is the same as their order in \(\omega\).
- There are \(N\) child nodes of the root node, labelled with \((o_1), \ldots, (o_N)\) respectively.
- The order of the nodes in the tree depends on the ordering of \(\omega\).

Liepins, Kunnathur and Garfinkel give a slightly different definition of a field code forest: they omit the root node () and thereby have a set of trees, hence the use of the word “forest”. For our purposes the two definitions are equivalent; our definition makes the explanation of the algorithm slightly shorter.

The FCF Algorithm works through the forest in a depth-first order.\(^2\) A precise statement of the Algorithm is in later paragraphs but in words it is described as follows. We start with a set \(E\) of explicit edits. At each node \(\sigma\) we generate an edit set associated with node \(\sigma\) and called \(\text{GenI}(\sigma, E)\) (where “I” stands for “initial”). The edit set associated with the node \(\sigma\) may be modified as the algorithm traverses the nodes: the edit set at node \(\sigma\) just after the node \(\tau\) has been traversed will be called \(\text{GenV}(\sigma, E, \tau)\) (where “V” stands for “varying”). After the last node of the field code forest has been traversed the edit set associated with the node \(\sigma\) will be called \(\text{GenF}(\sigma, E)\) (where “F” stands for “final”). Hence, if \(\nu\) is the last node of the field code forest, then \(\text{GenF}(\sigma, E) = \text{GenV}(\sigma, E, \nu)\).

\(^2\)In fact any order works so long as deeper nodes are considered after higher nodes on the same branch.
The set \( \text{GenI}(\sigma, E) \) is calculated using FH edit generation where the generating field is the new field added at the node \( \sigma \). The edits that may be used at the node labelled \( \sigma = (i_1, \ldots, i_k) \) are not all previously generated edits: rather they are only the edits generated in nodes in the branch above the current node with fields \( i_1, \ldots, i_{k-1} \) uninvolved and with field \( i_k \) involved. Only certain edits are allowed at node \((i_1, \ldots, i_k)\): in any generated edit the field \( i_k \) must be uninvolved. If no edits can be generated at a node then the remainder of that branch is not traversed.

After calculating the set \( \text{GenI}(\sigma, E) \), each node \( \tau \) traversed thus far is revisited. At each node \( \tau \) the current edit set is updated by replacing any dominated edits by the maximal generated edits that dominate them. This gives the set \( \text{GenV}(\tau, E, \sigma) \).

A precise statement of the FCF Algorithm follows. By way of an example, Tables 2.1 and 2.2, in the appendix to this chapter, give the calculation for the explicit edit set \( E \) defined in Example 2.9.1 on page 50.

**Definition 2.6.2 (Statement of the FCF Algorithm).**

**Input to the algorithm:** a set \( E \) of edits on fields \( \{1, \ldots, N\} \), and a total ordering \( \omega \) of the set \( \{1, \ldots, N\} \).

**Steps in the algorithm:** Traverse the field code forest \( F(\omega) \) in depth-first order.
At each node \( \sigma \) of \( F(\omega) \):

1. Calculate the set \( \text{GenI}(\sigma, E) \) according to Definition 2.6.4 below.
2. Calculate the set \( \text{GenV}(\sigma, E, \sigma) \) using Domination Rule 1 given in Definition 2.6.5 below.
3. For each node \( \tau \) visited before \( \sigma \), calculate \( \text{GenV}(\tau, E, \sigma) \) using Domination Rule 2 given in Definition 2.6.5 below.

For each node \( \sigma \), let \( \text{GenF}(\sigma, E) \) be the set of edits at node \( \sigma \) after the last node \( \nu \) of the field code forest has been traversed. That is, \( \text{GenF}(\sigma, E) = \text{GenV}(\sigma, E, \nu) \).

**Output of the algorithm:** the set

\[
\text{FCF}_\omega(E) = \bigcup \{ \text{GenF}(\sigma, E) \mid \text{\sigma is a node of } F(\omega) \}.
\]

In order to define \( \text{GenI}(\sigma, E) \), we define the set \( \text{BranchV}(\sigma, E, \tau) \) to be the set of edits appearing in the tree at or above node \( \sigma \) just after node \( \tau \) has been traversed, as follows:

**Definition 2.6.3.** Suppose \( E \) is an edit set and \( \omega \) is a total ordering of the fields. Suppose also that \( \sigma \) and \( \tau \) are nodes of the field code forest \( F(\omega) \), and that either \( \sigma = \tau \) or \( \tau \) is traversed after \( \sigma \). Write \( \sigma = (i_1, \ldots, i_k) \) where \( k \geq 0 \). Then

\[
\text{BranchV}(\sigma, E, \tau) = \bigcup \{ \text{GenV}((i_1, \ldots, i_j), E, \tau) \mid j = 0, \ldots, k \}.
\]

We can now define the set \( \text{GenI}(\sigma, E) \).
Definition 2.6.4 (Definition of GenI(σ, E) for step (i) of Definition 2.6.2). Let 
σ = (i_1, ..., i_m) ⊆ ω. Define the set GenI((i_1, ..., i_m), E) inductively as follows:

a. GenI((), E) = E.

b. If m ≥ 1 and GenV((i_1, ..., i_{m-1}), E, (i_1, ..., i_{m-1})) = ∅, then
   GenI((i_1, ..., i_m), E) = ∅.

c. If m ≥ 1 and GenV((i_1, ..., i_{m-1}), E, (i_1, ..., i_{m-1})) ≠ ∅ then
   GenI((i_1, ..., i_m), E) = \{ FHG(i_m, X) | X ⊆ BranchV((i_1, ..., i_{m-1}), E, (i_1, ..., i_{m-1})), and
   FHG(i_m, X) ≠ ∅, and
   i_m is not involved in FHG(i_m, X), and
   i_m is involved in each element of X, and
   each of i_1, ..., i_{m-1} is uninvolved in each element of X \}.

For the purposes of later proofs, it is useful to list and number the properties in part
of Definition 2.6.4 of GenI given above, as follows:

For m ≥ 1, the edit α ∈ GenI((i_1, ..., i_m), E) if and only if

G1. GenI((i_1, ..., i_{m-1}), E) ≠ ∅;

G2. α = FHG(i_m, X);

G3. X ⊆ BranchV((i_1, ..., i_{m-1}), E, (i_1, ..., i_{m-1}));

G4. α ≠ ∅;

G5. the field i_m is not involved in α;

G6. the field i_m is involved in each element of X;

G7. each of the fields i_1, ..., i_{m-1} is uninvolved in each element of X.

The statement of the FCF Algorithm of Definition 2.6.2 also depends on the following
domination rules.

Definition 2.6.5 (Domination rules for steps (ii) and (iii) of Definition 2.6.2). Let
S_σ be the set of nodes visited prior to visiting σ. If σ ≠ (), then let σ’ be the
node visited immediately before σ. The domination rules, applied after calculating
GenI(σ, E), are:

Domination Rule 1. Calculate GenV(σ, E, σ):
The covering set method

(a) If \( \sigma = () \), then remove all dominated edits from \( \text{GenI}(\sigma, E) \). That is,

\[
\text{GenV}((), E, ()) = \text{Max} \circ \text{GenI}((), E).
\]

(b) If \( \sigma \neq () \), then replace each edit in \( \text{GenI}(\sigma, E) \) by all maximal dominating edits already generated. That is,

\[
\text{GenV}(\sigma, E, \sigma) = \text{Max}\left(\text{GenI}(\sigma, E) \cup \{ \beta \mid \beta \in \bigcup_{\tau \in S_{\sigma}} \text{GenV}(\tau, E, \sigma') \text{, and there is an } \alpha \text{ in } \text{GenI}(\sigma, E) \text{ such that } \beta \supset \alpha \} \right).
\]

Domination Rule 2. If \( S_{\sigma} \neq \emptyset \), then for each \( \tau \) in \( S_{\sigma} \), calculate \( \text{GenV}(\tau, E, \sigma) \) as follows: Replace any already generated edit by all maximal edits that contain it and that are in \( \text{GenV}(\sigma, E, \sigma) \). That is,

\[
\text{GenV}(\tau, E, \sigma) = \text{Max}\left(\text{GenV}(\tau, E, \sigma') \cup \{ \beta \mid \beta \in \text{GenV}(\sigma, E, \sigma), \text{ and there is an } \alpha \text{ in } \text{GenV}(\tau, E, \sigma') \text{ such that } \beta \supset \alpha \} \right).
\]

The above algorithm statement differs in places from those given in the five papers by Garfinkel, Kunnathur and Liepins (G, K & L 1984, 1986b; Kunnathur 1982; Liepins 1980a, 1984), which differ in places from each other. The appendix to this chapter gives details.

Chapter 6 will introduce a modified version of the FCF Algorithm, called the FCFS Algorithm, which is the same as the FCF Algorithm, except without the Domination Rules. We will use the FCFS Algorithm in the proof of the covering set correctibility of the function \( \text{FCF}_\omega \).

In this section we have defined the function \( \text{FCF}_\omega \) in terms of the FCF Algorithm. Although the definition of the function is complex, it is a subfunction of the more easily defined function \( \text{ENFH} \). The proof of its covering set correctibility is also complex, and we leave it to Chapter 6.

2.7 Practicalities of the covering set method

The covering set method is slow and uses much memory, because the number of generated edits increases exponentially with the number of explicit edits. As a consequence, several improvements have been suggested to the edit generation process. This section describes some of the methods to reduce the number of edits generated.

The first method for reducing the number of generated edits is to seek only maximal edits, as in the functions \( \text{MFH}, \text{MENFH} \) and \( \text{FCF}_\omega \). An additional improvement is to prevent the generation of dominated edits rather than merely deleting them after generating them. Some examples, further discussed by Winkler (1997, 1998), are as follows.
1. The edit generation step \( e = FHG(i, X) \) need only be calculated if the generating field \( i \) is involved in each edit of the generating set \( X \). Otherwise the generated edit \( e \) is dominated by one of the edits of \( X \). (Fellegi and Holt 1976, page 29, item 1.)

2. The edit generation step \( e = FHG(i, X) \) need not be calculated if there is an edit \( e' \) in the edit set \( X \), a superset \( X' \) of \( X \setminus \{e'\} \), and a field \( k \) (which may or may not equal \( i \)) such that \( e' = FHG(k, X') \). For, if \( k = i \), then \( e = e' \). On the other hand, if \( k \neq i \), then \( e \) is dominated by the edit \( FHG(i, X \setminus \{e'\}) \). (Fellegi and Holt 1976, page 29, item 2, except that Fellegi and Holt required that the edit \( e \) be essentially new.)

3. The edit generation step \( e = FHG(i, X) \) need not be calculated if there is a subset \( X' \) of \( X \) such that the edit \( e' = FHG(i, X') \) is essentially new and has already been calculated. Otherwise the edit \( e \) is dominated by the edit \( e' \). (Fellegi and Holt 1976, page 29, item 3.) Hence it is only necessary to use the minimal edit sets for which it is possible to generate an essentially new edit: Chen (1998) gives algorithms to find such minimal subsets.

4. Discard any dominated edits generated at any stage of the edit generation process (Kunnathur 1982, page 39). Otherwise, as spelt out in Lemma 2.4.10, edits generated from the dominated edits will be dominated by edits generated from the dominating edits.

Another method of dealing with the slowness of edit generation is to compromise by not completing the full edit generation process. For example, Barcaroli and Venturi (1993) partition the set of edits to reduce the size of the generated edit set. For arithmetic edits expressed as certain linear inequalities, some approaches are presented by Draper and Winkler (1997) and by Garcia (2003, 2005).

Yet another method of dealing with the slowness of edit generation is to give up altogether on generating all edits prior to the error localisation process, and instead, to treat each erroneous record as a separate stand-alone problem. That is, the method for each erroneous record is to attempt to generate only those edits that will be relevant to finding a useful covering set for that record. Such record-by-record covering set approaches are presented by Garfinkel (1979), Garfinkel, Kunnathur and Liepins (1984, 1986b), Winkler and Chen (2002), and Chen and Winkler (2004). There are similar approaches for arithmetic edits expressed as linear inequalities, presented by Garfinkel, Kunnathur and Liepins (1986a, 1988), Ragsdale and McKeown (1996), de Waal (2003a, Chapter 10), de Waal and Coutinho (2005, Section 9), and Riera-Ledesma and Salazar-González (2007a). The general scheme for such solutions is to first determine whether a smallest covering set of a small set of generated edits (such as of the explicit edits failed by the given record) is an error localisation solution, and if not, to generate additional edits until a smallest covering set is an error localisation solution.

Of course there are also many other record-by-record methods of solving the error localisation problem without using the covering set method or any variations to it. The various solution methods have been described in Chapter 1, Section 1.6.
Most of the record-by-record methods can be seen as a trade-off between search and generation, where search means systematically testing potential solutions and generation means finding new edits or other constraints. Thus the large-scale generation of edits is avoided.

Although the record-by-record methods avoid large-scale edit generation, the workload for each record is increased. In contrast, the idea of the covering set method is to do a large amount of work even before the data are received, with correspondingly less work for each individual record. The covering set method would therefore be best applied to data sets such as population censuses where there is a long time between the finalisation of the questionnaire and data entry. It is during this time that the large-scale edit generation might be completed. Once the data are entered, the error localisation itself should take less time than the record by record approach.

The use of logic also provides some hope. Provided that the edits are suitably formalised in logical terms, logical “consequence finders” can potentially be used to efficiently generate huge numbers of edits. For example, Simon and del Val (2001) present consequence finders that can generate over $10^{70}$ edits, represented by Binary Decision Diagrams.

In order to use such logical consequence finders, two problems must be solved. Firstly, edit generation functions must be formalised in terms of logic. Such a formalisation is the topic of the next two chapters: Chapter 3 presents some background on logic, while Chapter 4 presents the actual formalisation of edit generation functions. Secondly, it must be demonstrated that the function calculated by the logical consequence finder is in fact covering set correctible, so that the covering set method can be applied. Therefore, in Chapter 5 we present a formalisation of covering set correctibility in terms of some natural constructs of logic. The corresponding results for arithmetic edits are presented in Chapter 7.

While the covering set method seems promising, it has its problems in terms of speed and memory usage. There are many techniques for improving or compromising the covering set method. One method that has not been fully explored is the use of fast consequence finders. In order to use fast consequence finders, all aspects of the covering set method must be formalised in terms of logic. The aspects include both edit generation and covering set correctibility.

### 2.8 Conclusion

This chapter has introduced the covering set method for data editing, as applied to categorical edits. It has also introduced a range of basic definitions and properties relevant to the covering set method for categorical edits. As for arithmetic edits, we leave the corresponding introduction to Chapter 7.

The main assumptions and concepts introduced in this chapter relate to: data records, edits, error localisation, edit generation functions, the covering set method itself, and properties of edit generation functions. We assume that the data is arranged in records, and that edits are used to specify potential errors in the data and apply to one record at a time. The covering set method solves the error localisation problem:
given a record and a set of edits, find a (smallest) set of fields on which the record can be corrected. The covering set method depends on finding a set of generated edits that are found via an edit generation function, such as the function FH. Provided that the edit generation function is covering set correctible, then the error localisation solutions for a given record \( v \) are exactly the covering sets of the generated edits failed by the record \( v \).

There are many edit generation functions: those described in this chapter are subfunctions (Definition 1.9.3) of the function FH. Their inter-relationships are presented in Figure 2.2, which shows the function FH and four subfunctions, some of which are subfunctions of each other. The functions MENFH and FCF\( _{\omega} \) are subfunctions of the function ENFH, which in turn is a subfunction of FH; and the function MFH is also a subfunction of FH.

![Diagram of the inter-relationships amongst the various edit generation functions mentioned in this chapter. The functions are all subfunctions of the function FH. Each function is a subfunction of the function above it in the tree.](image)

Each of the functions described in this chapter, and displayed in Figure 2.2, is covering set correctible and can therefore be used with the covering set method. This chapter contains summaries of the proofs of the covering set correctibility of each of the functions except for the function FCF\( _{\omega} \), which has a complex proof, left until Chapter 6.

The proof of covering set correctibility consists, for all the functions presented here, of two parts, corresponding to the two parts of the definition of covering set correctibility. The two parts are: error correction guarantee and error correction totality.

The proofs of the error correction guarantee of the various functions have mostly been published in the past, although the term "error correction guarantee" had not been defined. The proof of the error correction guarantee for FH and the essence of the proof for ENFH was presented by Fellegi and Holt (1973, 1976). The essence of the proof of the error correction guarantee of the functions MFH and MENFH was presented by Kunnathur (1982). In this chapter we also saw that the error correction guarantee of MFH and MENFH is a consequence of a general result about superior functions (Definition 1.9.4). The proof of the error correction guarantee of FCF\( _{\omega} \) is
The covering set method presented in Chapter 6.

The proofs of error correction totality of the various functions have not been published in the past, partly because the concept of error correction totality had not been defined and partly because the proofs are straightforward. The error correction totality of FH follows quickly from a lemma of Fellegi and Holt (1973, 1976), while the proof of the error correction totality of the other functions follows from a general result about subfunctions.

Since the covering set method is slow and uses much memory, there have been techniques developed for improving or compromising the covering set method. Logic provides some potential for improvement, because of the existence of fast consequence finders. But in order to use such consequence finders it is necessary to formalise in terms of logic the two main components of the covering set method. These two components are: edit generation and covering set correctibility. Most of the remainder of this thesis deals with the formalisation of the two components.
2.9 Appendix: Different statements of the FCF Algorithm

The above algorithm statement (Section 2.6) differs in places from those given in the five papers by Garfinkel, Kunnathur and Liepins (G, K & L 1984, 1986b; Kunnathur 1982; Liepins 1980a, 1984), which differ in places from each other. The differences appear in two aspects: the choice of edits to use in any edit generation, and the domination rules.

Firstly, in the statement given in Section 2.6, the edits used at node \((i_1, \ldots, i_m)\), that is the edits in the set \(X\) of Definition 2.6.4, part c, must appear at nodes above \((i_1, \ldots, i_m)\). Liepins (1980a, 1984) appears to agree with this, referring to “antecedent” nodes, each of which is defined however as the single node immediately above the current node. Differently, Kunnathur (1982) and G, K & L (1984, 1986b) require that the edits in the set \(X\) are at any nodes already traversed in the forest (Kunnathur) or in the current tree (G, K & L 1984, 1986b). The proofs in Chapter 6 show that Kunnathur’s and G, K & L’s sets \(X\) are bigger than necessary: it is enough to use Liepins’ definition of \(X\) - but using all antecedent nodes rather than just the immediate antecedent node.

The second difference between the statement of Section 2.6 and those of G, K & L is about the domination rules. The statement of Section 2.6 follows Kunnathur (1982). Differently, G, K & L (1984, 1986b) - albeit ambiguously - and the two papers by Liepins (1980a, 1984) say that newly generated dominated edits should be discarded at their nodes, rather than replaced by the dominating edits. Instead of the Domination Rule 1 of Definition 2.6.5, they use:

\(G, K \& L\) Domination Rule 1: Let the current node be \(\sigma\). Let \(S_\sigma\) be the set of nodes visited prior to visiting the node \(\sigma\). If \(\sigma \neq ()\), then let \(\sigma'\) be the node visited immediately before \(\sigma\). Calculate \(\text{GenV}(\sigma, E, \sigma)\) as follows:

(a) If \(\sigma = ()\), then remove all dominated edits from \(\text{GenI}(\sigma, E)\). That is,

\[
\text{GenV}((), E, ()) = \text{Max} \circ \text{GenI}((), E).
\]

(b) If \(\sigma \neq ()\), then, for each \(\alpha\) in \(\text{GenI}(\sigma, E)\), if there is a node \(\tau\) in \(S_\sigma\) and an edit \(\beta\) in \(\text{GenV}(\tau, E, \sigma')\) with \(\beta \supset \alpha\), then exclude the edit \(\alpha\) from the set \(\text{GenV}(\sigma, E, \sigma)\). That is,

\[
\text{GenV}(\sigma, E, \sigma) = \{ \alpha \in \text{GenI}(\sigma, E) \mid \text{for each edit } \beta \text{ in } (\text{GenI}(\sigma, E) \cup \bigcup_{\tau \in S_\sigma} \text{GenV}(\tau, E, \sigma')) \text{ we have that } \alpha \not\subset \beta \}.
\]

However, discarding dominated edits can cause a node to become empty, preventing traversal of the subtree with that node as root. Below is an example to show that
The covering set method

discarding a dominated edit, as required by G, K & L Domination Rule 1, would cause
the function $\text{FCF}_\omega$ to not be covering set correctible. In particular $\text{FCF}_\omega$ would not
have the error correction guarantee.

**Example 2.9.1.** This is a case where using G, K & L Domination Rule 1 causes the
FCF function to no longer have the error correction guarantee. Let $A_1 = A_2 = \{1, 2, 3\}$. Let $A_3 = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Let the order of the fields $= \omega = (\text{field 1, field 2, field 3})$.

Let the set $E$ of explicit edits be $\{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$, where

- $e_1 = \{1, 2\} \times \{2, 3\} \times \{3, 4, 9\}$
- $e_2 = \{3\} \times \{2, 3\} \times \{3, 4, 8\}$
- $e_3 = \{1, 3\} \times \{1\} \times \{3, 5, 6\}$
- $e_4 = \{2\} \times \{1\} \times \{3, 5, 7\}$
- $e_5 = \{1\} \times \{2\} \times \{2, 3\}$
- $e_6 = \{1\} \times \{1, 3\} \times \{1, 3\}$
- $e_7 = \{1\} \times A_2 \times \{1, 2, 4, 5, 6, 7, 8, 9\}$.

The calculation of $\text{FCF}_\omega(E)$ (using Definition 2.6.2 of Section 2.6) is given in
Table 2.1. Each cell of the table lists the generated edits at the various stages of the
algorithm, and the actual calculation of the generated edits is in the subsequent table,
Table 2.2. The first row of Table 2.1 lists all nodes $\sigma$ in order of traversal, except $\sigma = (1, 2, 3)$ and $\sigma = (1, 3)$ for which $\text{GenI}(\sigma, E)$ is empty. The second row lists the
values of $\text{GenI}(\sigma, E)$, which (other than $\text{GenI}((), E)$) depend on the relevant values of
$\text{GenV}(\sigma, E, \tau)$. These latter are given in the remaining rows: each row gives, for some
node $\tau$, the values of $\text{GenV}(\sigma, E, \tau)$ for all the values of $\sigma$ before or equal to $\tau$. Each
row corresponds to one node $\tau$, and the nodes $\tau$ are in the order of the traversal of the
field code forest. Thus the various sets $\text{GenV}(\sigma, E, \tau)$ are calculated in order row by row
and left to right in each row. The last column gives the edit domination relationships
that are used in the table. The last row, corresponding to the last node (3), also gives
the values of $\text{GenF}(\sigma, E)$.

From Table 2.1, $\text{FCF}_\omega(E) = \bigcup \{\text{GenF}(\sigma, E) \mid \sigma \in F(\omega)\}$

\[= \{e_1, e_2, e_3, e_4, e_5, e_8, e_9, e_{15}, e_{19}\}. \]

Chapter 6 will show that the covering set method can be successfully used with this set
to do error localisation.

If the Domination Rule 1 of Definition 2.6.5 is replaced by G, K & L’s Domination
Rule 1, we obtain a different set of edits, which we will call $\text{FCF}_\omega'(E)$, for which the
covering set method cannot be successfully applied. The calculation of $\text{FCF}_\omega'(E)$ is in
Table 2.3, which has the same structure as Table 2.1. The calculations that are changed
are circled. The intermediate calculations (rows) at Nodes $()$, (1), (1,2), (1,2,3) and
(1,3) are unchanged. However Node (2) is empty because the edits generated there
are dominated by a previous edit ($e_{15}$). This causes Node (2,3) to also be empty.
Then, when traversing Node (2,3), the edits at Node () are not updated by dominating edits. Then, unlike in Table 2.1, new edits $e_{20}, e_{21}, e_{22}, e_{23}$ and $e_{24}$ can be calculated at Node (3) from the edits at Node () - the calculation is given in Table 2.4.

From Table 2.3, $\text{FCF}'_\omega (E) = \bigcup \{ \text{GenF}(\sigma, E) \mid \sigma \in F(\omega) \} = \{e_1, e_2, e_3, e_4, e_7, e_8, e_9, e_{15}, e_{20}, e_{23}\}$.

The two sets $\text{FCF}_\omega (E)$ and $\text{FCF}'_\omega (E)$ are the same except that $\text{FCF}_\omega (E)$ contains edit $e_{19}$ while $\text{FCF}'_\omega (E)$ contains edits $e_7, e_{20}$ and $e_{23}$. This difference causes a difference in the success of the covering set method.

We now demonstrate that the function $\text{FCF}'_\omega$ does not have the error correction guarantee. We do this by demonstrating that the error in the record $v = (1,3,3)$ cannot be localised using the covering set method with the edit set $\text{FCF}'_\omega (E)$. The edits of $\text{FCF}'_\omega (E)$ failed by $v$ are $\{e_1, e_8, e_{15}, e_{20}, e_{23}\}$. A smallest covering set of this set is the set $\{\text{field 2, field 3}\}$. However there is no way to correct $v$ by changing only fields 2 and 3, as seen next:

Field 3 must take the value 3, in order to satisfy edit $e_7$ (since field 1 is to remain unchanged).

If field 2 is changed to the value 2 then the new record (1,2,3) fails edit $e_1$.

If field 2 is changed to value 1 then the new record (1,1,3) fails edit $e_3$.

However it is possible to use the function $\text{FCF}_\omega$, which does have the error correction guarantee, to correct $v$. The edits of $\text{FCF}_\omega (E)$ which are failed by $v$ are $\{e_1, e_8, e_{15}, e_{19}\}$ - the same set as above but with the edit $e_{19}$ added and the edits $e_{20}$ and $e_{23}$ removed. The set $\{\text{field 2, field 3}\}$ no longer covers this set, but the set $\{\text{field 1, field 3}\}$ does, and indeed it is possible to correct $v$ by changing the values of fields 1 and 3. For example, the record (2,3,1) satisfies all the edits.
Table 2.1: Calculation of $\text{FCF}_\omega(E)$ for Example 2.9.1. Each cell of the table lists the generated edits at the various stages of the algorithm, and the actual calculation of the generated edits is in the subsequent table, Table 2.2. The first row of the table below lists all nodes $\sigma$ in order of traversal, except $\sigma = (1, 2, 3)$ and $\sigma = (1, 3)$ for which $\text{GenI}(\sigma, E)$ is empty. The second row lists the values of $\text{GenI}(\sigma, E)$, which (other than $\text{GenI}(() , E)$) depend on the relevant values of $\text{GenV}(- , E , -)$. These latter are given in the remaining rows: each row gives, for some node $\tau$, the values of $\text{GenV}(\sigma, E, \tau)$ for all values of $\sigma$ before or equal to $\tau$. Each row corresponds to one node $\tau$, and the nodes $\tau$ are in the order of the traversal of the field code forest. Thus the various sets $\text{GenV}(\sigma, E, \tau)$ are calculated in order row by row and left to right in each row. The last column gives the edit domination relationships that are used in the table. The last row, corresponding to the last node (3), also gives the values of $\text{GenF}(\sigma, E)$.

<table>
<thead>
<tr>
<th>$\sigma$</th>
<th>(())</th>
<th>((1))</th>
<th>((1, 2))</th>
<th>((2))</th>
<th>((2, 3))</th>
<th>((3))</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{GenI}(\sigma, E)$</td>
<td>{(e_1, e_2, e_3, e_4, e_5, e_6, e_7}}</td>
<td>{(e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}, e_{14})}</td>
<td>{(e_{15})}</td>
<td>{(e_{16}, e_{17}, e_{18})}</td>
<td>{(e_{19})}</td>
<td>(\emptyset)</td>
<td></td>
</tr>
<tr>
<td>$\text{GenV}(\sigma, E, ())$</td>
<td>{(e_1, e_2, e_3, e_4, e_5, e_6, e_7)}</td>
<td>{(e_8, e_9)}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{GenV}(\sigma, E, (1))$</td>
<td>{(e_1, e_2, e_3, e_4, e_5, e_6, e_7)}</td>
<td>{(e_8, e_9)}</td>
<td>{(e_{15})}</td>
<td></td>
<td></td>
<td></td>
<td>(e_8 \supseteq e_{10}) (e_8 \supseteq e_{11}) (e_8 \supseteq e_{12}) (e_9 \supseteq e_{13}) (e_9 \supseteq e_{14})</td>
</tr>
<tr>
<td>$\text{GenV}(\sigma, E, (1, 2))$</td>
<td>{(e_1, e_2, e_3, e_4, e_5, e_6, e_7)}</td>
<td>{(e_8, e_9)}</td>
<td>{(e_{15})}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{GenV}(\sigma, E, (2))$</td>
<td>{(e_1, e_2, e_3, e_4, e_5, e_6, e_7)}</td>
<td>{(e_8, e_9)}</td>
<td>{(e_{15})}</td>
<td>{(e_{15})}</td>
<td></td>
<td></td>
<td>(e_{15} \supseteq e_{16}) (e_{15} \supseteq e_{17}) (e_{15} \supseteq e_{18})</td>
</tr>
<tr>
<td>$\text{GenV}(\sigma, E, (2, 3))$</td>
<td>{(e_1, e_2, e_3, e_4, e_{19})}</td>
<td>{(e_8, e_9)}</td>
<td>{(e_{15})}</td>
<td>{(e_{15})}</td>
<td>{(e_{19})}</td>
<td></td>
<td>(e_{19} \supseteq e_{5}) (e_{19} \supseteq e_{6}) (e_{19} \supseteq e_{7})</td>
</tr>
<tr>
<td>$\text{GenV}(\sigma, E, (3))$</td>
<td>(= \text{GenF}(\sigma, E))</td>
<td>{(e_1, e_2, e_3, e_4, e_{19})}</td>
<td>{(e_8, e_9)}</td>
<td>{(e_{15})}</td>
<td>{(e_{15})}</td>
<td>{(e_{19})}</td>
<td>(\emptyset)</td>
</tr>
</tbody>
</table>
Table 2.2: Calculation of generated edits used in Table 2.1 for the calculation of $\text{FCF}_\omega(E)$ for Example 2.9.1.

<table>
<thead>
<tr>
<th>$e_8$</th>
<th>$\text{FHG}(1, {e_1, e_2})$</th>
<th>$= A_1 \times {2, 3} \times {3, 4}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_9$</td>
<td>$\text{FHG}(1, {e_3, e_4})$</td>
<td>$= A_1 \times {1} \times {3, 5}$</td>
</tr>
<tr>
<td>$e_{10}$</td>
<td>$\text{FHG}(1, {e_1, e_2, e_5})$</td>
<td>$= A_1 \times {2} \times {3}$</td>
</tr>
<tr>
<td>$e_{11}$</td>
<td>$\text{FHG}(1, {e_1, e_2, e_6})$</td>
<td>$= A_1 \times {3} \times {3}$</td>
</tr>
<tr>
<td>$e_{12}$</td>
<td>$\text{FHG}(1, {e_1, e_2, e_7})$</td>
<td>$= A_1 \times {2} \times {4}$</td>
</tr>
<tr>
<td>$e_{13}$</td>
<td>$\text{FHG}(1, {e_3, e_4, e_6})$</td>
<td>$= A_1 \times {1} \times {3}$</td>
</tr>
<tr>
<td>$e_{14}$</td>
<td>$\text{FHG}(1, {e_3, e_4, e_7})$</td>
<td>$= A_1 \times {1} \times {5}$</td>
</tr>
<tr>
<td>$e_{15}$</td>
<td>$\text{FHG}(2, {e_8, e_9})$</td>
<td>$= A_1 \times A_2 \times {3}$</td>
</tr>
<tr>
<td>$e_{16}$</td>
<td>$\text{FHG}(2, {e_1, e_3})$</td>
<td>$= {1} \times A_2 \times {3}$</td>
</tr>
</tbody>
</table>

(Also, $e_{16} = \text{FHG}(2, \{e_1, e_6\}) = \text{FHG}(2, \{e_3, e_6\}) = \text{FHG}(2, \{e_1, e_5, e_6\})$
\[= \text{FHG}(2, \{e_3, e_5, e_6\}) = \text{FHG}(2, \{e_1, e_3, e_6\})\].)

| $e_{17}$ | $\text{FHG}(2, \{e_1, e_4\})$ | $= \{2\} \times A_2 \times \{3\}$ |
| $e_{18}$ | $\text{FHG}(2, \{e_2, e_3\})$ | $= \{3\} \times A_2 \times \{3\}$ |
| $e_{19}$ | $\text{FHG}(3, \{e_7, e_{15}\})$ | $= \{1\} \times A_2 \times A_3$ |
Table 2.3: Calculation, using $G, K \& L$ Domination Rule 1, of $FCF'_{\sigma}(E)$ for Example 2.9.1. This table has been calculated in order to compare $FCF'_{\sigma}(E)$ with $FCF_{\omega}(E)$, which was given in Table 2.1. The table has the same structure as Table 2.1. The entries that are different from Table 2.1 are circled.

<table>
<thead>
<tr>
<th>$\sigma$</th>
<th>$()$</th>
<th>$(1)$</th>
<th>$(1, 2)$</th>
<th>$(2)$</th>
<th>$(2, 3)$</th>
<th>$(3)$</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>GenI($\sigma, E$)</td>
<td>${e_1, e_2, e_3, e_4, e_5, e_6, e_7}$</td>
<td>${e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}, e_{14}}$</td>
<td>${e_{15}}$</td>
<td>${e_{16}, e_{17}, e_{18}}$</td>
<td>$\emptyset$</td>
<td>${e_{20}, e_{21}, e_{22}, e_{23}, e_{24}}$</td>
<td>Notes</td>
</tr>
<tr>
<td>GenV($\sigma, E, ()$)</td>
<td>${e_1, e_2, e_3, e_4, e_5, e_6, e_7}$</td>
<td>${e_8, e_9}$</td>
<td>$\emptyset$</td>
<td>${e_{15}}$</td>
<td>$\emptyset$</td>
<td>${e_{20}, e_{21}, e_{22}, e_{23}, e_{24}}$</td>
<td>Notes</td>
</tr>
<tr>
<td>GenV($\sigma, E, (1)$)</td>
<td>${e_1, e_2, e_3, e_4, e_5, e_6, e_7}$</td>
<td>${e_8, e_9}$</td>
<td>${e_{15}}$</td>
<td>$\emptyset$</td>
<td>${e_{20}, e_{21}, e_{22}, e_{23}, e_{24}}$</td>
<td>Notes</td>
<td></td>
</tr>
<tr>
<td>GenV($\sigma, E, (1, 2)$)</td>
<td>${e_1, e_2, e_3, e_4, e_5, e_6, e_7}$</td>
<td>${e_8, e_9}$</td>
<td>${e_{15}}$</td>
<td>$\emptyset$</td>
<td>${e_{20}, e_{21}, e_{22}, e_{23}, e_{24}}$</td>
<td>Notes</td>
<td></td>
</tr>
<tr>
<td>GenV($\sigma, E, (2)$)</td>
<td>${e_1, e_2, e_3, e_4, e_5, e_6, e_7}$</td>
<td>${e_8, e_9}$</td>
<td>${e_{15}}$</td>
<td>$\emptyset$</td>
<td>${e_{20}, e_{21}, e_{22}, e_{23}, e_{24}}$</td>
<td>Notes</td>
<td></td>
</tr>
<tr>
<td>GenV($\sigma, E, (2, 3)$)</td>
<td>${e_1, e_2, e_3, e_4, e_5, e_6, e_7}$</td>
<td>${e_8, e_9}$</td>
<td>${e_{15}}$</td>
<td>$\emptyset$</td>
<td>${e_{20}, e_{21}, e_{22}, e_{23}, e_{24}}$</td>
<td>Notes</td>
<td></td>
</tr>
<tr>
<td>GenV($\sigma, E, (3)$)</td>
<td>${e_1, e_2, e_3, e_4, e_{20}, e_{23}, e_7}$</td>
<td>${e_8, e_9}$</td>
<td>${e_{15}}$</td>
<td>$\emptyset$</td>
<td>${e_{20}, e_{21}, e_{22}, e_{23}, e_{24}}$</td>
<td>Notes</td>
<td></td>
</tr>
<tr>
<td>$= \text{GenF}(\sigma, E)$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>${e_{20}, e_{23}}$</td>
<td>$\emptyset$</td>
<td>${e_{20}, e_{23}}$</td>
<td>$\emptyset$</td>
<td>Notes</td>
</tr>
</tbody>
</table>
Table 2.4: Calculation of additional generated edits used in Table 2.3 for the calculation of \( FCF'_{\omega}(E) \) for Example 2.9.1. The calculations for the other generated edits are in Table 2.2.

\[

e_{20} = \text{FHG}(3, \{e_1, e_7\}) = \{1\} \times \{2, 3\} \times A_3
\]

\[
e_{21} = \text{FHG}(3, \{e_3, e_7\}) = \{1\} \times \{1\} \times A_3
\]

(Also, \( e_{21} = \text{FHG}(3, \{e_3, e_6, e_7\}) \).

\[
e_{22} = \text{FHG}(3, \{e_5, e_7\}) = \{1\} \times \{2\} \times A_3
\]

(Also, \( e_{22} = \text{FHG}(3, \{e_3, e_5, e_7\}) \).

\[
e_{23} = \text{FHG}(3, \{e_6, e_7\}) = \{1\} \times \{1, 3\} \times A_3
\]

\[
e_{24} = \text{FHG}(3, \{e_1, e_6, e_7\}) = \{1\} \times \{3\} \times A_3
\]
The covering set method
3.1 Introduction

The covering set method requires the generation of useful new edits from old - a similar process to “deduction” in formal logic. In later chapters we will formalise precisely the nature of the similarity, while in this chapter we provide the necessary background about formal logic and related topics.

There are many different types of formal logic, and we will find in Chapter 4 that the relevant one for our formalisation is propositional logic. Therefore this chapter is confined to propositional logic, with the relevant background given in Section 3.2.

In Chapter 4 we will formalise edit generation functions as so-called logical deduction functions. As part of the formalisation, we also need to formalise the relationship between the domains of the two types of functions: while the domain of any edit generation function is based on the set of all possible edits, the domain of any deduction function is based on the set of all possible “logical formulae”. Both sets can be described in terms of Boolean algebras, about which Section 3.3 gives relevant background information.

Although the material in this chapter is well-known, it is included to make the thesis self-contained. In some places, the perspective is not standard, and where proofs are not readily available they are provided here.

3.2 Propositional logic

Formal logic is a method of representing aspects of human reasoning. There are many different types of formal logic but each one consists of three components:

Syntax. The syntax of the logic is the set of symbols that may be used and the rules for legal expressions.

Semantics. The semantics of the logic is a method of assigning meaning to the legal expressions of logic. We can use the semantics to determine which expressions are logically implied by which other expressions.
Deduction function. A deduction function for the logic maps each set of legal expressions to a set of “consequences” also expressed as legal expressions, without any consideration of the semantics. In this thesis we allow more than one deduction function for any one logic.

We will define the propositional logic \( \text{Prop}(\text{PropAtom}, \text{Thy}) \), where \( \text{PropAtom} \) is a set relevant to the syntax (explained in Section 3.2.1) and \( \text{Thy} \) is a set relevant to the semantics (explained in Section 3.2.2). We will define deduction functions in Section 3.2.4, where we will also define a particular deduction function called resolution. Resolution is not in fact defined on all possible legal logical expressions but only on certain ones known as clauses which will be defined in Section 3.2.3.

An important problem in any logic is whether the logical implications of the semantics and the outputs of the deduction function are the same. If they are the same we say that the deduction function is sound and complete with respect to the semantics. We will discuss the soundness and completeness of \( \text{Prop}(\text{PropAtom}, \text{Thy}) \) in more detail in Section 3.2.5.

Another important problem in any logic is to decide whether a given set of legal expressions can be consistently assigned a meaning. For propositional logic this decision problem is called the “propositional satisfiability” problem, which is discussed in more detail in Section 3.2.6.

3.2.1 Syntax

The syntax of a logic is the language of the symbols that may be used and the rules for legal expressions. The allowed symbols of \( \text{Prop}(\text{PropAtom}, \text{Thy}) \) are propositional atoms and connectives (defined below), as well as parentheses.

Definition 3.2.1. The set of propositional atoms is the set \( \text{PropAtom} \) of some chosen symbols, often \( \{ p_1, p_2, \ldots \} \).

The connectives or operations are the binary symbols \( \lor \) (“or”), \( \land \) (“and”) and the unary symbol \( \neg \) (“not”).

In our application of propositional logic to edit generation, we will require the set of propositional atoms to be finite.

The legal expressions of propositional logic are called formulae:

Definition 3.2.2. Formulae are defined inductively:

Each propositional atom is a formula. If \( \alpha \) and \( \beta \) are formulae, then so are \( (\alpha \lor \beta) \), \( (\alpha \land \beta) \) and \( (\neg \alpha) \). The expression \( (\alpha \rightarrow \beta) \) is an abbreviation for \( (\neg \alpha \lor \beta) \). The set of formulae will be called PropForm. We will treat certain formulae as equal, according to the following rules, for \( \alpha, \beta, \gamma \) formulae:

- **Idempotence:** \( (\alpha \lor \alpha) = \alpha \); \( (\alpha \land \alpha) = \alpha \);  
- **Commutativity:** \( (\alpha \lor \beta) = (\beta \lor \alpha) \); \( (\alpha \land \beta) = (\beta \land \alpha) \);
§3.2 Propositional logic

Associativity: \((\alpha \lor \beta) \lor \gamma = (\alpha \lor (\beta \lor \gamma))\) and can be written \((\alpha \lor \beta \lor \gamma)\); 
\((\alpha \land \beta) \land \gamma = (\alpha \land (\beta \land \gamma))\) and can be written \((\alpha \land \beta \land \gamma)\).

We will omit parentheses where possible according to the order of operations, with \(\neg\) having priority over \(\land\), which has priority over \(\lor\). We will also omit the outer parentheses of formulae, where this omission causes no confusion.

Consequences of commutativity and associativity are, for the formulae \(\alpha_1, \ldots, \alpha_n\), that:

1. we can use the notation \(\bigvee_{i=1}^n \alpha_i\) to represent \((\cdots ((\alpha_1 \lor \alpha_2) \lor \alpha_3) \cdots \lor \alpha_n)\); and
2. we can use the notation \(\bigwedge_{i=1}^n \alpha_i\) to represent \((\cdots ((\alpha_1 \land \alpha_2) \land \alpha_3) \cdots \land \alpha_n)\);
3. the notations \(\bigvee_{\alpha \in X} \alpha\) and \(\bigwedge_{\alpha \in X} \alpha\), where \(X\) is a finite set of formulae, are unambiguous.

We will use the shorthand \(\Box\) (called “box”) for the formula \(\bigvee_{\alpha \in \emptyset} \alpha\) and for \(\bigvee_{i=a}^b \alpha_i\) when \(a > b\). Similarly, we will use the shorthand \(\top\) (called “top”) for the formula \(\bigwedge_{\alpha \in \emptyset} \alpha\) and for \(\bigwedge_{i=a}^b \alpha_i\) when \(a > b\).

Having defined the syntax of the logic \(\text{Prop(PropAtom, Thy)}\), we are now ready to consider its semantics.

3.2.2 Semantics

The semantics of the logic gives meaning to the formulae by way of a theory and a truth function.

**Definition 3.2.3.** A *theory* is a chosen set of formulae.

Note: There are other ways of defining theories: here we follow the definition used by Wójcicki (1988, page 45) and by Keisler (1977, page 50).

The theory is chosen to represent the truths of the particular example under consideration.

**Example 3.2.4.** Let \(\text{PropAtom} = \{p_1, p_2, \ldots\}\). A possible theory is the set 
\[
\{\neg p_j \lor \neg p_k \mid j \neq k\}.
\]

In this case the example under consideration requires that at most one of the propositional atoms in PropAtom be considered “true”.

In the above example, the theory is invariant under any permutation of the set PropAtom. However, we will not in general require such invariance.

Each Thy-truth function gives a method of calculating the “truth” or “falseness” of each formula.
Definition 3.2.5. We will write the set of possible truth values as

\[ 2 = \{\text{true}, \text{false}\}. \]

If \( \text{Thy} \) is a theory, then a \textbf{Thy-truth function} is a function

\[ f : \text{PropForm} \to 2, \]

such that for each formula \( \alpha \) and each finite set \( X \) of formulae:

1. if \( \alpha \in \text{Thy} \) then \( f(\alpha) = \text{true} \);
2. \( f(\neg \alpha) = \begin{cases} \text{true} & \text{if } f(\alpha) = \text{false} \\ \text{false} & \text{if } f(\alpha) = \text{true} \end{cases} \);
3. \( f(\bigvee_{\xi \in X} \xi) = \begin{cases} \text{true} & \text{if there exists a formula } \xi \text{ in } X \text{ such that } f(\xi) = \text{true} \\ \text{false} & \text{otherwise} \end{cases} \);
4. \( f(\bigwedge_{\xi \in X} \xi) = \begin{cases} \text{true} & \text{if for each formula } \xi \text{ in } X \text{ we have that } f(\xi) = \text{true} \\ \text{false} & \text{otherwise} \end{cases} \).

If \( \text{Thy} = \emptyset \) we will write “truth function” rather than “\( \emptyset \)-truth function”.

Note 1: Items 3 and 4 above give the truth values of formulae such as \( \xi_1 \vee \xi_2 \) and \( \xi_1 \land \xi_2 \).

Note 2: Thy-truth functions are usually called models of Thy, but in this thesis it is more convenient to refer to Thy-truth functions.

Applying items 3 and 4 of the above definition to \( X = \emptyset \) gives us that that \( f(\Box) = \text{false} \) and \( f(\top) = \text{true} \), for any Thy-truth function \( f \).

The next theorem tells us that we can define a truth function by defining its value on the propositional atoms only.

Theorem 3.2.6. Given a function \( f : \text{PropAtom} \to 2 \) there is a unique truth function \( f' : \text{PropForm} \to 2 \) such that for each atom \( p \), we have \( f(p) = f'(p) \).


Note that although the function \( f' \) of the above theorem is a truth function, it is not necessarily a Thy-truth function. Whether or not it is a Thy-truth function depends on the particular value of Thy.

We are interested in those truth functions \( f \) and formulae \( \sigma \) for which \( f(\sigma) = \text{true} \), in which case we use the term “satisfies” as per the next definition.

Definition 3.2.7. A Thy-truth function \( f \) \textbf{satisfies} a formula \( \sigma \), or is a \textbf{model} of \( \sigma \), if \( f(\sigma) = \text{true} \). If \( f(\sigma) = \text{false} \), we say that \( f \) \textbf{fails} the formula \( \sigma \). The Thy-truth function \( f \) \textbf{satisfies} a set \( \Sigma \) of formulae, or is a model of \( \Sigma \), if \( f \) satisfies every formula in \( \Sigma \). The Thy-truth function \( f \) \textbf{fails} a set \( \Sigma \) of formulae if \( f \) fails some formula in \( \Sigma \).
We say that the set $\Sigma$ is **Thy-satisfiable** when there exists some Thy-truth function that satisfies $\Sigma$. If $\text{Thy} = \emptyset$ then we say that “$\Sigma$ is satisfiable” instead of “$\Sigma$ is $\emptyset$-satisfiable”.

If the formula $\sigma$ is satisfied by every Thy-truth function, then we say that $\sigma$ is **Thy-valid**. If $\text{Thy} = \emptyset$ then we say “$\sigma$ is valid” instead of “$\sigma$ is $\emptyset$-valid”.

Note: $\Sigma$ is Thy-satisfiable if and only if $\Sigma \cup \text{Thy}$ is satisfiable.

We will be interested in the relationships between sets of Thy-satisfiable formulae. In particular, we are interested in the situation when every Thy-truth function that satisfies the set $\Sigma$ of formulae also satisfies the formula $\alpha$: in such a case we will use the term “Thy-logically implies” as per the next definition.

**Definition 3.2.8.** If Thy is a theory, then the set $\Sigma$ of formulae **Thy-logically implies** the formula $\alpha$ if the following holds:

$$
\text{if } f \text{ is a Thy-truth function that satisfies } \Sigma \text{ then } f(\alpha) = \text{true}.
$$

If $\Sigma$ Thy-logically implies $\alpha$, we write $\Sigma \equiv_{\text{Thy}} \alpha$, or $\Sigma \cup \text{Thy} \equiv \alpha$, or $\Sigma, \text{Thy} \equiv \alpha$.

If $\Sigma = \{\sigma\}$ we will write $\sigma \equiv_{\text{Thy}} \alpha$ instead of $\{\sigma\} \equiv_{\text{Thy}} \alpha$; and we will write $\sigma, \text{Thy} \equiv \alpha$ instead of $\{\sigma\}, \text{Thy} \equiv \alpha$.

If $\text{Thy} = \emptyset$, then we will say “$\Sigma$ logically implies $\alpha$” instead of “$\Sigma \emptyset$-logically implies $\alpha$” and we will write $\Sigma \equiv \alpha$ instead of $\Sigma \emptyset \equiv \alpha$ or $\Sigma, \emptyset \equiv \alpha$.

Note: $\Sigma$ Thy-logically implies $\alpha$ if and only if $\Sigma \cup \text{Thy}$ logically implies $\alpha$.

Replacing the formula $\alpha$ in the above definition by the formula $\Box$ (defined on page 59) gives us another definition for Thy-satisfiability:

$$
\Sigma \equiv_{\text{Thy}} \Box \text{ if and only if } \Sigma \text{ is not Thy-satisfiable,}
$$

(3.2.1) since $f(\Box) = \text{false}$ for any Thy-truth function $f$.

The relation of Thy-logical implication is an equivalence relation on formulae, and so we will use the expression “Thy-logical equivalence” as per the next definition.

**Definition 3.2.9.** If Thy is a theory, then the formula $\alpha$ is **Thy-logically equivalent**, or Thy-**equivalent**, to the formula $\beta$ if $\alpha \equiv_{\text{Thy}} \beta$ and $\beta \equiv_{\text{Thy}} \alpha$. In this case we write $\alpha \equiv_{\text{Thy}} \beta$. If $\text{Thy} = \emptyset$ then we say “$\alpha$ is logically equivalent to $\beta$” instead of “$\alpha$ is $\emptyset$-logically equivalent to $\beta$”, and we write $\alpha \equiv \beta$ instead of $\alpha \equiv_{\emptyset} \beta$. We write $\mathbf{[}\alpha\mathbf{]}_{\text{Thy}}$ for the equivalence class of $\alpha$ under Thy-logical equivalence. If $\text{Thy} = \emptyset$ we write $\mathbf{[}\alpha\mathbf{]}_{\emptyset}$.

The relation of Thy-logical implication is not only an equivalence relation but also a congruence relation:

**Proposition 3.2.10.** Thy-logical equivalence is a congruence relation with respect to the logical connectives. That is, if $X$ and $X'$ are finite sets of formulae such that
Propositional logic and Boolean algebras

\[ X' = \{ \xi' \mid \xi \in X \} \text{ and for each } \xi \text{ in } X, \text{ we have that } \xi' \equiv_{\text{Thy}} \xi; \text{ and } \alpha \text{ and } \alpha' \text{ are formulae with } \alpha \equiv_{\text{Thy}} \alpha', \text{ then} \]

\[ \begin{align*}
\neg \alpha & = \neg \alpha'_{\text{Thy}} \\
\bigvee \{ \xi \mid \xi \in X \} & = \bigvee \{ \xi' \mid \xi' \in X' \}_{\text{Thy}} \\
\bigwedge \{ \xi \mid \xi \in X \} & = \bigwedge \{ \xi' \mid \xi' \in X' \}_{\text{Thy}}.
\end{align*} \]

**Proof.** Any Thy-truth function \( f \) preserves the connectives. Thence

\[ \begin{align*}
f(\neg \alpha) & = f(\neg \alpha') \\
f\left( \bigvee \{ \xi \mid \xi \in X \} \right) & = f\left( \bigvee \{ \xi' \mid \xi' \in X' \} \right) \\
f\left( \bigwedge \{ \xi \mid \xi \in X \} \right) & = f\left( \bigwedge \{ \xi' \mid \xi' \in X' \} \right).
\]

\[ \square \]

The above proposition tells us that the Thy-truth functions and logical connectives can be considered to act on the Thy-equivalence classes and we can consider the syntax and semantics to act on the equivalence classes of formulae instead of on the formulae themselves. That is, if \( \alpha \) and \( \beta \) are formulae and \( f \) is a Thy-truth function, then we can unambiguously write

\[ f([\alpha]_{\text{Thy}}) \text{ instead of } f(\alpha); \]
\[ \neg[\alpha]_{\text{Thy}} \text{ instead of } \neg[\alpha]_{\text{Thy}}; \]
\[ [\alpha]_{\text{Thy}} \lor [\beta]_{\text{Thy}} \text{ instead of } [\alpha \lor \beta]_{\text{Thy}}; \]
\[ [\alpha]_{\text{Thy}} \land [\beta]_{\text{Thy}} \text{ instead of } [\alpha \land \beta]_{\text{Thy}}; \text{ and} \]
\[ [\alpha]_{\text{Thy}} \Rightarrow [\beta]_{\text{Thy}} \text{ instead of } \alpha \Rightarrow_{\text{Thy}} \beta. \]

We will also write \( \square \) instead of \( [\square]_{\text{Thy}} \); and \( \top \) instead of \( [\top]_{\text{Thy}} \).

We will see in Section 3.3 that the set of Thy-equivalence classes forms a Boolean algebra, although PropForm does not. We give a name to the set of Thy-equivalence classes in the following definition.

**Definition 3.2.11.** Let \( \text{LE}_{\text{Thy}}(\text{PropAtom}) \) be the set of equivalence classes of Thy-logically equivalent formulae on the propositional atoms PropAtom. We will write \( \text{LE}_{\text{Thy}} \) instead of \( \text{LE}_{\text{Thy}}(\text{PropAtom}) \) when the context is clear.

Having defined the syntax and the semantics of the logic \( \text{Prop}(\text{PropAtom}, \text{Thy}) \), we are ready to define its third component, namely a deduction function. The deduction function we will define will depend on a special type of formula called a “clause”, which is the topic of the next section.
3.2.3 Clauses

In this section we define clauses and some special terms and properties related to clauses. Clauses are built up from literals, and positive clauses are special types of clauses:

Definition 3.2.12.

A literal is a propositional atom or a negated propositional atom (of form \( p \) or \( \neg p \)).

The negation of the literal \( \ell \) is written \( \ell' \). That is, if \( \ell \) is \( p \) or \( \neg p \) then \( \ell' \) is \( \neg p \) or \( p \) respectively.

A clause is a disjunction (\( \lor \)) of literals. That is, a clause is a formula of the form

\[
\ell_1 \lor \cdots \lor \ell_n,
\]

where \( n \geq 0 \), and \( \ell_1, \ldots, \ell_n \) are literals. If \( n = 0 \) then \( \ell_1 \lor \cdots \lor \ell_n \), previously introduced as \( \Box \) and called “box”, is also called the empty clause.

We will write Clauses for the set of clauses.

A positive clause is a disjunction of atoms. That is, a positive clause is a formula of the form

\[
p_1 \lor \cdots \lor p_n,
\]

where \( n \geq 0 \) and \( p_1, \ldots, p_n \) are propositional atoms.

Clauses are of interest because each formula can be broken down as a conjunction of clauses.

Lemma 3.2.13. Each formula is logically equivalent to some conjunction of clauses. That is, to each formula \( \alpha \), there are clauses \( \gamma_1, \ldots, \gamma_n \), where \( n \geq 0 \), such that

\[
\alpha \text{ is logically equivalent to } \gamma_1 \lor \cdots \lor \gamma_n.
\]


Note that in the above lemma, if Thy is a theory then the formula \( \alpha \) is also Thy-logically equivalent to \( \gamma_1 \lor \cdots \lor \gamma_n \).

We will also be interested in relationships between clauses as follows.

Definition 3.2.14. The clause \( \alpha \) subsumes, or is a subset of, the clause \( \beta \), written \( \alpha \subseteq \beta \), if every literal in \( \alpha \) appears in \( \beta \). That is,

\[
\beta = \alpha \lor \ell_1 \lor \cdots \lor \ell_n,
\]

where \( n \geq 0 \) and \( \ell_1, \ldots, \ell_n \) are literals.

If \( \alpha = \ell'_1 \lor \cdots \lor \ell'_m \) where \( m \geq 1 \) and \( \ell'_1, \ldots, \ell'_m \) are literals, then we write \( \ell'_i \in \alpha \) for \( i = 1, \ldots, m \).

Having defined clauses, we are ready to define deduction functions for clauses.
3.2.4 Deduction

In general, a deduction function takes as input a set of formulae and gives as output a set of formulae. However since each formula is logically equivalent to a conjunction of clauses, we will work with clauses rather than formulae. We will define deduction functions in terms of clauses as follows:

**Definition 3.2.15.** A **deduction function** is a function

\[ D : \mathcal{P}(\text{Clauses}) \to \mathcal{P}(\text{Clauses}). \]

If \( \Sigma \) is a set of clauses and \( \sigma \) is a clause, then the relation \( \sigma \in D(\Sigma) \) is also written \( \Sigma \vdash_D \sigma \).

We will also allow deduction functions on subsets of Clauses. That is if \( X, Y \subseteq \text{Clauses} \) then we allow deduction functions of form \( D : \mathcal{P}(X) \to \mathcal{P}(Y) \).

Note that the above definition of a deduction function is a generalisation of a consequence operator, defined by Tarski (1929).

In Chapter 4 we will use a deduction function called “resolution”. The definition of resolution depends on the “resolvent” of two clauses, given in the next definition. In the subsequent definition we give an inductive definition of resolution.

**Definition 3.2.16.** Let \( p \) be a propositional atom and let \( \ell_1, \ldots, \ell_m \) and \( \ell'_1, \ldots, \ell'_n \) be literals. Then the **resolvent** on \( p \) of the two clauses \( p \lor \ell_1 \lor \cdots \lor \ell_m \) and \( \neg p \lor \ell'_1 \lor \cdots \lor \ell'_n \) is \( \ell_1 \lor \cdots \lor \ell_m \lor \ell'_1 \lor \cdots \lor \ell'_n \). If \( m = 0 \) then the resolvent is \( \ell'_1 \lor \cdots \lor \ell'_n \). If \( n = 0 \) then the resolvent is \( \ell_1 \lor \cdots \lor \ell_m \). If \( m = n = 0 \) then the resolvent is \( \Box \).

We can also say that “we resolve \( p \lor \ell_1 \lor \cdots \lor \ell_m \) with \( \neg p \lor \ell'_1 \lor \cdots \lor \ell'_n \) on \( p \) to get \( \ell_1 \lor \cdots \lor \ell_m \lor \ell'_1 \lor \cdots \lor \ell'_n \)”.

**Definition 3.2.17.** Given a theory \( \text{Thy} \), the **\( \text{Thy} \)-resolution deduction function** is

\[ R_{\text{Thy}} : \mathcal{P}(\text{Clauses}) \to \mathcal{P}(\text{Clauses}), \]

where for each set \( \Sigma \) of clauses the set \( R_{\text{Thy}}(\Sigma) \) is defined inductively as follows:

1. \( \Sigma \cup \text{Thy} \subseteq R_{\text{Thy}}(\Sigma) \).

2. If \( \sigma_1, \sigma_2 \in R_{\text{Thy}}(\Sigma) \) and \( \sigma \) is a resolvent of \( \sigma_1 \) and \( \sigma_2 \), then \( \sigma \in R_{\text{Thy}}(\Sigma) \).

If \( \text{Thy} = \emptyset \), then we write \( R \) instead of \( R_{\emptyset} \). As before for \( D \), instead of writing \( \sigma \in R_{\text{Thy}}(\Sigma) \), we can write \( \Sigma \vdash_{R_{\text{Thy}}} \sigma \), or, where the meaning is clear, simply \( \Sigma \vdash_{\text{Thy}} \sigma \).

Note 1: If \( \text{Thy} \) is non-empty then for any set \( \Sigma \) of clauses, the set \( R_{\text{Thy}}(\Sigma) \) is also non-empty because \( \text{Thy} \subseteq R_{\text{Thy}}(\Sigma) \).

Note 2: \( R(\emptyset) = \emptyset \); and \( R_{\text{Thy}}(\Sigma) = R(\Sigma \cup \text{Thy}) \).

An equivalent definition of the function \( R_{\text{Thy}} \) is in terms of sequences of clauses, as stated in the next proposition.
Proposition 3.2.18. Let $\Sigma$ be a set of clauses and let $\sigma$ be a clause. Then $\sigma \in R_{Thy}(\Sigma)$ if and only if there is a finite sequence of clauses $\sigma_1, \ldots, \sigma_n = \sigma$ such that for each $i = 1, \ldots, n$ at least one of the following holds:

1. $\sigma_i \in \Sigma \cup Thy$;
2. there exist $j, k < i$ such that $\sigma_i$ is a resolvent of clauses $\sigma_j$ and $\sigma_k$.

Proof. Apply Theorem 1.9.7 with:

1. $S = \text{Clauses}$;
2. $B = \Sigma \cup Thy$; and
3. the relation $R$ on $\mathcal{P}(\text{Clauses}) \times \text{Clauses}$ defined as follows: $(X, \beta) \in R$ if and only if $X$ consists of exactly two clauses whose resolvent is $\beta$.

Nerode and Shore (1997, pages 53 and 61) also give an outline of the proof.

Terminology. Let $\Sigma$ be a set of clauses and let $Thy$ be a theory. Let $s = (\sigma_1, \ldots, \sigma_n)$ be a sequence of clauses with the properties given in Proposition 3.2.18. Then we say that the sequence $s$ is an $R_{Thy}$-deduction of $\sigma_n$ from $\Sigma$. We also say that $s$ is a Thy-resolution deduction of $\sigma_n$ from $\Sigma$. For each $i = 1, \ldots, n$, we say that $\sigma_i$ is obtained by one step of $R_{Thy}$-deduction. We also say that $\sigma_i$ is obtained by one step of Thy-resolution [deduction]. We refer to the set $\Sigma$ as the input set of clauses for the $R_{Thy}$-deduction, or alternatively as the starting set of clauses for the $R_{Thy}$-deduction. We refer to the clause $\sigma_n$ as an output clause of an $R_{Thy}$-deduction. We also say that the clause $\sigma_n$ is the final clause of an $R_{Thy}$-deduction, or the end result of an $R_{Thy}$-deduction, or the resultant clause of an $R_{Thy}$-deduction. We also say that an $R_{Thy}$-deduction results in $\sigma_n$.

The following deduction function, called “ordered resolution”, is derived from the resolution deduction function.

Definition 3.2.19. Given a theory $Thy$, and a total ordering $\prec$ on the set $\text{PropAtom}$ of propositional atoms, the ordered Thy-resolution deduction function is

$$R_{Thy}^\prec : \mathcal{P}(\text{Clauses}) \to \mathcal{P}(\text{Clauses}),$$

where for each set $\Sigma$ of clauses the set $R_{Thy}^\prec(\Sigma)$ is defined inductively as follows:

1. $\Sigma \cup Thy \subseteq R_{Thy}^\prec(\Sigma)$.
2. If $\sigma_1, \sigma_2 \in R_{Thy}^\prec(\Sigma)$, and $\sigma$ is a resolvent of $\sigma_1$ and $\sigma_2$ on the propositional atom $p$, and $p$ has lower order than any propositional atom in $\sigma$, then $\sigma \in R_{Thy}^\prec(\Sigma)$.

If $Thy = \emptyset$, then we write $R^\prec$ instead of $R_{\emptyset}^\prec$.

Another name for ordered resolution is directional resolution (Dechter and Rish 1994).
Note: $R_{\text{Thy}}^\prec (\Sigma) = R^\prec (\Sigma \cup \text{Thy})$.

**Lemma 3.2.20.** Let $\Sigma$ be a set of clauses; let $\prec$ be a total ordering on the set of propositional atoms; and let $\sigma$ be a clause. Then $\sigma \in R_{\text{Thy}}^\prec (\Sigma)$ if and only if there is a finite sequence of clauses $\sigma_1, \ldots, \sigma_n = \sigma$ such that for each $i = 1, \ldots, n$ at least one of the following holds:

1. $\sigma_i \in \Sigma \cup \text{Thy}$;

2. there exist $j, k < i$ such that $\sigma_i$ is a resolvent of clauses $\sigma_j$ and $\sigma_k$ on some propositional atom which has lower order than any propositional atom in $\sigma_i$.

**Proof.** Apply Theorem 1.9.7 with:

1. $S = \text{Clauses}$;

2. $B = \Sigma \cup \text{Thy}$; and

3. the relation $R$ on $\mathcal{P}(\text{Clauses}) \times \text{Clauses}$ defined as follows: $(X, \beta) \in R$ if and only if $X$ consists of exactly two clauses whose resolvent is $\beta$ and the resolution is taken on a propositional atom that has lower order than any atom in $\beta$.

We have defined deduction functions, which take sets of clauses as input and give sets of clauses as output. Two particular deduction functions which will be of interest in Chapter 4 are resolution and ordered resolution. We are now ready to compare deduction to logical implication.

### Relationship between deduction and logical implication

We have seen two ways of describing whether a given clause $\alpha$ “follows from” a given set $\Sigma$ of clauses. In Section 3.2.2 we took a semantic approach and defined the case when $\alpha$ is Thy-logically implied by $\Sigma$, using the Thy-truth functions. In Section 3.2.4 we took a syntactic approach and defined the case when $\alpha$ is a deduced clause from $\Sigma$, using the deduction function $D$.

In this section we discuss the relationship between Thy-logical implication and the deduction function $D$. When every deduced clause using the function $D$ is also a Thy-logically implied clause, then we say that the deduction function $D$ is “strongly Thy-sound”, defined precisely below in Definition 3.2.21 (although strong soundness is usually just called soundness). In the converse situation, when every Thy-logically implied clause is also a deduced clause using $D$, we say that $D$ is “strongly Thy-complete”, defined precisely below in Definition 3.2.24. We will also define weaker versions of Thy-soundness and Thy-completeness, called “refutation Thy-soundness” and “refutation Thy-completeness”, in Definitions 3.2.22 and 3.2.25.
§3.2  Propositional logic

**Definition 3.2.21.** The deduction function $\mathcal{D}$ is **strongly** Thy-**sound** if for all sets $\Sigma$ of clauses and all clauses $\sigma$ the following statement holds:

\[
\text{if } \Sigma \vdash_{\mathcal{D}} \sigma \text{ then } \Sigma \models_{\text{Thy}} \sigma.
\]

If $\text{Thy} = \emptyset$, then we say that “$\mathcal{D}$ is strongly sound” instead of “$\mathcal{D}$ is strongly $\emptyset$-sound”.

Although strong soundness is usually called just “soundness”, we use the longer name because we will also consider another, unusual, type of soundness called “refutation soundness”, defined below in Definition 3.2.22. We will therefore not use the term “soundness” on its own. From the definition it will be apparent that a function that is refutation sound but not strongly sound seems of no use, as in Example 3.2.23 after the definition. However, since we will be dealing with deduction functions in general, we will allow at least the discussion of functions that are refutation sound but not strongly sound.

While strong Thy-soundness means that every deduced clause using $\mathcal{D}$ is also a Thy-logically implied clause, refutation Thy-soundness applies to just the one deduced clause $\Box$, that is, refutation Thy-soundness means that if $\Box$ is a deduced clause using $\mathcal{D}$ then it is also a Thy-logically implied clause.

**Definition 3.2.22.** The deduction function $\mathcal{D}$ is **refutation** Thy-**sound** if for all sets $\Sigma$ of clauses the following statement holds:

\[
\text{if } \Sigma \vdash_{\mathcal{D}} \Box \text{ then } \Sigma \models_{\text{Thy}} \Box.
\]

If $\text{Thy} = \emptyset$, then we say that “$\mathcal{D}$ is refutation sound” instead of “$\mathcal{D}$ is refutation $\emptyset$-sound”.

The following is an example of a seemingly useless deduction function that is refutation sound but not strongly sound.

**Example 3.2.23.** Let $p$ be a propositional atom. Define the deduction function $\mathcal{D}$ for each clause set $X$ by:

\[\mathcal{D}(X) = X \cup \{p\}.\]

Then $\mathcal{D}$ is refutation sound but is not strongly sound: It is refutation sound because if $X \vdash_{\mathcal{D}} \Box$, then by the definition of $\mathcal{D}$ we have that $\Box \in X$, so that $X \models \Box$. It is not strongly sound, because for example if $X = \{q\}$ and $q \neq p$ then $X \vdash_{\mathcal{D}} p$ but $X \not\models p$.

Having defined strong Thy-soundness and refutation Thy-soundness, we now define the converses, namely strong Thy-completeness and refutation Thy-completeness.

**Definition 3.2.24.** The deduction function $\mathcal{D}$ is **strongly** Thy-**complete** if for all sets $\Sigma$ of clauses and all clauses $\sigma$ the following statement holds:

\[
\text{if } \Sigma \models_{\text{Thy}} \sigma \text{ then } \Sigma \vdash_{\mathcal{D}} \sigma.
\]

If $\text{Thy} = \emptyset$, then we say that “$\mathcal{D}$ is strongly complete” instead of “$\mathcal{D}$ is strongly $\emptyset$-complete”.
In contrast to the term “strong soundness”, the term “strong completeness” is not usually replaced by just “completeness”, because in general there is a need to distinguish it from another type of completeness called “refutation completeness”, defined below in Definition 3.2.25. Similarly to refutation Thy-soundness, refutation Thy-completeness applies to just the one clause $\square$, that is, refutation Thy-completeness means that if $\square$ is a Thy-logically implied clause, then it is also a deduced clause using the deduction function $\mathcal{D}$.

**Definition 3.2.25.** The deduction function $\mathcal{D}$ is **refutation** Thy-complete if for all sets $\Sigma$ of clauses the following statement holds:

$$\text{if } \Sigma \vdash_{\text{Thy}} \square \text{ then } \Sigma \vdash_{\mathcal{D}} \square.$$ 

If $\text{Thy} = \emptyset$, then we say that “$\mathcal{D}$ is refutation complete” instead of “$\mathcal{D}$ is refutation $\emptyset$-complete”.

The following lemma gives equivalent definitions in terms of satisfiability for each of refutation Thy-soundness, refutation Thy-completeness, and strong Thy-soundness. As to the property of strong Thy-completeness, the lemma gives an implied property rather than an equivalent one.

**Lemma 3.2.26.** Let $\mathcal{D}$ be a deduction function.

1. $\mathcal{D}$ is refutation Thy-sound if and only if the following statement holds: For each set $\Sigma$ of clauses

   $$\Sigma \text{ is Thy-satisfiable } \Rightarrow \square \notin \mathcal{D}(\Sigma).$$

2. $\mathcal{D}$ is refutation Thy-complete if and only if the following statement holds: For each set $\Sigma$ of clauses

   $$\square \notin \mathcal{D}(\Sigma) \Rightarrow \Sigma \text{ is Thy-satisfiable}.$$

3. $\mathcal{D}$ is strongly Thy-sound if and only if the following statement holds: For each set $\Sigma$ of clauses and each Thy-truth function $f$

   $$f \text{ satisfies } \Sigma \Rightarrow f \text{ satisfies } \mathcal{D}(\Sigma). \quad (3.2.2)$$

4. If $\mathcal{D}$ is strongly Thy-complete then the following statement holds: For each set $\Sigma$ of clauses and each Thy-truth function $f$

   $$f \text{ satisfies } \mathcal{D}(\Sigma) \Rightarrow f \text{ satisfies } \Sigma. \quad (3.2.3)$$

**Proof.**

1. and 2. Follow from Statement 3.2.1 (page 61), Definition 3.2.22 and Definition 3.2.25.
3. **Forward direction.** Suppose that \( \mathcal{D} \) is strongly Thy-sound and \( f \) satisfies \( \Sigma \). Let \( \delta \in \mathcal{D}(\Sigma) \). Then \( \Sigma \vdash_{\text{Thy}} \delta \), by Thy-soundness. Then since \( f \) is a Thy-truth function that satisfies \( \Sigma \), we have that \( f \) satisfies \( \delta \). But \( \delta \) is any clause in \( \mathcal{D}(\Sigma) \). Hence \( f \) satisfies \( \mathcal{D}(\Sigma) \).

**Backward direction.** Assume that if \( f \) satisfies \( \Sigma \) then \( f \) satisfies \( \mathcal{D}(\Sigma) \). Suppose that \( \Sigma \vdash \alpha \). We will show that \( \Sigma \vdash_{\text{Thy}} \alpha \). Suppose that \( g \) is a Thy-truth function that satisfies \( \Sigma \). Then, by assumption, \( g \) satisfies \( \mathcal{D}(\Sigma) \). But \( \alpha \in \mathcal{D}(\Sigma) \), hence \( g \) satisfies \( \alpha \).

4. Suppose that \( f \) satisfies \( \mathcal{D}(\Sigma) \). Suppose that \( \sigma \in \Sigma \). Then \( \Sigma \vdash_{\text{Thy}} \sigma \). By strong Thy-completeness, \( \sigma \in \mathcal{D}(\Sigma) \). Hence \( f \) satisfies \( \sigma \). But this holds for any \( \sigma \) in \( \Sigma \). Hence \( f \) satisfies \( \Sigma \).

The converse of Lemma 3.2.26 part 4 does not hold. That is, there are deduction functions where Expression 3.2.3 holds, but \( \mathcal{D} \) is not strongly Thy-complete, as seen in the next example.

**Example 3.2.27.** Let \( \mathcal{D} \) be the resolution deduction function \( \mathcal{R} \), and let \( \text{Thy} \) be the empty set. Then for each clause set \( \Sigma \), we have that \( \mathcal{R}(\Sigma) \supseteq \Sigma \) and so Expression 3.2.3 in Lemma 3.2.26 part 4 holds. However \( \mathcal{R} \) is not strongly complete as seen in the following situation. Let \( p \) and \( q \) be distinct propositional atoms and let \( \Sigma = \{ p \} \). Then \( \Sigma \vdash p \lor q \), but \( \Sigma \nvdash \mathcal{R} p \lor q \).

Parts 3 and 4 of Lemma 3.2.26 cannot be modified by replacing strongly Thy-sound and strongly Thy-complete by refutation Thy-sound and refutation Thy-complete respectively, as seen in the next example.

**Example 3.2.28.**

1. This is an example of a deduction function \( \mathcal{D} \) that is refutation sound but does not satisfy Statement 3.2.2 of part 3 of Lemma 3.2.26. Let \( \alpha \) be a clause other than \( \Box \). Define the deduction function \( \mathcal{D} \) for each clause set \( \Sigma \) by

\[
\mathcal{D}(\Sigma) = \Sigma \cup \{ \alpha \}.
\]

2. This is an example of the deduction function \( \mathcal{D} \) that is refutation complete but does not satisfy Statement 3.2.3 of part 4 of Lemma 3.2.26. Define the deduction function \( \mathcal{D} \) for each clause set \( \Sigma \) by

\[
\mathcal{D}(\Sigma) = \{ \Box \}.
\]

We conclude this section by noting, firstly, that resolution and ordered resolution are strongly sound and refutation complete, and, secondly, that the proof of refutation completeness has parallels with the proof of the error correction guarantee of FH.

**Theorem 3.2.29.** \( \mathcal{R}_{\text{Thy}} \) is strongly Thy-sound and refutation Thy-complete.
Proof. There are many published proofs for the case when Thy = ∅, for example that of Buss (1998, page 19–21). The original proof was published by Robinson (1965). The dual, about the “consensus” function, was proved by Quine (1955). The proof for any theory Thy follows from the case when Thy = ∅:

Strong Thy-soundness: if \( \Sigma \vdash_{R_{Thy}} \delta \) then by definition \( \Sigma \cup \text{Thy} \vdash_{R} \delta \), and hence by the soundness of \( R \), we have that \( \Sigma \cup \text{Thy} \vdash_{R} \delta \).

Refutation Thy-completeness: if \( \Sigma \cup \text{Thy} \vdash \Box \) then by the completeness of \( R \) we have that \( \Sigma \cup \text{Thy} \vdash_{R} \Box \), and hence by definition \( \Sigma \vdash_{R_{Thy}} \Box \).

The main ideas in the proof of the refutation completeness of \( R \) are as follows. Given a set \( \Sigma \) of clauses, the proof depends on the construction of a sequence of clause sets \( \Sigma_0, \ldots, \Sigma_n \), where \( \Sigma_0 = \Sigma \), and \( \Sigma_n = \{ \Box \} \) or \( \emptyset \), and with the following properties:

1. If \( 1 \leq j \leq n \) then \( \Sigma_j \subseteq R(\Sigma_{j-1}) \).
2. If \( 1 \leq j \leq n \) and \( \Sigma_j \) is satisfiable is then \( \Sigma_{j-1} \) is satisfiable.

Then the refutation completeness of \( R \) follows from the following argument. If \( \Sigma = \Sigma_0 \) is unsatisfiable then, by repeated application of item 2, we have that \( \Sigma_n \) is also unsatisfiable. Then by definition \( \Sigma_n = \{ \Box \} \), so that by item 1, we have that \( \Box \in R(\Sigma) \).

The proof outlined above for the refutation completeness of \( R \) has strong parallels with the proof of the error correction guarantee of FH (Theorem 2.4.6). Both proofs depend on a sequence of sets (of clauses or edits) with a property related to satisfiability: in the case of \( R \), the property is given as item 2 in the above proof; while in the case of FH, the property is the lifting property (Definition 2.4.5). Note that item 2 of the above proof is a special case of the “reduction property for counter-examples” defined by Bachmair and Ganzinger (2001, page 38).

**Theorem 3.2.30.** \( R_{Thy}^{\succ} \) is strongly Thy-sound and refutation Thy-complete.

Proof. See for example the proof given by Nerode and Shore (1997, page 65). The proof of refutation completeness is similar to the proof of the refutation completeness of \( R \), using a sequence of clause sets with the reduction property for counter-examples.

In this section we have examined the relationship between deduction and logical implication in terms of soundness and completeness, both of which can also be related to the concept of satisfiability, which leads us to the next section about the propositional satisfiability problem.

### 3.2.6 The propositional satisfiability problem

The propositional satisfiability problem (known as SAT) is the problem of deciding whether a given set of propositional formulae is satisfiable. In general the set of formulae is replaced by a logically equivalent set of clauses.
Most algorithms for solving the propositional satisfiability problem use a combination of search and deduction, where search means systematically testing potential models, and deduction means finding the set $D(\Sigma)$ of clauses deduced from the given clause set $\Sigma$ using the deduction function $D$ from the given set $\Sigma$ of clauses. Descriptions of a large range of algorithms are given by Dechter (2003) and Gu et al. (1997) among many others.

A pure deduction method finds the set $D(\Sigma)$ of clauses deduced using the deduction function $D$ from the given set $\Sigma$ of clauses. If $D$ is chosen to be both refutation sound and refutation complete, then, by Lemma 3.2.26 parts 1 and 2, deciding whether the set $\Sigma$ is satisfiable is equivalent to observing whether $\square \not\in D(\Sigma)$. An example of a suitable deduction function $D$ is the resolution deduction function $R$, since it is both refutation sound and refutation complete. Such an algorithm was published by Davis and Putnam (1960), although it is not strictly a pure deduction method because it uses an additional rule called the “affirmative-negative rule” (also called the “pure-literal rule” (Chang and Lee 1973)). The algorithm has received less attention than the later algorithm of Davis, Putnam, Loveland and Logemann (Davis et al. 1962), discussed below. But the earlier Davis-Putnam algorithm was further developed by Dechter and Rish (Dechter and Rish 1994; Rish and Dechter 2000).

In contrast to pure deduction methods, search methods systematically test potential models of the given clause set. The best-known example of a search algorithm is the Davis-Putnam-Logemann-Loveland (DPLL) algorithm (Davis et al. 1962).

The DPLL algorithm uses the so-called “splitting rule”, which is based on the fact that any model of $\Sigma$ must assign each propositional atom to either true or false. The splitting rule chooses one literal $\ell$ and tests the two cases, of $\ell$ assigned to true or false, to see if either assignment can be expanded to a model of the clause set. Thus the splitting rule creates two new clause sets (defined below), the set $\Sigma^\ell$ for when $\ell$ is assigned to true, and the set $\Sigma^\overline{\ell}$ for when $\ell$ is assigned to false.

To calculate the sets $\Sigma^\ell$ and $\Sigma^\overline{\ell}$ we first assign the truth value true or false respectively to the literal $\ell$, then recalculate each clause $\sigma$ in $\Sigma$, and finally delete any of the recalculated clauses that are logically valid, as follows:

if $\ell$ is a literal and $\Sigma$ is a set of clauses, then $\Sigma^\ell = \{ \sigma \setminus \{\ell\} \mid \sigma \in \Sigma \text{ and } \ell \not\in \sigma \}$.

Since $\Sigma$ is satisfiable if and only if at least one of $\Sigma^\ell$ and $\Sigma^\overline{\ell}$ is satisfiable, the splitting rule allows us to replace the satisfiability problem for $\Sigma$ by the two smaller satisfiability problems, for $\Sigma^\ell$ and $\Sigma^\overline{\ell}$. If repeated application of the splitting rule gives the empty set of clauses then the original set $\Sigma$ is satisfiable. Otherwise it is unsatisfiable.

This concludes the background on logic needed in the rest of this thesis. The remainder of this chapter deals with the related topic of Boolean algebra.
3.3 Boolean algebra

In order to formalise edit generation functions as logical deduction functions, we need to
formalise the relationship between the domains of the two types of functions. The do-
main of any edit generation function is a set of edit sets (that is, a subset of $\mathcal{P} (\mathcal{P}(D))$),
while the domain of any logical deduction function is a set of sets of logical formulae
(that is a subset of $\mathcal{P} (\text{PropForm})$). Both domains can be related to Boolean algebras,
which we will use in Chapter 4 to formalise the relationship between edit generation
functions and logical deduction functions. This section gives some background about
Boolean algebras, starting with the definition and some immediate properties, in Sec-
 tion 3.3.1.

Not only can logical formulae be related to Boolean algebras, but logical truth
functions can be related to Boolean homomorphisms between certain Boolean alge-
bras. I give the definition of a Boolean homomorphism and some basic properties in
Section 3.3.2.

Since all problems in editing relate to finite data domains, we will consider only finite
cases of Boolean algebras. The finiteness assumption results in simpler results than
those usually published in textbooks about Boolean algebra. Therefore this section
should not be seen as an introduction to Boolean algebra in general. However, where
practical I give the general results, and state where finiteness is assumed.

In order to relate edit sets to logical formulae, we will use the finite case of Stone’s
Representation Theorem, which characterises finite Boolean algebras as the power sets
of finite sets. The theorem is presented in Section 3.3.4, along with some consequences.

Finally, also in Section 3.3.4, we characterise truth functions in terms of special
elements called Boolean atoms. The definition and basic properties of Boolean atoms
are given in Section 3.3.3.

We start, in the next section, with the definition and some basic properties.

3.3.1 Definition and immediate properties of Boolean algebras

There are many different equivalent definitions of Boolean algebras. The following is
the one first given by Huntington (1904) and used since then by many authors.

Definition 3.3.1. A Boolean algebra is a system

$$\mathfrak{A} = \langle A, +_\mathfrak{A}, \cdot_\mathfrak{A}, \overline{\mathfrak{A}}, 0_\mathfrak{A}, 1_\mathfrak{A} \rangle,$$

where $A$ is a set and where $+_\mathfrak{A}$ and $\cdot_\mathfrak{A}$ are binary operations on $A$; $\overline{\mathfrak{A}}$ is a unary
operation on $A$; $0_\mathfrak{A} \in A; 1_\mathfrak{A} \in A$; and the following conditions hold for all $x, y, z$ in $A$:

- **Commutativity:** $x +_\mathfrak{A} y = y +_\mathfrak{A} x$
  $x \cdot_\mathfrak{A} y = y \cdot_\mathfrak{A} x$

- **Distributivity:** $x +_\mathfrak{A} (y \cdot_\mathfrak{A} z) = (x +_\mathfrak{A} y) \cdot_\mathfrak{A} (x +_\mathfrak{A} z)$
  $x \cdot_\mathfrak{A} (y +_\mathfrak{A} z) = (x \cdot_\mathfrak{A} y) +_\mathfrak{A} (x \cdot_\mathfrak{A} z)$
identity: \( x + \_A 0_A = x \)  
\( x \cdot \_A 1_A = x \)

complementation: \( x \cdot \_A (\neg_A x) = 0_A \)  
\( x + \_A (\neg_A x) = 1_A \).

We define \( x \leq_A y \) to mean \( x + \_A y = y \). We will drop the subscript \( A \) whenever the context is clear.

The next example, about power sets, is of relevance to edits, because the set of all edits is the power set of the data domain.

**Example 3.3.2.** For any set \( X \), the power set \( \mathcal{P}(X) \) forms a Boolean algebra in two ways, which are duals:

1. \( \mathcal{P}_X = \langle \mathcal{P}(X), \cup, \cap, \sim, \emptyset, X \rangle \) where for \( S \in \mathcal{P}(X) \), the expression \( \sim S \) means the complement of \( S \) in \( X \) i.e. \( \sim S = X \setminus S \). The inequality \( \leq_{\mathcal{P}_X} \) means \( \subseteq \).

2. \( \mathcal{P}'_X = \langle \mathcal{P}(X), \cap, \cup, \sim, X, \emptyset \rangle \). The inequality \( \leq_{\mathcal{P}'_X} \) means \( \supseteq \).

By replacing the set \( X \) of the above example by the data domain \( D \), we see from the first part of the above example that the set of all edits forms a Boolean algebra, while from the second part we see that the set of the acceptance regions of all the edits also forms a Boolean algebra.

The next example, about the set of possible truth values, is of relevance to logic.

**Example 3.3.3.** The set \( 2 = \{ \text{true}, \text{false} \} \) forms a Boolean algebra

\[ 2 = \langle 2, \lor, \land, \neg, \text{false}, \text{true} \rangle, \]

with the usual meanings of \( \land, \lor, \text{and} \neg \) applied to \( \text{false} \) and \( \text{true} \).

I earlier foreshadowed that the set PropForm of all logical formulae can be related to Boolean algebras. The set PropForm itself does not form a Boolean algebra, because the 0 and the 1 are not unique. However the set \( \text{LE}_{Thy} \) of equivalence classes of formulae (defined in Definition 3.2.11) does form a Boolean algebra and has a special name as per the next definition.

**Definition 3.3.4.** Let \( Thy \) be a theory. The *Lindenbaum algebra* for \( Thy \) is the Boolean algebra formed from the set \( \text{LE}_{Thy} \) with the usual logical connectives, and is written

\[ \mathcal{L}E_{Thy} = \langle \text{LE}_{Thy}, \lor, \land, \neg, \Box, \top \rangle. \]

If \( \alpha \) and \( \beta \) are formulae, then the expression \( [\alpha]_{Thy} \leq_{\mathcal{L}E_{Thy}} [\beta]_{Thy} \) means \( \alpha \vdash_{Thy} \beta \).

The Lindenbaum algebra for \( Thy \) is a Boolean algebra because (1) Thy-logical equivalence is a congruence relation, and (2) the equivalence classes of propositional formulae under Thy-logical equivalence satisfy Definition 3.3.1.

The following lemma gives an equivalent definition of Boolean algebras.
Lemma 3.3.5. Let $\mathfrak{A}$ be the system

$$\mathfrak{A} = \langle A, +_{\mathfrak{A}}, \cdot_{\mathfrak{A}}, -_{\mathfrak{A}}, 0_{\mathfrak{A}}, 1_{\mathfrak{A}} \rangle,$$

where $A$ is a set and where $+_{\mathfrak{A}}$ and $\cdot_{\mathfrak{A}}$ are binary operations on $A$; $-_{\mathfrak{A}}$ is a unary operation on $A$; $0_{\mathfrak{A}} \in A$; and $1_{\mathfrak{A}} \in A$.

Then $\mathfrak{A}$ is a Boolean algebra if and only if it has the commutativity, distributivity, and complementation properties of Definition 3.3.1, as well as the following two properties:

- **Associativity:**
  $$x + (y + z) = (x + y) + z \quad \text{and} \quad x \cdot (y \cdot z) = (x \cdot y) \cdot z$$

- **Absorption:**
  $$x + (x \cdot y) = x \quad \text{and} \quad x \cdot (x + y) = x$$

**Proof.** There are many published proofs, including, for the forward direction that by Rueff and Jeger (1970, pages 57, 59), and for the backward direction that by Monk (1976, pages 142–143). The proof that associativity follows from Definition 3.3.1 was first published by Huntington (1904). ⊣

As a consequence of commutativity and associativity, we can use the symbols $\sum$ and $\prod$ for finitely repeated applications of the operations $+$ and $\cdot$ to obtain expressions such as $\sum\{x \mid x \in X\}$ and $\prod\{x \mid x \in X\}$.

We now turn to some basic properties of Boolean algebras which will be used later. In the next lemma we observe a basic property of $+$ and $\cdot$, as well as an equivalent definition of $\leq$.

Lemma 3.3.6. For the Boolean algebra $\mathfrak{A} = \langle A, +, \cdot, 0, 1 \rangle$ and elements $x, y$ of $A$ the following hold:

1. Idempotence: $x + x = x$ and $x \cdot x = x$.

2. Definition of $\leq$ in terms of $\cdot$: $x \leq y$ if and only if $x = x \cdot y$.

**Proof.** See for example the book by Koppelberg (1989, page 13). ⊣

The next lemma presents some basic properties of the inequality $\leq$. It is a partial order that makes the Boolean algebra into a lattice in which $+$ and $\cdot$ give the least upper bound and the greatest lower bound respectively:

Lemma 3.3.7. For the Boolean algebra $\mathfrak{A} = \langle A, +, \cdot, 0, 1 \rangle$ and elements $x, y, z$ of $A$ the following hold:

1. Partial order.
   
   (a) Reflexivity: $x \leq x$.

   (b) Anti-symmetry: If $x \leq y$ and $y \leq x$ then $x = y$.
(c) Transitivity: If \( x \leq y \) and \( y \leq z \) then \( x \leq z \).

2. \( + \) gives the least upper bound and \( \cdot \) gives the greatest lower bound.

(a) \( x \leq x + y \).
(b) If \( x \leq z \) and \( y \leq z \) then \( x + y \leq z \).
(c) \( x \cdot y \leq x \).
(d) If \( z \leq x \) and \( z \leq y \) then \( z \leq x \cdot y \).


We use the definitions of least upper bound and greatest lower bound to write \( \sum \{ x \mid x \in \emptyset \} = 0 \) and \( \prod \{ x \mid x \in \emptyset \} = 1 \).

Another important property of Boolean algebras is Stone’s Representation Theorem (Stone 1936), which in the finite case states that each finite Boolean algebra is “isomorphic” to some power set. We will state Stone’s Representation Theorem precisely in Section 3.3.4, but we first need to define isomorphisms and more generally homomorphisms, which we do in the next section. In order to be precise about the power set used in Stone’s Representation Theorem we will also need to define “Boolean atoms”, which we will do in the subsequent section, Section 3.3.3.

### 3.3.2 Boolean homomorphisms

In this section we define Boolean homomorphisms in two ways, and present a relationship between Boolean homomorphisms and truth functions. A Boolean homomorphism preserves all the structure of a Boolean algebra, as per the next definition.

**Definition 3.3.8.** Let \( \mathfrak{A} = \langle A, +, \cdot, -, 0, 1 \rangle \) and \( \mathfrak{A'} = \langle A', +', \cdot', -, 0', 1' \rangle \) be Boolean algebras. A function \( f : A \rightarrow A' \) is a **Boolean homomorphism** from \( \mathfrak{A} \) to \( \mathfrak{A'} \) if for all \( x, y \) in \( A \)

1. \( f(x + y) = f(x) +' f(y) \);
2. \( f(x \cdot y) = f(x) \cdot' f(y) \);
3. \( f(-x) = -' f(x) \);
4. \( f(0) = 0' \);
5. \( f(1) = 1' \).

We also write \( f : \mathfrak{A} \rightarrow \mathfrak{A'} \). If \( f \) is a bijection from \( A \) to \( A' \) then we say \( f \) is a **Boolean isomorphism**. We write \( \text{BoolHom}(\mathfrak{A}, \mathfrak{A'}) \) for the set of Boolean homomorphisms from \( \mathfrak{A} \) to \( \mathfrak{A'} \).

**Example 3.3.9.** Let \( X \) be a set. Let \( \mathfrak{P}_X \) and \( \mathfrak{P}'_X \) be the power set Boolean algebras defined in Example 3.3.2. Then the complement function \( c : \mathfrak{P}_X \rightarrow \mathfrak{P}'_X \) defined by \( S \subseteq X \Rightarrow c(S) = \sim S \) is a Boolean isomorphism.
The above definition of Boolean homomorphisms contains redundant requirements. The next lemma gives a shorter definition, in terms of just two requirements.

**Lemma 3.3.10.** Let $\mathfrak{A} = \langle A, +, \cdot, -, 0, 1 \rangle$ and $\mathfrak{A}' = \langle A', +', \cdot', -, 0', 1' \rangle$ be two Boolean algebras. Then $f : A \to A'$ is a Boolean homomorphism if and only if for all $x, y$ in $A$:

$$f(x + y) = f(x) +' f(y), \text{ and } f(-x) = -' f(x).$$

*Proof.* The operations $+$ and $-$ are enough to construct $\cdot$, 0 and 1. See for example the proof presented by Monk (1976, page 145).

We now consider the relationship between Boolean homomorphisms and Thy-truth functions. The Thy-truth functions themselves are not Boolean homomorphisms because their domain PropForm does not form a Boolean algebra. However they can be naturally related to Boolean homomorphisms from the Lindenbaum algebra $\mathcal{L}E_{Thy}$ to $\mathbb{2}$, because Thy-logical equivalence is a congruence relation and Thy-truth functions preserve $\lor, \land, \lnot, \text{false}$ and $\text{true}$. Conversely, the Boolean homomorphisms from $\mathcal{L}E_{Thy}$ to $\mathbb{2}$ can be considered to be Thy-truth functions because Thy-logical equivalence is a congruence relation. The next lemma gives a formal statement of the one-to-one relationship between Boolean homomorphisms and Thy-truth functions.

**Lemma 3.3.11.** Let $\text{ThyTF}$ be the set of Thy-truth functions on PropForm. Let

$$\text{BH} : \text{ThyTF} \to \text{BoolHom}(\mathcal{L}E_{Thy}, \mathbb{2})$$

be defined for each Thy-truth function $f$ and each equivalence class $[\alpha]_{Thy}$ of $\mathcal{L}E_{Thy}$ by

$$(\text{BH}(f))([\alpha]_{Thy}) = f(\alpha).$$

Then $\text{BH}$ is well-defined and a bijection.

*Proof.* The function $\text{BH}$ is well-defined because (1) if $\alpha \equiv_{Thy} \beta$ and $f$ is a Thy-truth function then $f(\alpha) = f(\beta)$, and (2) for each Thy-truth function $f$, the function $\text{BH}(f)$ is a homomorphism because Thy-logical equivalence is a congruence relation.

The function $\text{BH}$ is a bijection because its inverse is

$$\text{TF} : \text{BoolHom}(\mathcal{L}E_{Thy}, \mathbb{2}) \to \text{ThyTF},$$

defined for each Boolean homomorphism $g$ in $\text{BoolHom}(\mathcal{L}E_{Thy}, \mathbb{2})$ and each propositional formula $\alpha$ by:

$$(\text{TF}(g))(\alpha) = g([\alpha]_{Thy}).$$

The function $\text{TF}(g)$ is a Thy-truth function because Thy-logical equivalence is a congruence relation.
The above lemma tells us that there is a one-to-one correspondence between the Boolean homomorphisms $\text{BoolHom}(\mathcal{LE}_{\text{Thy}}, 2)$ and the Thy-truth functions. Therefore, for any Boolean homomorphism $f : \mathcal{LE}_{\text{Thy}} \to 2$ we will ambiguously but comprehensively write $f$ instead of $\text{TF}(f)$ for the corresponding Thy-truth function.

We will see in Section 3.3.4 that the set $\text{BoolHom}(\mathcal{LE}_{\text{Thy}}, 2)$ is in one-to-one correspondence with the set of so-called “Boolean atoms” of $\mathcal{LE}_{\text{Thy}}$. Thus the Thy-truth functions are also in one-to-one correspondence with the Boolean atoms. We first need to introduce Boolean atoms, in the next section.

### 3.3.3 Boolean atoms

In this section we define Boolean atoms and give some of their basic properties.

#### Definition 3.3.12
Let $\mathfrak{A} = (A, +, \cdot, -, 0, 1)$ be a Boolean algebra. An element $a$ of $A$ is a Boolean atom of $\mathfrak{A}$ under the following conditions:

1. $a \neq 0$; and

2. if $x \in A$ and $x \leq a$ then $x = a$ or $x = 0$.

We will write $\text{BoolAtm}(\mathfrak{A})$ for the set of Boolean atoms of $\mathfrak{A}$.

Note that in Lindenbaum algebras, the Boolean atoms are usually different from the propositional atoms, as will be seen in parts 4 and 5 of the next example, which also gives the Boolean atoms of some other Boolean algebras.

#### Example 3.3.13.

1. Given a set $X$, the set of Boolean atoms of the power set algebra $\mathcal{P}_X$ is $\{\{x\} \mid x \in X\}$.

2. The set of Boolean atoms of the dual power set algebra $\mathcal{P'}_X$ is $\{\sim\{x\} \mid x \in X\}$.

3. The set of Boolean atoms of $2$ is $\{1\}$.

4. Consider the propositional logic $\text{Prop}(\{p_1, \ldots, p_n\}, \emptyset)$, with just a finite set of propositional atoms and an empty theory. The set of Boolean atoms of the corresponding Lindenbaum algebra $\mathcal{LE}_\emptyset$ is

   $$\{[\ell_1 \land \cdots \land \ell_n] \mid \text{for each } j = 1, \ldots, n, \text{ either } \ell_j = p_j \text{ or } \ell_j = \neg p_j\}.$$

5. The Lindenbaum algebra for the propositional logic $\text{Prop}(\{p_1, p_2, \ldots\}, \emptyset)$, with infinitely many propositional atoms and an empty theory, has no Boolean atoms.

The following are some immediate consequences of the definition of Boolean atoms.

#### Proposition 3.3.14
Let $\mathfrak{A} = (A, +, \cdot, -, 0, 1)$ be a Boolean algebra. Let $a$ be a Boolean atom of $\mathfrak{A}$ and let $x$ and $y$ be in $A$. Then the following hold:
1. If $a \leq x + y$, then and only if $a \leq x$ or $a \leq y$.

2. $a \leq x$ or $a \leq -x$ but not both.

3. If $A$ is finite and $x \neq 0$, then there is a Boolean atom $b$ such that $b \leq x$. (If the condition that $A$ be finite is dropped and the property still holds then $\mathfrak{A}$ is called “atomic”.)

4. If $A$ is finite then $x = \sum \{b \mid b \in \text{BoolAtm}(\mathfrak{A}) \text{ and } b \leq x\}$. (This property applies not just to finite Boolean algebras but also to atomic Boolean algebras.)

Proof. See for example the proofs of parts 1 and 2 given by Koppelberg (1989, page 29), and the proofs of parts 3 and 4 given by Monk (1976, pages 151–152).

Having defined Boolean atoms and given some of the basic properties, we are now ready to examine the two characterisation results.

### 3.3.4 Characterisations

This section contains two results characterising aspects of Boolean algebras. The first result, which is the finite case of Stone’s Representation Theorem, characterises each finite Boolean algebra as a power set Boolean algebra. The second result states that the Thy-truth functions (for Thy a logical theory for a finite propositional logic) are in one-to-one correspondence with the Boolean atoms of the Lindenbaum algebra, although I present it in greater generality by replacing Lindenbaum algebras by arbitrary finite Boolean algebras.

**Proposition 3.3.15 (Finite case of Stone’s Representation Theorem (Stone 1936)).** If $\mathfrak{A}$ is a finite Boolean algebra, and $\mathfrak{P}_{\text{BoolAtm}(\mathfrak{A})}$ is the power set Boolean algebra defined in Example 3.3.2, then the function

$$h_\mathfrak{A} : \mathfrak{A} \to \mathfrak{P}_{\text{BoolAtm}(\mathfrak{A})}$$

defined for each $x$ in $\mathfrak{A}$ by

$$h_\mathfrak{A}(x) = \{a \mid a \in \text{BoolAtm}(\mathfrak{A}) \text{ and } a \leq x\}$$

is a Boolean isomorphism.

Proof. The proof of the dual of the general case was first published by Stone (1936). Books that present proofs include those by Monk (1976, pages 151–152) and by Koppeleberg (1989, pages 29–30).

Note that the inverse of $h_\mathfrak{A}$ is given for each $Y$ in $\mathfrak{P}_{\text{BoolAtm}(\mathfrak{A})}$ by $h_\mathfrak{A}^{-1}(Y) = \sum Y$. 

using the following calculation:

\[ h^{-1}_\mathfrak{A}(Y) = h^{-1}_\mathfrak{A}\left(\bigcup\{\{y\} \mid y \in Y\}\right) \]

\[ = \sum\left\{h^{-1}_\mathfrak{A}(\{y\}) \mid y \in Y\right\}, \text{ since } h^{-1}_\mathfrak{A} \text{ is a Boolean homomorphism} \]

\[ = \sum\{y \mid y \in Y\}, \text{ by the definition of } h_\mathfrak{A}, \text{ and since } Y \subseteq \text{BoolAtm}(\mathfrak{A}). \]

A consequence of the above proposition is that all finite Boolean algebras with the same number of Boolean atoms are in effect the same:

**Corollary 3.3.16.** If two finite Boolean algebras have the same number of Boolean atoms then they are isomorphic as Boolean algebras.

**Example 3.3.17.** \(\mathcal{P}_X\) and \(\mathcal{P}'_X\), defined in Example 3.3.2, have the same number of Boolean atoms and so are isomorphic as Boolean algebras.

Another consequence of the above proposition is that the homomorphisms between finite Boolean algebras are uniquely determined by the mappings between their respective sets of Boolean atoms, as given in the next proposition.

**Proposition 3.3.18.** Suppose that \(\mathfrak{A}\) and \(\mathfrak{A}'\) are finite Boolean algebras and that the function

\[ f : \text{BoolAtm}(\mathfrak{A}) \to \text{BoolAtm}(\mathfrak{A}') \]

is a bijection. Then there is a unique Boolean isomorphism

\[ f^* : \mathfrak{A} \to \mathfrak{A}' \]

that extends \(f\). That is, if \(a \in \text{BoolAtm}(\mathfrak{A})\) then \(f^*(a) = f(a)\).

**Proof.** If the Boolean homomorphism \(f^*\) exists, then for each \(x\) in \(\mathfrak{A}\) we can calculate \(f^*(x)\) in terms of the function \(f\) as follows:

\[ f^*(x) = f^*\left(\sum\{b \mid b \in \text{BoolAtm}(\mathfrak{A}) \text{ and } b \leq x\}\right), \text{ by Proposition 3.3.14, part 4} \]

\[ = \sum\{f^*(b) \mid b \in \text{BoolAtm}(\mathfrak{A}) \text{ and } b \leq x\}, \text{ since } f^* \text{ is a homomorphism} \]

\[ = \sum\{f(b) \mid b \in \text{BoolAtm}(\mathfrak{A}) \text{ and } b \leq x\}, \text{ since } f^* \text{ extends } f. \]

Thus, if the function \(f^*\) defined by the above calculation is indeed a Boolean homomorphism, then it is unique.

In order to see that the function \(f^*\) is a Boolean isomorphism, we relate it to another Boolean isomorphism: the function

\[ \mathcal{P}_f : \mathcal{P}_{\text{BoolAtm}(\mathfrak{A})} \to \mathcal{P}_{\text{BoolAtm}(\mathfrak{A}')} \]

is defined for each subset \(X\) of \(\text{BoolAtm}(\mathfrak{A})\) by

\[ \mathcal{P}_f(X) = \{f(x) \mid x \in X\} \]
and is a Boolean isomorphism because the function \( f \) is a bijection. We write the function \( f^* \) in terms of \( \mathcal{P}_f \) as follows:

\[
f^*(x) = \sum \{ f(b) \mid b \in \text{BoolAtm}(\mathfrak{A}) \text{ and } b \leq x \}, \quad \text{from above}
\]

\[
= \sum \{ f(b) \mid b \in h_{\mathfrak{A}}(x) \}, \quad \text{by the definition of } h_{\mathfrak{A}} \text{ in Proposition 3.3.15}
\]

\[
= h^{-1}_{\mathfrak{A}} \circ \mathcal{P}_f \circ h_{\mathfrak{A}}(x), \quad \text{by the definition of } h_{\mathfrak{A}}', \text{ which is an isomorphism by Proposition 3.3.15.}
\]

Thus \( f^* \) is the product of three Boolean isomorphisms, and so \( f^* \) is also a Boolean isomorphism. \( \dashv \)

The following is a simple application of the above proposition.

**Example 3.3.19.** The complement function \( c : \mathcal{P}_X \to \mathcal{P}_X' \) is the unique Boolean isomorphism that extends the function \( f : \text{BoolAtm}(\mathcal{P}_X) \to \text{BoolAtm}(\mathcal{P}_X') \) where if \( x \in X \) then \( f(\{x\}) = \sim\{x\} \).

We now turn to the other characterisation, of Thy-truth functions, where Thy is a logical theory for a finite propositional logic. For each Thy-truth function we will characterise the corresponding Boolean homomorphism from \( \mathcal{L}E_{\text{Thy}} \) to \( \mathcal{Z} \). Indeed, since we deal only with finite sets of propositional atoms, we will generalise and replace \( \mathcal{L}E_{\text{Thy}} \) by an arbitrary finite Boolean algebra \( \mathfrak{A} \). Thus we will consider a characterisation of Boolean homomorphisms from \( \mathfrak{A} \) to \( \mathcal{Z} \): the next proposition (Proposition / Definition 3.3.20) tells us that the Boolean homomorphisms from the finite Boolean algebra \( \mathfrak{A} \) to \( \mathcal{Z} \) are in one-to-one correspondence with the Boolean atoms of \( \mathfrak{A} \).

Proposition 3.3.20 follows from two results about the so-called “ultrafilters” of a Boolean algebra. The first result states that the ultrafilters of a Boolean algebra (finite or otherwise) are in one-to-one correspondence with the Boolean homomorphisms from that Boolean algebra to \( \mathcal{Z} \) (see for example the book by Koppelberg (1989, page 32)). The second result states that, in a finite Boolean algebra, the ultrafilters are in one-to-one correspondence with Boolean atoms. The two results together tell us that the Boolean homomorphisms from a Boolean algebra to \( \mathcal{Z} \) are in one-to-one correspondence with the Boolean atoms of that algebra, as required for Proposition / Definition 3.3.20.

Although the use of ultrafilters would give a general result, we are only interested in the finite situation. Therefore I present below a direct proof of Proposition / Definition 3.3.20 that takes advantage of the finiteness assumption and does not use ultrafilters. This saves introducing and proving basic properties of ultrafilters, which are not used anywhere else in this thesis.

**Proposition and Definition 3.3.20.** Suppose that \( \mathfrak{A} = \langle A, +, \cdot, - \rangle \) is a finite Boolean algebra. Let the function

\[
\chi : \text{BoolAtm}(\mathfrak{A}) \to \text{BoolHom}(\mathfrak{A}, \mathcal{Z})
\]

be defined for each \( a \) of \( \text{BoolAtm}(\mathfrak{A}) \) to be the “characteristic” function \( \chi(a) \), where
for each \( x \) in \( A \):

\[
(\chi(a))(x) = \begin{cases} 
  \text{true} & \text{if } a \leq x \\
  \text{false} & \text{otherwise.}
\end{cases}
\]

Then

1. for each \( a \) in \( \text{BoolAtm}({\mathfrak{A}}) \) the function \( \chi(a) \) is indeed a homomorphism; and

2. \( \chi \) is a bijection.

Proof.

1. Suppose that \( a \in \text{BoolAtm}({\mathfrak{A}}) \). In order to show that \( \chi(a) \) is a homomorphism, we use Lemma 3.3.10 which tells us that we need only show that \( \chi(a) \) preserves the operations + and −.

The fact that \( \chi(a) \) preserves the operation + follows from the following sequence of equivalent statements, where \( x, y \in A \):

\[
(\chi(a))(x + y) = \text{true} \\
\Leftrightarrow a \leq x + y, \quad \text{by the definition of } \chi(a) \\
\Leftrightarrow a \leq x \text{ or } a \leq y, \quad \text{by Proposition 3.3.14 part 1, since } a \text{ is a Boolean atom} \\
\Leftrightarrow (\chi(a))(x) = \text{true} \text{ or } (\chi(a))(y) = \text{true}, \quad \text{by the definition of } \chi(a) \\
\Leftrightarrow (\chi(a))(x) \lor (\chi(a))(y) = \text{true}.
\]

The fact that \( \chi(a) \) preserves the operation − follows from the following sequence of equivalent statements, where \( x \in A \):

\[
(\chi(a))(\neg x) = \text{true} \\
\Leftrightarrow a \leq \neg x, \quad \text{by the definition of } \chi(a) \\
\Leftrightarrow a \not\leq x, \quad \text{by Proposition 3.3.14 part 2} \\
\Leftrightarrow \neg(\chi(a)(x)) = \text{true}, \quad \text{by the definition of } \chi(a).
\]

Hence by Lemma 3.3.10, \( \chi(a) \) is a homomorphism.

2. We first show that \( \chi \) is one-one. Suppose \( a, b \in \text{BoolAtm}({\mathfrak{A}}) \) and that \( \chi(a) = \chi(b) \), so that for each \( x \) in \( A \), we have that \( (\chi(a))(x) = (\chi(b))(x) \). Hence by the definition of \( \chi \), we have that \( a \leq x \) if and only if \( b \leq x \). Let \( x = b \) and note that \( b \leq b \) by the reflexivity of \( \leq \): then we have that \( a \leq b \). Similarly, by letting \( x = a \), we have that \( b \leq a \). From the anti-symmetry of \( \leq \), we then have that \( a = b \), as required.

We now show that \( \chi \) is onto. Suppose that \( f \in \text{BoolHom}({\mathfrak{A}}, 2) \). We will find an \( a \) in \( \text{BoolAtm}({\mathfrak{A}}) \) such that \( \chi(a) = f \). We will first define \( a \) and will then show
that it has the required properties. Define

\[ F = \{ x \mid x \in A \text{ and } f(x) = \text{true} \} \]

and define

\[ a = \prod \{ x \mid x \in F \}, \]

which exists because \( A \) is finite.

We will show that \( a \) is a Boolean atom and \( \chi(a) = f \).

To show that \( a \) is a Boolean atom, we use Proposition 3.3.14, part 4, to write \( a \) as the sum of the Boolean atoms under it: if

\[ B = \{ b \mid b \in \text{BoolAtm}(A) \text{ and } b \leq a \} \]

then

\[ a = \sum \{ b \mid b \in B \}. \]

We now have two expressions for \( a \); taking \( f \) of both and using the fact that \( f \) is a Boolean homomorphism, we have that

\[ f(a) = \bigwedge \{ f(x) \mid x \in F \} = \bigvee \{ f(b) \mid b \in B \}, \]

which, by the definition of \( F \), we can write as

\[ f(a) = \text{true} = \bigvee \{ f(b) \mid b \in B \}. \]

Hence there is a \( b^* \) in \( B \) such that \( f(b^*) = \text{true} \). Hence by the definition of \( F \), we have that \( b^* \in F \). Then, since \( a = \prod \{ x \mid x \in F \} \), we have by Lemma 3.3.7 part 2c that \( a \leq b^* \). We also have that \( b^* \leq a \) because \( b^* \in B \) and \( b^* \leq \sum \{ b \mid b \in B \} = a \) by Lemma 3.3.7 part 2a. Hence by anti-symmetry, \( a = b^* \), which means that \( a \) is a Boolean atom, as required.

To show that \( \chi(a) = f \) we use the following sequence of equivalent statements for \( x \in A \):

\[
\begin{align*}
f(x) &= \text{true} \\
\Leftrightarrow x &\in F, \quad \text{by the definition of } F \\
\Leftrightarrow a &\leq x, \quad \text{by the definition of } a \text{ and because } f \text{ is a homomorphism} \\
\Leftrightarrow (\chi(a))(x) &= \text{true}, \quad \text{by the definition of } \chi(a).
\end{align*}
\]

Hence we have found a Boolean atom \( a \) such that \( \chi(a) = f \), and therefore \( \chi \) is onto.

\[ \square \]

The above proposition tells us that, for a finite Boolean algebra \( \mathfrak{A} \), each Boolean homomorphism \( f : \mathfrak{A} \to \mathbb{2} \) is of the form \( \chi(a) \) for a unique Boolean atom \( a \) of \( \mathfrak{A} \).
Indeed, the unique Boolean atom $a$ is exactly that atom on which $f$ is true:

**Corollary 3.3.21.** Suppose that $a$ is a Boolean atom of the Boolean algebra $\mathcal{A}$ and that the Boolean homomorphism $f: \mathcal{A} \rightarrow 2$ has $f(a) = \text{true}$. Then $f = \chi(a)$.

**Proof.** Since $\chi$ is a bijection, there is a Boolean atom $b$ such that $f = \chi(b)$. Then by the definition of $\chi$, since $f(a) = \text{true}$ we have that $b \leq a$. But $a$ is a Boolean atom, hence $b = 0$ or $b = a$. But $b$ is a Boolean atom, hence $b \neq 0$ and hence $b = a$. Hence $f = \chi(a)$, as required. $\dashv$

We can convert the above proposition and corollary to a statement about Thy-truth functions, by replacing $\mathcal{A}$ by $\mathcal{LE}_{Thy}$ for a finite set of propositional atoms:

**Corollary 3.3.22.** Suppose that Thy is a theory, and that PropAtom is a finite set. As in Lemma 3.3.11, we write ThyTF for the set of Thy-truth functions. Let the function $\chi: \text{BoolAtm}(\mathcal{LE}_{Thy}) \rightarrow \text{ThyTF}$ be defined for each Boolean atom $[\alpha]_{Thy}$ of $\mathcal{LE}_{Thy}$ and each propositional formula $\beta$ by

$$
\left(\chi([\alpha]_{Thy})\right)(\beta) = \begin{cases} 
\text{true} & \text{if } \alpha \models_{Thy} \beta \\
\text{false} & \text{otherwise}.
\end{cases}
$$

1. Then $\chi$ is well-defined and a bijection.

2. If $[\alpha]_{Thy}$ is a Boolean atom of $\mathcal{LE}_{Thy}$ and the Thy-truth function $f$ has $f(\alpha) = \text{true}$, then $f = \chi([\alpha]_{Thy})$.

**Proof.** Apply Proposition / Definition 3.3.20 and Corollary 3.3.21 with $\mathcal{A} = \mathcal{LE}_{Thy}$, and note that since $\chi([\alpha]_{Thy})$ is a Boolean homomorphism from $\mathcal{LE}_{Thy}$ to $2$, then by Lemma 3.3.11 it can be considered to be a Thy-truth function on Prop(PropAtom, Thy). $\dashv$

In this section we have considered two ways of characterising aspects of Boolean algebras. The first result is the finite case of Stone’s Representation Theorem, telling us that each finite Boolean algebra is in effect the same as a finite power set Boolean algebra. The second characterisation result tells us that each Boolean homomorphism mapping from the finite Boolean algebra $\mathcal{A}$ to $2$ is in effect the “same” as a Boolean atom of $\mathcal{A}$.

In the next chapter, we will apply both of the results of this section to the logic that we will use to formalise edits. We will be able to characterise the logic itself as a power set Boolean algebra, and we will be able to characterise the truth functions of the logic in terms of Boolean atoms.

### 3.4 Conclusion

This chapter has introduced some basic ideas of propositional logic and Boolean algebras. Like any type of logic, propositional logic consists of three components: syntax,
semantics and deduction functions. One particular deduction function for propositional logic is the resolution deduction function, which is defined in terms of certain formulae called clauses. An important consideration for any logic including propositional logic is the relationship between logical implication and a given deduction function. In those situations where the two turn out to be the same, the deduction function is called strongly sound and strongly complete, although often weaker properties such as refutation soundness and refutation completeness are useful. Another important problem is the SAT problem, of deciding whether a set of formulae is satisfiable. Deduction functions can be used to solve the SAT problem; useful deduction functions are those that are refutation sound and refutation complete, such as resolution. The set of logically equivalent formulae of a propositional logic forms a Boolean algebra called a Lindenbaum algebra, while the set \{true, false\} forms the Boolean algebra \(2\). Each finite Boolean algebra is in effect the same as the power set of some set. The Boolean homomorphisms from a finite Boolean algebra to \(2\) are characterised by special elements called Boolean atoms.

The material introduced in this chapter will be used in the logical formalisations of later chapters. Edit generation functions will be formalised via logic as deduction functions. The edits themselves will be formalised as propositional formulae. The data records will be formalised as Boolean atoms of the Lindenbaum algebra of the logic: the Boolean atoms themselves characterise the truth functions of the logic. The property of a record satisfying an edit will be formalised by logical models that satisfy logical formulae. The next chapter will systematically work through the correspondences between editing and logic.
Chapter 4

FH edit generation and logic

4.1 Introduction

The Fellegi-Holt edit generation function FH (Definition 2.4.3) takes as input a set of explicit edits, and returns as output a set of generated edits. It is similar to a logical deduction function, for three reasons. Firstly, the edits are similar to logical formulae because they give the rules according to which records are correct or incorrect. Secondly, the correct records are similar to logical models because records can satisfy or fail edits. Thirdly, the function FH is similar to a sound deduction function because every correct record satisfies the edits generated by the function.

Since the function FH is similar to a logical deduction function, it might be useful to create a logical formalisation of FH, for two reasons. It might help people to reason about the Fellegi-Holt method, and it might help develop new techniques for implementing the Fellegi-Holt method. I give a logical formalisation of the function FH in the next section, Section 4.2.

It turns out that the logical formalisation of the function FH is essentially the same as a resolution deduction function. Consequently, the vast field of results for resolution deduction translate to Fellegi-Holt edit generation, giving a new and potentially faster technique for implementing Fellegi-Holt edit generation. In Section 4.3, I give a precise statement, with proofs, of the relationship between the logical formalisation of FH and resolution.

Before proceeding to the details of the chapter, I give, in this introduction, an overview of the main concepts and insights of this chapter. I first discuss the various aspects of the logical formalisation, which then leads us to the insight into why Fellegi-Holt edit generation might be related to resolution. Summaries of this chapter can be found in my previous papers (Boskovitz, Górecki and Hegland 2003a, 2003b).

In order to formalise the function FH, we will express in terms of a logical system the main concepts used in creating the function FH. The precise definitions of the logical formalisations will be given in Section 4.2; for now we list the concepts to be formalised and the methods of formalisation:

1. edits - which we will formalise as logical formulae;
2. records - which we will formalise as truth functions;
3. normal edits - which we will formalise as certain positive clauses, to be called normal clauses;

4. the function FH itself - which we will formalise as a deduction function.

In addition, the formalisation will preserve the inter-relationships amongst the concepts. It will preserve:

1. the correctness of a record with respect to an edit - if the record \( v \) is correct with respect to the edit \( e \), then the formalisation of \( v \) will be a model of the formalisation of the edit \( e \);

2. the breakdown of an edit into a set of normal edits - if the edit \( e \) can be broken down as the union of the set \( E \) of normal edits, then the formalisation of \( e \) will be able to be broken down as the conjunction of the positive clauses forming the formalisation of the set \( E \);

3. the relationship between the edits generated by the Fellegi-Holt method and the set of explicit edits - if \( E \) is a set of explicit normal edits then the formalisation of the set \( FH(E) \) will be the same as the deduction function formalising FH applied to the clauses that formalise the edit \( E \).

The first of these inter-relationships (correctness of the record with respect to an edit) can cause some confusion, because of a difference in perspective between data editing and logic. On the one hand, each edit is a set of incorrect records, or a failure region. In contrast, each logical formula represents a set of models, or an acceptance region. Thus the logical formalisation of each edit will (in Section 4.2.3) be directly related to the complement of the edit.

The difference in perspective between data editing and logic affects the formalisation of the set operations union and intersection. For example, suppose that the edit \( e \) is the union of a set \( S \) of other edits, ie \( e = \bigcup_{s \in S} s \). Then the formalisation of \( e \) is directly related to the intersection \( \bar{e} = \bigcap_{s \in S} \bar{s} \). Similarly, the formalisation of an intersection of edits is directly related to the union of the complements of the edits rather than to the intersection.

The logical formalisation of set unions and set intersections affects the formalisation of the edit generation function FHG, because the definition of FHG (Definition 2.4.1) depends on unions and intersections. While the definition of the function FHG uses intersections over all fields except one where a union is taken, the definition of the formalisation of FHG uses unions over all fields except one where an intersection is taken.

The formalisation of FHG is reminiscent of resolution, because resolution also takes the union over all components (literals) except one pair. That is, a resolvent of two clauses is the union of all literals in those two clauses, except for one pair of literals that are cancelled out. What is more, the cancelling out of literals, of resolution, has much in common with the cancelling out that occurs in creating an intersection of a set of field values, as used in the formalisation of FHG. In this chapter, in Section 4.3,
we will formalise the similarity between Fellegi-Holt edit generation and resolution and show that they are essentially the same.

Which logical system is suitable for formalising Fellegi-Holt edit generation? Since Fellegi-Holt edit generation is about databases, one might expect to use the usual logical system for databases, namely classical first-order logic. The strength of first-order logic is that it provides a method of representing the relationships between different records in a database. However, in the Fellegi-Holt method of editing, each record is treated individually and the relationships between records are not considered. Thus first-order logic is richer than needed for formalising the Fellegi-Holt method. Thus in this work we use classical propositional logic to formalise the Fellegi-Holt method.

4.2 The formalisation of FH edit generation in terms of propositional logic

A simple minded way of formalising Fellegi-Holt edit generation in terms of propositional logic is to represent the basic elements of Fellegi-Holt edit generation, namely the sets $A^e_j$, as propositional atoms. I will first briefly present, in Section 4.2.1, such a logic, but we will very quickly notice an abundance of redundant propositional atoms. Subsequently, in Section 4.2.2, I will give a simpler logic that has fewer redundant propositional atoms.

Having defined the logic for the formalisation, our next step will be the formalisations of the various concepts used in creating the function FH. I devote one section to each concept, that is Section 4.2.3 to edits, Section 4.2.4 to records, Section 4.2.5 to normal edits, and Section 4.2.6 to the function FH itself. In each section I give the formalisation of the concept, its key properties, and conclude with a summary.

We will find that edits are naturally formalised as certain equivalence classes of formulae in the logic. However when we come to formalise FH-edit generation we will want to formalise it as a function of formulae, not of equivalence classes. So for each edit in the domain or range of FH we will choose, as its formalisation, one representative from the corresponding equivalence class of formulae: we will call the representative a normal clause, since the domain and range of FH are the normal edits. In the end we will have two formalisation functions, one to represent edits as equivalence classes of formulae and another to represent normal edits as normal clauses.

We start with the direct simple minded formalisation.

4.2.1 Direct formalisation

The most direct way to formalise Fellegi-Holt edit generation in propositional logic would be to base the logical building blocks, or propositional atoms, on the basic building blocks of the Fellegi-Holt method, which are the sets of the form $A^e_j$ that build the normal edits. That is, while the building blocks of the Fellegi-Holt method have the form $A^e_j$, the building blocks or propositional atoms of the logical formalisation
would be of the form $A^e_j$, as follows:

the set of propositional atoms = \{A^e_j \mid j = 1, \ldots, N, \text{ and } e \in \mathcal{N}(D)\}.

The propositional atom $A^e_j$ would end up representing the set of records whose $j$\textsuperscript{th} field takes a value in the $j$\textsuperscript{th} component of the normal edit $e$. This would mean that our formalisation would include a theory ThyA where, if two edits $e$ and $e'$ have the same $j$\textsuperscript{th} field, then the propositional atoms $A^e_j$ and $A^{e'}_j$ are ThyA-logically equivalent. Thus we would have a wasteful excess of duplicated propositional atoms, as in the next example.

**Example 4.2.1.** Suppose that $N = 2$, that $A_1 = A_2 = \{1,2,3,4,5,6\}$, that $e = \{1,2\} \times \{3,4\}$, and that $e' = \{5,6\} \times \{3,4\}$. Then $A^e_2 = A^{e'}_2$.

In order to reduce the duplication, we could use only the $j$\textsuperscript{th} component, $A^e_j$, of the edit $e$ in the superscript of each propositional atom, which we might call $B^e_j$ so that

the set of propositional atoms = \{B^e_j \mid j = 1, \ldots, N, \text{ and } x \subseteq A_j \}.

But even then we would have an excess of propositional atoms. In this case the propositional atom $B^e_j$ would end up representing the set of records whose $j$\textsuperscript{th} field is in the subset $x$ of $A_j$. Then our formalisation would include a theory ThyB where the propositional atom $B^e_j$ is ThyB-logically equivalent to the clause $\bigvee \{B^e_j(v) \mid v \in x\}$. Hence we do not need the propositional atom $B^e_j$ when the set $x$ has more than one element, as seen in the example below. We also do not need the propositional atom $B^e_j$ since it would have to be ThyB-logically equivalent to the empty clause.

**Example 4.2.2.** Suppose that $N = 2$, that $A_1 = A_2 = \{1,2,3,4,5,6\}$ and suppose that $x = \{3,4\}$. Then the propositional atom $B^e_j$ would end up representing the set $S$ of those records whose second field value equals 3 or 4. But $S$ is also represented by the formula $B^e_j(3) \lor B^e_j(4)$. Thus the atom $B^e_j$ seems redundant in the formalisation.

The above discussion brings us to a simpler formalisation, with

the set of propositional atoms = \{B^e_j(v) \mid j = 1, \ldots, N, \text{ and } v \in A_j \}.

We will replace the unwieldy $B^e_j(v)$ with $p^e_j$, which we will use in the simpler formalisation in the next section.

### 4.2.2 The logic for the formalisation

In this section, we will define a set PAtm of propositional atoms and a theory Th that will give us the logic Prop(PAtm, Th), which we will use to formalise Fellegi-Holt edit generation. I will also explain how the logical formulae relate to sets of records, although we will leave the definitions of the formalisations to later sections.
The formalisation of FH edit generation in terms of propositional logic

The deduction function for the logic will be the resolution deduction function $\mathcal{R}_{Th}$. We will later define another deduction function $\mathcal{F}$ which will be a logical formalisation of FH-edit generation. The two functions, $\mathcal{F}$ and $\mathcal{R}_{Th}$, will turn out to be almost the same, in a sense to be defined later in the chapter, in Section 4.3.

The propositional atoms represent the individual field values, as done by Bruni and Sassano (2001a), and also by Franconi et al. (2001), who use ground atomic formulae of first-order logic rather than propositional atoms. The set $\text{PAtm}$ of propositional atoms is defined as follows:

**Definition 4.2.3.** The set $\text{PAtm}$ of propositional atoms is

$$\text{PAtm} = \{p^v_j \mid j = 1, \ldots, N \text{ and } v \in A_j\}.$$ 

The propositional atom $p^v_j$ will end up representing the set of records whose $j^{\text{th}}$ field takes the value $v$. In fact we will use the word “field” formally:

**Definition 4.2.4.** The subscript $j$ in the propositional atom $p^v_j$ will be called the **field** of $p^v_j$.

In the formalisation, each formula will end up representing a set of records in the domain $D$. The symbol $\neg$ will end up representing a set complement; the symbol $\land$ will end up representing set intersection; and the symbol $\lor$ will end up representing set union - as in the next example.

**Example 4.2.5.** Suppose that the domain has two fields, that is $N = 2$, and that $A_1 = A_2 = \{1, 2, 3, 4, 5\}$.

1. The formula $p^5_1$ will represent the set $\{5\} \times A_2$ of all records whose first field takes the value 5.
2. The formula $\neg p^5_1$ will represent the set $D \setminus (\{5\} \times A_2)$.
3. The formula $p^5_1 \land p^3_2$ will represent the set $\{5, 3\}$ containing just one record.
4. The formula $p^5_1 \lor p^3_2 \lor p^4_1$ will represent the set $(\{5\} \times A_2) \cup (A_1 \times \{3, 4\})$ of all records where the first field takes value 5 or the second field takes the value 3 or 4.
5. The formula $p^1_1 \lor p^2_1 \lor p^3_1 \lor p^4_1 \lor p^5_1$ will represent the whole domain $D$.
6. The empty clause will represent the empty set.

Each record set will end up having more than one formula to represent it, because each set be written in different ways, as in the next example.

**Example 4.2.6.** As in the previous example, suppose that the domain has two fields, that is $N = 2$, and that $A_1 = A_2 = \{1, 2, 3, 4, 5\}$.

1. Since the set $D \setminus (\{5\} \times A_2)$ can also be written $\{1, 2, 3, 4\} \times A_2$, it will be represented by both $\neg p^5_1$ and $p^1_1 \lor p^2_1 \lor p^3_1 \lor p^4_1 \lor p^5_1$. 


2. Since the set \( \{5\} \times A_2 \) can also be written as the union of its single element subsets, namely as \( \bigcup \{\{(5, v)\} \mid v \in A_2\} \), it will be represented by both \( p_5^v \) and \( \bigvee \{p_1^v \land p_2^v \mid v \in A_2\} \).

Where multiple formulae end up representing the same set, those formulae will have to be logically equivalent under the theory of the semantics, which is given below in the next definition. The theory, to be called \( \text{Th} \), will reflect the underlying reason why each set of records can be written in multiple ways, which is that any field of a record must take exactly one value in the field domain. We will split the theory \( \text{Th} \) into two subsets, \( \text{Th}_1 \) and \( \text{Th}_2 \). The formulae in the set \( \text{Th}_1 \) ensure that each field of each record takes at most one value; while the formulae in the set \( \text{Th}_2 \) ensure that each field of each record takes at least one value. That is:

**Definition 4.2.7.** The theory for the semantics is the set \( \text{Th} \) where

\[
\text{Th} = \text{Th}_1 \cup \text{Th}_2,
\]

and

\[
\text{Th}_1 = \{ \neg p_j^v \lor \neg p_j^w \mid j = 1, \ldots, N, \text{ and } v, w \in A_j, \text{ and } v \neq w \},
\]

and

\[
\text{Th}_2 = \{ \bigvee_{v \in A_j} p_j^v \mid j = 1, \ldots, N \}.
\]

Note that \( \text{Th} \) is satisfiable, for example by the \( \text{Th} \)-truth function \( f \) defined in terms of the record \((w_1, \ldots, w_N)\) for \( j = 1, \ldots, N \) and \( v \in A_j \) by \( f(p_j^v) = \text{true} \) if and only if \( v = w_j \).

Note that the theory \( \text{Th} \) was called a set of “axioms” in the previously mentioned papers (Boskovitz et al. 2003a; Boskovitz et al. 2003b).

We can now use the theory \( \text{Th} \) to prove that the various formulae given in Example 4.2.6 to represent the same set are indeed \( \text{Th} \)-logically equivalent. The next lemma gives a generalisation of the property of part 1 of Example 4.2.6.

**Lemma 4.2.8.** Let \( p_j^v \in \text{PAtm} \). Then

\[
\neg p_j^v \equiv_{\text{Th}} \bigvee_{w \in A_j \setminus \{v\}} p_j^w.
\]

**Proof.** Write \( \alpha \) for the right side of the equation to be proved, that is \( \alpha = \bigvee_{w \in A_j \setminus \{v\}} p_j^w \).

We first show that \( \neg p_j^v \models_{\text{Th}} \alpha \). Suppose \( f \) is a \( \text{Th} \)-truth function and \( f(\neg p_j^v) = \text{true} \). Since \( \text{Th}_2 \) contains \( \bigvee_{x \in A_j} p_j^x \), which equals \( \alpha \lor p_j^v \), we have that \( f(\alpha \lor p_j^v) = \text{true} \). Since \( f(p_j^v) = \text{false} \), we have that \( f(\alpha) = \text{true} \). Hence \( \neg p_j^v \models_{\text{Th}} \alpha \).

We now show that \( \alpha \models_{\text{Th}} \neg p_j^v \). Suppose that \( f \) is a \( \text{Th} \)-truth function and \( f(\alpha) = \text{true} \). Then there is an \( x \) in \( A_j \setminus \{v\} \) such that \( f(p_j^x) = \text{true} \). But \( \neg p_j^y \lor \neg p_j^z \in \text{Th}_1 \) since \( x \neq v \), and hence \( f(\neg p_j^y \lor \neg p_j^z) = \text{true} \). Since \( f(\neg p_j^y) = \text{false} \), we have that \( f(\neg p_j^z) = \text{true} \), and hence \( \alpha \models_{\text{Th}} \neg p_j^y \). \( \dashv \)
Before proceeding to also generalise part 2 of Example 4.2.6, we note that a consequence of the above lemma is that we can avoid the symbol \( \neg \) in all clauses and formulae, as in the next corollary.

**Corollary 4.2.9.** To each formula there is a Th- logically equivalent conjunction of positive clauses.

*Proof.* Let \( \alpha \) be a formula. By Lemma 3.2.13, \( \alpha \) is logically equivalent to, and hence Th-logically equivalent to, a conjunction of clauses. By Lemma 4.2.8, each clause in this conjunction is Th-logically equivalent to a positive clause. \( \dashv \)

A generalisation of part 2 of Example 4.2.6 is that each formula is the disjunction of representations of records, as given in the next corollary.

**Corollary 4.2.10.** If \( \alpha \) is a formula then there is a subset \( X \) of \( D \) such that

\[
\alpha \equiv_{\text{Th}} \bigvee_{v \in X} \bigwedge_{j=1}^{N} p^v_j,
\]

where \( v \) is written \((v_1, \ldots, v_N)\).

*Proof.* By Corollary 4.2.9, \( \neg \alpha \) is Th-logically equivalent to a conjunction of positive clauses. Write:

\[
\neg \alpha \equiv_{\text{Th}} \bigwedge_{(S_1, \ldots, S_N) \in U} \bigvee_{j=1}^{N} \bigvee_{v \in S_j} p^v_j,
\]

where \( U \subseteq \prod_{i=1}^{N} P(A_i) \).

We use Th2 to omit some elements of \( U \), as follows. If, for some \((S_1, \ldots, S_N)\) in \( U \) and some \( k \) in \( \{1, \ldots, N\} \), we have that \( S_k = A_k \), then, using Th2, we have that \( \bigvee_{j=1}^{N} \bigvee_{v \in S_j} p^v_j \equiv_{\text{Th}} \top \), so that \((S_1, \ldots, S_N)\) could be omitted from \( U \) while still retaining a Th-logically equivalent expression for \( \alpha \). Hence we define the subset \( V \) of \( U \) by:

\[
V = \{(S_1, \ldots, S_N) \in U \mid \text{for each } k \text{ in } \{1, \ldots, N\}, S_k \neq A_k\},
\]

and then

\[
\neg \alpha \equiv_{\text{Th}} \bigwedge_{(S_1, \ldots, S_N) \in V} \bigvee_{j=1}^{N} \bigvee_{v \in S_j} p^v_j.
\]

We now find a formula in the required form that is Th-logically equivalent to \( \alpha \). We first apply De Morgan’s Law to \( \neg \neg \alpha \), using the above formula for \( \neg \alpha \):

\[
\alpha \equiv_{\text{Th}} \neg \neg \alpha \equiv_{\text{Th}} \bigvee_{(S_1, \ldots, S_N) \in V} \bigwedge_{j=1}^{N} \bigwedge_{v \in S_j} \neg p^v_j, \tag{4.2.1}
\]
We now note that the innermost conjunction can be rewritten:

\[
\bigwedge_{v \in S_j} \neg p^v_j \equiv_{\text{Th}} \bigwedge_{v \in S_j} \bigvee_{w \in A_j \setminus \{v\}} p^w_j,
\]

by Lemma 4.2.8

\[
\equiv_{\text{Th}} \bigvee_{w \in \prod_{v \in S_j} (A_j \setminus \{v\})} \bigwedge_{i=1}^n p^w_j,
\]

by the distributive law, where \( w = (w_1, \ldots, w_n) \) and \( n = |S_j| \)

\[
\equiv_{\text{Th}} \bigvee_{z \in \bigcap_{v \in S_j} (A_j \setminus \{v\})} p^z_j,
\]

because if \( x \neq y \) then \( p^x_j \land p^y_j \equiv_{\text{Th}} \square \) using Th1

\[
\equiv_{\text{Th}} \bigvee_{z \in A_j \setminus S_j} p^z_j,
\]

since \( \bigcap_{v \in S_j} (A_j \setminus \{v\}) = A_j \setminus S_j \).

We apply this rewriting to the previous expression for \( \alpha \) (Equation 4.2.1):

\[
\alpha \equiv_{\text{Th}} \bigvee_{(S_1, \ldots, S_N) \in V} \bigwedge_{j=1}^N \bigvee_{z \in A_j \setminus S_j} p^z_j
\]

\[
\equiv_{\text{Th}} \bigvee_{(S_1, \ldots, S_N) \in V} \bigvee_{(z_1, \ldots, z_N) \in Z} \bigwedge_{j=1}^N p^z_j, \text{ where } Z = (A_1 \setminus S_1) \times \cdots \times (A_N \setminus S_N),
\]

using the distributive law

\[
\equiv_{\text{Th}} \bigvee_{(z_1, \ldots, z_N) \in X} \bigwedge_{j=1}^N p^z_j, \text{ where } X = \bigcup \{Z \mid (S_1, \ldots, S_N) \in V\}.
\]

The last formula is of the required form.

Thus not only does each formula represent some set of records, but it can be expressively written as a disjunction of formulae representing records.

We have defined the logic \( \text{Prop}(\text{PAtm}, \text{Th}) \) in which we can relate formulae to sets of records. We have also demonstrated that certain formulae representing the same record set are Th-logically equivalent. In the next sections we will formalise the connections between formulae and sets of records. It turns out that there is a Boolean isomorphism between the Boolean algebra of Th-equivalence classes of formulae (the Lindenbaum algebra \( \mathcal{L}_{\text{Th}} \)) and the Boolean algebra of record sets. Such a connection can be formalised by using the Boolean isomorphism between the Boolean algebra of Th-equivalence classes of formulae (the Lindenbaum algebra \( \mathcal{L}_{\text{Th}} \)) and the Boolean algebra of record sets. The next section concentrates on the logical formalisation of edits.
4.2.3 Formalisation of edits

We will now show how edits can be formalised in the logic \( \text{Prop}(\text{PAtm}, \text{Th}) \). Since edits are sets of records, and each set of records can in general be represented by more than one formula, we will work in terms of the Boolean algebra of Th-equivalence classes of formulae, namely the Lindenbaum algebra \( \mathcal{LE}_{\text{Th}} \). We will first note that the edits on \( D \) form a Boolean algebra \( \mathcal{E} \), to be defined below. It turns out that the two Boolean algebras, \( \mathcal{LE}_{\text{Th}} \) and \( \mathcal{E} \), are isomorphic as Boolean algebras, and so \( \mathcal{LE}_{\text{Th}} \) can be used to formalise \( \mathcal{E} \).

We first consider how logical formulae can be related to edits rather than record sets. Although each formula \( \alpha \) will represent some set \( S \) of records, the edit satisfied by the records in \( S \) is the complement of the set \( S \). Thus the formula \( \alpha \) will represent the edit \( D \setminus S \).

**Example 4.2.11.** As in the previous examples, suppose that the domain has two fields, that is \( N = 2 \), and that \( A_1 = A_2 = \{1, 2, 3, 4, 5\} \).

1. Although the formula \( \alpha = \neg p^1_5 \) will represent the record set \( S = D \setminus (\{5\} \times A_2) \), the edit represented by \( \alpha \) is the complement of \( S \), namely \( D \setminus S = \{5\} \times A_2 \). That is, the edit specifies that field 1 must not equal 5.

2. Although the Th-valid formulae, such as \( p^1_1 \lor p^2_1 \lor p^3_1 \lor p^4_1 \lor p^5_1 \), will represent the domain \( D \) of records, the edit represented by the Th-valid formulae is the complement of \( D \), namely the empty set, which is the uninformative edit satisfied by all records.

As before for record sets, each formula will end up representing an edit in \( D \). However because of the complementary relationship between edits and record sets, the symbol \( \land \) will end up representing the union of edits rather than the intersection of record sets, and the symbol \( \lor \) will end up representing the intersection of edits rather than the union of record sets. Thus the Boolean algebra of edits that we are interested in is the dual power set algebra \( \mathcal{P}'_D \) which we will write simply as \( \mathcal{E} \), as in the next definition.

**Definition 4.2.12.** The Boolean algebra of all possible edits on \( D \) is the dual power set algebra \( \mathcal{P}'_D \). We will write \( \mathcal{E} = \mathcal{P}'_D \), and

\[
\mathcal{E} = \langle \mathcal{P}(D), \cap, \cup, \sim, D, \emptyset \rangle,
\]

where for any set \( X \), we define \( \sim X = D \setminus X = X' \).

Thus, in the notation of Chapter 3, \(+_\mathcal{E} \) is \( \cap \), \( \cdot_\mathcal{E} \) is \( \cup \), \( \sim_\mathcal{E} \) is \( \sim \), \( 0_\mathcal{E} \) is \( D \), and \( 1_\mathcal{E} \) is \( \emptyset \).

We are now ready to show that the Boolean algebra \( \mathcal{E} \) of edits is isomorphic as a Boolean algebra to the Lindenbaum algebra \( \mathcal{LE}_{\text{Th}} \). The two Boolean algebras are isomorphic because they have the same number of Boolean atoms. We saw in Chapter 3

---

\( \S 4.2 \) The formalisation of FH edit generation in terms of propositional logic 93
(in Example 3.3.13) that $\mathcal{E} = \mathcal{Y}_D$ has $|D|$ Boolean atoms as follows:

$$\text{BoolAtm}(\mathcal{E}) = \{ D \setminus \{ v \} \mid v \in D \}.$$ 

The next proposition gives us that the Boolean algebra $\mathcal{L}_{\mathcal{E}_{\text{Th}}}$ has the same number $|D|$ of Boolean atoms as $\mathcal{E}$:

**Proposition 4.2.13.**

1. The set of Boolean atoms of $\mathcal{L}_{\mathcal{E}_{\text{Th}}}$ is

$$\text{BoolAtm}(\mathcal{L}_{\mathcal{E}_{\text{Th}}}) = \left\{ [p_1^{v_1} \land \cdots \land p_N^{v_N}]_{\text{Th}} \mid (v_1, \ldots, v_N) \in D \right\},$$

where $[p_1^{v_1} \land \cdots \land p_N^{v_N}]_{\text{Th}}$ is the equivalence class of the formula $p_1^{v_1} \land \cdots \land p_N^{v_N}$ under $\text{Th}$-logical equivalence, as defined in Definition 3.2.9.

2. Each listed element of the above set is distinct, that is, if $v = (v_1, \ldots, v_N) \in D$ and $w = (w_1, \ldots, w_N) \in D$ and $[p_1^{v_1} \land \cdots \land p_N^{v_N}]_{\text{Th}} = [p_1^{w_1} \land \cdots \land p_N^{w_N}]_{\text{Th}}$, then $v = w$.

**Proof.**

1. Let $X$ be equal to the right-hand side of the equation to be proved, that is

$$X = \left\{ [p_1^{v_1} \land \cdots \land p_N^{v_N}]_{\text{Th}} \mid (v_1, \ldots, v_N) \in D \right\},$$

We first show that $X \subseteq \text{BoolAtm}(\mathcal{L}_{\mathcal{E}_{\text{Th}}})$. Suppose that $\xi \in X$ and that $\xi = [p_1^{x_1} \land \cdots \land p_N^{x_N}]_{\text{Th}}$, where $(x_1, \ldots, x_N) = x \in D$. We show that $\xi$ is a Boolean atom of $\mathcal{L}_{\mathcal{E}_{\text{Th}}}$. Suppose that $\alpha \in \mathcal{L}_{\mathcal{E}_{\text{Th}}}$ and that $\alpha \leq \mathcal{L}_{\mathcal{E}_{\text{Th}}} \xi$. We will show that $\alpha = \Box$ or $\alpha = \xi$. By Corollary 4.2.10 there is a subset $W$ of $D$ such that

$$\alpha = \left[ \bigvee_{w \in W} \left[ \bigwedge_{j=1}^{N} p_j^{w_j} \right]_{\text{Th}} \right], \text{ where } w = (w_1, \ldots, w_N).$$

Since $\alpha \leq \mathcal{L}_{\mathcal{E}_{\text{Th}}} \xi$ we have that $\alpha = \alpha \land \xi$, by Lemma 3.3.6 part 2. Hence, using the distributive law, we have that

$$\alpha = \left[ \bigvee_{w \in W} \left( \bigwedge_{j=1}^{N} p_j^{w_j} \land \bigwedge_{j=1}^{N} p_j^{x_j} \right) \right]_{\text{Th}}.$$ (4.2.2)

But if $w \neq x$ then the inner bracket $\bigwedge_{j=1}^{N} p_j^{w_j} \land \bigwedge_{j=1}^{N} p_j^{x_j} = \Box$, using Th1. Hence Equation 4.2.2 becomes

$$\alpha = \begin{cases} \Box & \text{if } x \notin W \\ \xi & \text{if } x \in W. \end{cases}$$
Hence $\xi$ is a Boolean atom, and thus $X \subseteq \text{BoolAtm}(\mathcal{E}_{\text{Th}})$.

We now show the reverse subset relation, that is that $\text{BoolAtm}(\mathcal{E}_{\text{Th}}) \subseteq X$. Suppose that $\zeta \in \text{BoolAtm}(\mathcal{E}_{\text{Th}})$. Then by Corollary 4.2.10, there is a subset $Y$ of $D$ such that

$$\zeta = \left[ \bigvee_{\mathfrak{y} \in Y} \bigwedge_{j=1}^{N} p_{j}^{\mathfrak{y}} \right]_{\text{Th}}$$

where $\mathfrak{w} = (w_1, \ldots, w_N)$.

Since $\zeta$ is a Boolean atom, we have that $\zeta \neq \Box$, and hence there is a $\mathfrak{y} = (y_1, \ldots, y_N)$ in $Y$ such that $[p_1^{y_1} \land \cdots \land p_N^{y_N}]_{\text{Th}} \neq \Box$. Also, by part 2a of Lemma 3.3.7,

$$[p_1^{y_1} \land \cdots \land p_N^{y_N}]_{\text{Th}} \leq \mathcal{E}_{\text{Th}} \zeta.$$

But since $\zeta$ is a Boolean atom and $[p_1^{y_1} \land \cdots \land p_N^{y_N}]_{\text{Th}} \neq \Box$, we have that $[p_1^{y_1} \land \cdots \land p_N^{y_N}]_{\text{Th}} = \zeta$, giving $\zeta$ in the correct form to be in $X$, as required.

2. Let $\mathfrak{v} = (v_1, \ldots, v_N)$ and let $\mathfrak{w} = (w_1, \ldots, w_N)$ be in $D$, and suppose that $[p_1^{v_1} \land \cdots \land p_N^{v_N}]_{\text{Th}} = [p_1^{w_1} \land \cdots \land p_N^{w_N}]_{\text{Th}}$. We will show that $\mathfrak{v} = \mathfrak{w}$.

First note that $[p_1^{v_1} \land \cdots \land p_N^{v_N}]_{\text{Th}} \land [p_1^{w_1} \land \cdots \land p_N^{w_N}]_{\text{Th}} = [p_1^{v_1} \land \cdots \land p_N^{v_N}]_{\text{Th}}$, by Lemma 3.3.6 part 1. Then $[p_1^{v_1} \land \cdots \land p_N^{v_N}]_{\text{Th}} \land [p_1^{w_1} \land \cdots \land p_N^{w_N}]_{\text{Th}} \neq \Box$, since $[p_1^{v_1} \land \cdots \land p_N^{v_N}]_{\text{Th}}$ is a Boolean atom. But $[p_1^{v_1} \land \cdots \land p_N^{v_N}]_{\text{Th}} \land [p_1^{w_1} \land \cdots \land p_N^{w_N}]_{\text{Th}} = ([p_1^{v_1} \land p_1^{w_1}] \land \cdots \land (p_N^{v_N} \land p_N^{w_N}))_{\text{Th}}$, since the $\text{Th}$-equivalence relation is a congruence relation. Hence for all $j$ in $\{1, \ldots, N\}$, we have that $v_j = w_j$, and hence $\mathfrak{v} = \mathfrak{w}$.

Since the two Boolean algebras $\mathcal{E}$ and $\mathcal{E}_{\text{Th}}$ have the same number of Boolean atoms, we can now use the results of Chapter 3 to define a Boolean isomorphism between $\mathcal{E}$ and $\mathcal{E}_{\text{Th}}$. The isomorphism is an extension of the natural bijection $ba$ (standing for “Boolean atoms”) between $\text{BoolAtm}(\mathcal{E})$ and $\text{BoolAtm}(\mathcal{E}_{\text{Th}})$, where $ba$ is defined as follows:

**Definition 4.2.14.** The function

$$ba : \text{BoolAtm}(\mathcal{E}) \rightarrow \text{BoolAtm}(\mathcal{E}_{\text{Th}}),$$

is defined for each $\mathfrak{v} = (v_1, \ldots, v_N)$ in $D$ by

$$ba(D \setminus \{\mathfrak{v}\}) = [p_1^{v_1} \land \cdots \land p_N^{v_N}]_{\text{Th}}.$$

We now show that $ba$ is indeed a bijection, and that it can be uniquely extended to a Boolean isomorphism:
Lemma and Definition 4.2.15.

1. The function \( ba \) is a bijection.

2. The function \( ba \) extends uniquely to the Boolean isomorphism

\[
ec : \mathfrak{E} \rightarrow \mathfrak{L}E_{Th},
\]

where if \( e \) is in \( \mathfrak{E} \), then \( ec \) (standing for “equivalence class”) is defined by

\[
ec(e) = \left[ \bigvee_{v \in D \setminus e} (p_{v_1}^{w_1} \land \cdots \land p_{v_N}^{w_N}) \right]_{Th}.
\]

Proof.

1. The function \( ba \) is injective because, if the records \( v = (v_1, \ldots, v_N) \) and \( w = (w_1, \ldots, w_N) \) are in \( D \) and \([p_{v_1}^{w_1} \land \cdots \land p_{v_N}^{w_N}]_{Th} = [p_{v_1}^{w_1} \land \cdots \land p_{v_N}^{w_N}]_{Th}\) then, by Proposition 4.2.13, \( v = w \). The function \( ba \) is surjective because each element \([p_{v_1}^{w_1} \land \cdots \land p_{v_N}^{w_N}]_{Th}\) of \( \text{BoolAtm}(\mathfrak{L}E_{Th}) \) corresponds to the element \( v \) of \( D \).

2. By Proposition 3.3.18 of Chapter 3, the bijection \( ba \) can be uniquely extended to the Boolean isomorphism \( ba^* : \mathfrak{E} \rightarrow \mathfrak{L}E_{Th} \) where if \( e \in \mathcal{P}(D) \) then we have that

\[
ba^*(e) = \bigvee_{v \in D \setminus e} ba^*(D \setminus \{v\}), \quad \text{since } ba^* \text{ is a Boolean homomorphism and}
\]

\[
e = \bigcap_{v \in D \setminus e} (D \setminus \{v\})
\]

\[
= \bigvee_{v \in D \setminus e} ba(D \setminus \{v\}), \quad \text{since } ba^* \text{ extends } ba \text{ and } D \setminus \{v\} \in \text{BoolAtm}(\mathfrak{E})
\]

\[
= \left[ \bigvee_{v \in D \setminus e} (p_{v_1}^{w_1} \land \cdots \land p_{v_N}^{w_N}) \right]_{Th}, \quad \text{by the definition of } ba, \text{ and the fact that Th-equivalence is a congruence relation.}
\]

Let \( ec = ba^* \).

Summary: Formalisation of edits

The formalisation in \( \mathfrak{L}E_{Th} \) of the edit \( e \) is \( ec(e) \), where

\[
ec(e) = \left[ \bigvee_{v \in D \setminus e} (p_{v_1}^{w_1} \land \cdots \land p_{v_N}^{w_N}) \right]_{Th},
\]

and the function \( ec : \mathfrak{E} \rightarrow \mathfrak{L}E_{Th} \) is a Boolean isomorphism.
4.2.4 Formalisation of records

Having formalised edits as equivalence classes of formulae, we might wish to also formalise records in the same way, also as equivalence classes of formulae. Indeed, we earlier foreshadowed a method of formalisation for the record $v = (v_1, \ldots, v_N)$: since the propositional atom $p_j^{v_j}$ was to end up representing the set of records whose $j$-th field is $v_j$, then the formula $p_1^{v_1} \land \cdots \land p_N^{v_N}$ was to end up representing the record $v$. We could formalise the record $v$ as the equivalence class $[p_1^{v_1} \land \cdots \land p_N^{v_N}]_{Th}$.

In order to show that such a formalisation preserves the correctness of records, we would have to show that the record $v$ satisfies the edit $e$ if and only if the equivalence class $[p_1^{v_1} \land \cdots \land p_N^{v_N}]_{Th}$ of formulae Th-logically implies the equivalence class $\ec(e)$ of formulae. That is we would have to show that $v \notin e$ if and only if $[p_1^{v_1} \land \cdots \land p_N^{v_N}]_{Th} \models_{Th} \ec(e)$. In particular we would have to show that if $v \notin e$ and the Th-truth function $f$ has $f([p_1^{v_1} \land \cdots \land p_N^{v_N}]_{Th}) = \true$, then $f(\ec(e)) = \true$.

But $[p_1^{v_1} \land \cdots \land p_N^{v_N}]_{Th}$ is a Boolean atom of $\LTh$, and hence, by Corollary 3.3.22, the above Th-truth function $f$ is unique and equals $\chi([p_1^{v_1} \land \cdots \land p_N^{v_N}]_{Th})$ – which we will write more readably as $f_v$ – where $\chi : \BoolAtm(\LTh) \to \BoolHom(\LTh, 2)$ is the characteristic homomorphism introduced in Proposition / Definition 3.3.20. Thus we could equally have formalised the record $v$ as the Th-truth function $f_v$ instead of as the equivalence class $[p_1^{v_1} \land \cdots \land p_N^{v_N}]_{Th}$.

What is more, in the editing world we tend to think about a record $v$ in terms of its relationships with edits rather than as a subset of the domain $D$. Thus a formalisation in terms of a Th-truth function which focuses on its relationships with formulae is more natural than a formalisation as an equivalence class of formulae which focuses on the formulae in isolation.

We will formalise the record $v$ as the Th-truth function $f_v$. We proved in Chapter 3 that $f_v$ has exactly the properties we want, although we will restate those properties in Lemma 4.2.17, after giving a formal definition of $f_v$ below.

**Definition 4.2.16.** If $v = (v_1, \ldots, v_N) \in D$ we define the Th-truth function $f_v$ on $\Prop(\PAtm, Th)$ by

$$f_v = \chi([p_1^{v_1} \land \cdots \land p_N^{v_N}]_{Th}),$$

where $\chi : \BoolAtm(\LTh) \to \BoolHom(\LTh, 2)$ is as defined in Proposition / Definition 3.3.20, that is, if $\alpha$ is a formula, then $\chi([p_1^{v_1} \land \cdots \land p_N^{v_N}]_{Th})(\alpha) = \true$ if and only if $p_1^{v_1} \land \cdots \land p_N^{v_N} \models_{Th} \alpha$.

As we have done for $\chi([p_1^{v_1} \land \cdots \land p_N^{v_N}]_{Th})$, we will apply $f_v$ (ambiguously but comprehensibly) to both formulae and Th-equivalence classes of formulae.

The next lemma will show that the function $f_v$ formalises the record $v$ in two senses:

1. it gives a bijection between records and Th-truth functions; and
2. it preserves the “correctness” of records:

**Lemma 4.2.17.** Let $v$ be a record and let $e$ be an edit. Then

1. the mapping $v \mapsto f_v$ is a bijection between records and Th-truth functions;
2. \( v \) satisfies \( e \) if and only if \( f_v(\text{ec}(e)) = \text{true} \).

Proof.

1. Note that \( f_v \) is indeed a Th-truth function, by its definition. The required bijection follows from two intermediate bijections. Firstly, for \( v = (v_1, \ldots, v_N) \) the mapping \( v \mapsto [p_1^{v_1} \land \cdots \land p_N^{v_N}]_{\text{Th}} \) is a bijection from \( D \) to the set of Boolean atoms, by Proposition 4.2.13. Secondly, the mapping \( [p_1^{v_1} \land \cdots \land p_N^{v_N}]_{\text{Th}} \mapsto \chi([p_1^{v_1} \land \cdots \land p_N^{v_N}]_{\text{Th}}) \) is a bijection from the set of Boolean atoms to the set of Th-truth functions, by Corollary 3.3.22.

2. The result follows from the following sequence of equivalent statements, for \( v = (v_1, \ldots, v_N) \):

\[
\begin{align*}
v \notin e & \iff D \setminus \{v\} \supseteq e \\
& \iff \text{ec}(D \setminus \{v\}) \leq_{\text{Th}} \text{ec}(e), \text{ since ec is a Boolean isomorphism} \\
& \iff [p_1^{v_1} \land \cdots \land p_N^{v_N}]_{\text{Th}} \leq_{\text{Th}} \text{ec}(e), \text{ by the definition of ec (Lemma / Definition 4.2.15)} \\
& \iff (\chi([p_1^{v_1} \land \cdots \land p_N^{v_N}]_{\text{Th}}))(\text{ec}(e)) = \text{true}, \text{ by the definition of } \chi \text{ (Proposition / Definition 3.3.20), since } [p_1^{v_1} \land \cdots \land p_N^{v_N}]_{\text{Th}} \text{ is a Boolean atom} \\
& \iff f_v(\text{ec}(e)) = \text{true}, \text{ by the definition of } f_v \text{ (Definition 4.2.16).}
\end{align*}
\]

Rather than defining the Th-truth function \( f_v \) in terms of \( \chi \), the next lemma gives \( f_v \) in two more common ways, in terms of formulae and propositional atoms.

**Lemma 4.2.18.** Let \( \alpha \) be a formula; let \( v = (v_1, \ldots, v_N) \) be in \( D \); let \( j \) be a field; and let \( w \) be in \( A_j \). Then

1. \( f_v(\alpha) = \text{true} \) if and only if \( p_1^{v_1} \land \cdots \land p_N^{v_N} \models_{\text{Th}} \alpha \);

2. \( f_v(p_j^w) = \text{true} \) if and only if \( w = v_j \).

Proof.

1. The result follows from Corollary 3.3.22 using the definition that

\[
f_v = \chi([p_1^{v_1} \land \cdots \land p_N^{v_N}]_{\text{Th}}).
\]

2. The result follows from the following sequence of equivalences:

\[
\begin{align*}
f_v(p_j^w) & = \text{true} \\
& \iff p_1^{v_1} \land \cdots \land p_N^{v_N} \models_{\text{Th}} p_j^w, \text{ from the first part of this lemma} \\
& \iff w = v_j, \text{ using Th1}.
\end{align*}
\]
Summary: Formalisation of records

The formalisation in \( \text{Prop}(\text{PAtm}, \text{Th}) \) of the record \( v = (v_1, \ldots, v_N) \) is the \( \text{Th} \)-truth function \( f_v \), where

\[
f_v = \chi([p_{v1} \land \cdots \land p_{vN}]_{\text{Th}}),
\]

where \( \chi \) is as defined in Chapter 3.

The function \( f_v \) can also be defined by the following statement:

\[
\text{if } \alpha \text{ is a formula, then } f_v(\alpha) = \text{true} \text{ if and only if } p_{v1} \land \cdots \land p_{vN} \models_{\text{Th}} \alpha,
\]

and also by:

\[
\text{if } j \text{ is a field and } w \in A_j, \text{ then } f_v(p_j^w) = \text{true} \text{ if and only if } w = v_j.
\]

The formalisation of records is consistent with the concept of the correctness of records in the sense that:

\[
\text{if } e \text{ is an edit, then the record } v \text{ satisfies } e \text{ if and only if } f_v(\text{ec}(e)) = \text{true}.
\]

Thus the function \( f_v \) is a model of the formalisations of the edits satisfied by \( v \).

4.2.5 A more useful formalisation of normal edits

We have previously defined a formalisation of edits. However, since Fellegi-Holt edit generation is in terms of normal edits, we are more interested in the logical formalisation of normal edits than in the formalisation of edits in general.

We have seen that under the above formalisation, each normal edit \( e \) is formalised as the equivalence class \( \text{ec}(e) \). However there is a difficulty with using an equivalence class as the logical formalisation of a normal edit, because we want to formalise FH-edit generation as a deduction function on individual formulae, not on equivalence classes. Hence, as a more useful formalisation of the normal edit \( e \), we will choose one formula, to be called a “normal clause” \( \text{nc}(e) \), from the equivalence class \( \text{ec}(e) \).

The question is: which formula in the \( \text{Th} \)-equivalence class \( \text{ec}(e) \) should be chosen as the formula \( \text{nc}(e) \) to formalise the normal edit \( e \)? The answer is straightforward for non-empty normal edits and more complex for the empty edit. If \( e \) is a non-empty normal edit, then we will show below that the equivalence class \( \text{ec}(e) \) contains exactly one positive clause, which can be chosen as \( \text{nc}(e) \). As to the empty edit, the equivalence class \( \text{ec}(\emptyset) \) contains many positive clauses: we will choose from \( \text{ec}(\emptyset) \) one particular positive clause, called \( \top\top \), to be defined later in this section, as the formalisation \( \text{nc}(\emptyset) \).

We will then see that the function \( \text{nc} \) of normal edits is indeed a formalisation, for four reasons. Firstly, it is a bijection from the set of normal edits to the set of normal clauses, to be defined precisely later in this section. Secondly, it preserves the correctness of records, just as the function \( \text{ec} \) does. Thirdly, the breakdown of an arbitrary edit into normal edits is consistent with the breakdown of the corresponding
equivalence class $ec(e)$ into equivalence classes of corresponding normal clauses. And finally, it preserves the ordering relation $\supseteq$ on normal edits. We will prove these results later in this section.

Note that we have said that $nc$ is a bijection, not that it is a Boolean isomorphism. It cannot be a Boolean isomorphism because its domain (normal edits) does not form a Boolean algebra, since unions of sets of normal edits need not be normal. However, as listed in the properties above, the function $nc$ does preserve some structures in a similar way to a Boolean isomorphism.

We will first show that for any normal edit $e$, the equivalence class $ec(e)$ contains at least one positive clause, by directly calculating $ec(e)$ in the next proposition.

**Proposition 4.2.19.** If $e$ is a normal edit, with $e = A_1^e \times \cdots \times A_N^e$, then

$$ec(e) = \bigg[ \bigvee_{j=1}^N \bigg( \bigvee_{v \in \overline{A}_j^e} p_j^v \bigg) \bigg]_{Th},$$

where $\overline{A}_j^e = A_j^e \setminus A_j^e$.

**Proof.** We first show that the normal edit $e$ can be written

$$e = \bigcap_{j=1}^N \bigcap_{v \in \overline{A}_j^e} \bigcap_{w \in D} (D \setminus \{w\}), \text{ where } w = (w_1, \ldots, w_N) \quad (4.2.3)$$

by the following sequence of equivalent statements about sets:

$$x = (x_1, \ldots, x_N) \in \text{ right side of Equation 4.2.3}$$

$$\Leftrightarrow \forall j \in \{1, \ldots, N\} \quad \forall v \in \overline{A}_j^e \quad \forall w \in D \quad (w_j = v \Rightarrow x \in D \setminus \{w\})$$

$$\Leftrightarrow \forall j \in \{1, \ldots, N\} \quad \forall v \in \overline{A}_j^e \quad (x_j \neq v)$$

$$\Leftrightarrow \forall j \in \{1, \ldots, N\} \quad (x_j \in A_j^e)$$

$$\Leftrightarrow x \in A_1^e \times \cdots \times A_N^e = e.$$ 

We now calculate $ec(e)$. We apply $ec$ to Equation 4.2.3, and use the fact that $ec$ is a Boolean homomorphism.

$$ec(e) = \bigg[ \bigvee_{j=1}^N \bigg( \bigvee_{v \in \overline{A}_j^e} ec(D \setminus \{w\}) \bigg) \bigg]_{Th}$$

$$= \bigg[ \bigvee_{j=1}^N \bigg( \bigvee_{v \in \overline{A}_j^e} [p_1^{w_1} \land \cdots \land p_N^{w_N}]_{Th} \bigg) \bigg]_{Th}.$$ 

But the inner disjunction can be rewritten, as follows:

$$\bigvee_{w \in D} [p_1^{w_1} \land \cdots \land p_N^{w_N}]_{Th} = [p_j^v]_{Th} \land \bigvee_{w \in D} [p_k^{w_k}]_{Th}, \text{ by the distributive law}$$

$$\text{for } w \in D \text{ and } w_j = v.$$
The formalisation of FH edit generation in terms of propositional logic

\[
\text{ec}(e) = \left[ \bigvee_{j=1}^{N} \bigvee_{v \in A_j} p_j^v \right]_{\text{Th}}, \quad \text{as required.}
\]

Thus, for the normal edit \( e \), the equivalence class \( \text{ec}(e) \) contains at least one positive clause. Indeed, if \( e \) is non-empty, we will show next that \( \text{ec}(e) \) contains exactly one positive clause, which can be taken as the value of \( \text{nc}(e) \) used to formalise \( e \). Our result will in fact be a seemingly stronger statement, that there is exactly one positive clause in the Th-equivalence class of any non-Th-valid clause: it will be a corollary of the next lemma, which gives some connections between positive clauses and the sets of atoms that they contain.

**Lemma 4.2.20.** Let \( \alpha \) and \( \beta \) be positive clauses. Then:

1. \( \alpha \) is Th-valid if and only if \( \alpha \) contains an element of the set \( \text{Th2} \) (that is, for some \( j \in \{1, \ldots, N\} \), we have that \( \bigvee_{v \in A_j} p_j^v \subseteq \alpha \));

2. \( \alpha \models_{\text{Th}} \beta \) if and only if \( \alpha \subseteq \beta \) or \( \beta \equiv_{\text{Th}} \top \); and

3. \( \alpha \equiv_{\text{Th}} \beta \) if and only if \( \alpha = \beta \) or \( \alpha \) and \( \beta \) are both Th-valid.

**Proof.** In each of the three cases, the reverse direction is immediate. We consider the forward directions one by one:

1. Suppose that the forward result does not hold. Then for each \( j \) in \( \{1, \ldots, N\} \) there is a \( w_j \) in \( A_j \) such that \( p_j^{w_j} \notin \alpha \). Let \( w = (w_1, \ldots, w_N) \) and consider the truth function \( f_w \). By Lemma 4.2.18, for each \( j \) in \( \{1, \ldots, N\} \) and each \( x \) in \( A_j \) we have that \( f_w(p_j^x) = \text{true} \) if and only if \( x = w_j \). Since \( \alpha \) contains no propositional atom of the form \( p_j^{w_j} \), we have that \( f_w(\alpha) = \text{false} \), which is a contradiction.
2. Suppose the forward result does not hold. Then, since $\alpha \not\subseteq \beta$, there is a $k$ in
\{1, \ldots, N\} and a $w_k$ in $A_k$ such that $p_k^{w_k} \in \alpha$ but $p_k^{w_k} \not\in \beta$. Also, since $\beta \not\equiv \text{Th } \top$, by part 1 of this lemma we have that for each $j$ in \{1, \ldots, N\} there is an $x_j$ in $A_j$ such that $p_j^{x_j} \not\in \beta$. For $j = k$ we choose $x_k = w_k$. Let $\mathbf{x} = (x_1, \ldots, x_N)$ and consider the truth function $f_{\mathbf{x}}$. In particular, consider $f_{\mathbf{x}}(\alpha)$ and $f_{\mathbf{x}}(\beta)$:

- $f_{\mathbf{x}}(\alpha) = \text{true}$ since $p_k^{x_k} = p_k^{w_k} \in \alpha$ and $f_{\mathbf{x}}(p_k^{x_k}) = \text{true}$ by Lemma 4.2.18;
- $f_{\mathbf{x}}(\beta) = \text{false}$ because if $p_j^{v_j} \in \beta$ then $v \neq x_j$ and hence $f_{\mathbf{x}}(p_j^{v_j}) = \text{false}$ by Lemma 4.2.18.

Hence $\alpha \not\not\equiv_{\text{Th}} \beta$, which is a contradiction.

3. If $\alpha \equiv_{\text{Th}} \beta$ then, by part 2 of this lemma,

$$(\alpha \subseteq \beta \text{ or } \beta \equiv_{\text{Th}} \top) \text{ and } (\beta \subseteq \alpha \text{ or } \alpha \equiv_{\text{Th}} \top).$$

Hence

$$(\alpha = \beta) \text{ or } (\alpha \subseteq \beta \text{ and } \alpha \equiv_{\text{Th}} \top) \text{ or } (\beta \subseteq \alpha \text{ and } \beta \equiv_{\text{Th}} \top) \text{ or } (\alpha \equiv_{\text{Th}} \beta \equiv_{\text{Th}} \top).$$

But if $\alpha \subseteq \beta$ and $\alpha \equiv_{\text{Th}} \top$ then $\beta \equiv_{\text{Th}} \top$; and similarly if $\beta \subseteq \alpha$ and $\beta \equiv_{\text{Th}} \top$ then $\alpha \equiv_{\text{Th}} \top$. Hence the result follows.

\[\square\]

As foreshadowed, we can now use the above lemma to show that each Th-equivalence
class of any non-Th-valid clause contains exactly one positive clause.

**Corollary 4.2.21.** If the clause $\gamma$ is not Th-valid then the equivalence class $[\gamma]_{\text{Th}}$
contains exactly one positive clause.

**Proof.** By Corollary 4.2.9, the non-Th-valid clause $\gamma$ is Th-logically equivalent to some
non-Th-valid positive clause, which is unique by part 3 of Lemma 4.2.20. \[\square\]

Thus if the normal edit $e$ is non-empty (and hence $ec(e)$ is not Th-valid) then
we can choose the formalisation $nc(e)$ to be the one positive clause in $ec(e)$, as
calculated in Proposition 4.2.19. As to the empty edit, there are many Th-valid positive
clauses in $ec(\emptyset)$, for example all the clauses in Th2. But again we will choose the
formalisation $nc(\emptyset)$ to be the clause calculated in Proposition 4.2.19. That is,
$nc(\emptyset) = \bigvee_{j=1}^{N} \bigvee_{v \in A_j} p_j^{v_j}$, which we will call $\top \top$, as defined formally below, where we
use the fact that $A_j^\emptyset = \emptyset$.

**Definition 4.2.22.**

$$\top \top = \bigvee_{j=1}^{N} \bigvee_{v \in A_j} p_j^{v_j}.$$  

We will give a formal definition of the function $nc$ shortly. It will be in terms of
“normal clauses” which we will use to describe all the formulae of the form $nc(e)$
where $e$ is a normal edit. Thus the normal clauses are the non-Th-valid positive clauses together with the clause $\top\top$. We give a formal definition below of the set NC of normal clauses, in terms of the set PC of positive clauses.

**Definition 4.2.23.**

\[
PC = \text{the set of positive clauses of Prop(PAtm, Th)}. \\
\text{We define the set NC of normal clauses by} \ \\
NC = (PC \setminus \{\text{Th-valid clauses}\}) \cup \{\top\top\}.
\]

We confirm that a consequence of this definition is that each equivalence class of clauses contains exactly one normal clause:

**Lemma 4.2.24.** If $\gamma$ is a clause, then the equivalence class $[\gamma]_{\text{Th}}$ contains exactly one normal clause.

**Proof.**

\[
NC \cap [\gamma]_{\text{Th}} = \left((PC \setminus \{\text{Th-valid clauses}\}) \cup \{\top\top\}\right) \cap [\gamma]_{\text{Th}} \\
= \left((PC \cap [\gamma]_{\text{Th}}) \setminus \{\text{Th-valid clauses}\}\right) \cup \{\top\top\} \cap [\gamma]_{\text{Th}} \\
= \begin{cases} 
PC \cap [\gamma]_{\text{Th}}, & \text{if } \gamma \not\equiv_{\text{Th}} \top \\
\{\top\top\}, & \text{if } \gamma \equiv_{\text{Th}} \top,
\end{cases} \\
\text{each of which has exactly one element.}
\]

\[
-\]

It will be useful to have a notation for the unique normal clause in any Th-equivalence class:

**Definition 4.2.25.** If $\alpha$ is a clause, we will write $\text{norm}(\alpha)$ for the unique normal clause that is Th-logically equivalent to $\alpha$. If $\Sigma$ is a set of clauses we will write $\text{norm}(\Sigma)$ for the set of normal clauses Th-logically equivalent to the clauses of $\Sigma$.

We can now define nc as a function from normal edits to normal clauses.

**Definition 4.2.26.** The function $\text{nc} : N(D) \rightarrow NC$ is defined for each normal edit $e = A_1^e \times \cdots \times A_N^e$ by:

\[
\text{nc}(e) = \bigvee_{j=1}^N \bigvee_{v \in A_j^e} p_j^v, \ \text{where } A_j^e = A_j \setminus A_j^e.
\]

If $E$ is a set of normal edits, we write $\text{nc}(E) = \{\text{nc}(e) \mid e \in E\}$.

In order to confirm that the function nc is a formalisation we now show the four predicted properties. The first is that nc is a well-defined function and a bijection.
Lemma 4.2.27. The function $nc$ is:

1. well-defined, and
2. a bijection.

Proof.

1. To show that the function $nc$ is well-defined, we show that if $e$ is a normal edit then $nc(e)$ is a normal clause, as follows:

The clause $nc(e)$ is positive by its definition and hence if it is not Th-valid then it is automatically normal. If $nc(e)$ is Th-valid, ie $nc(e) \equiv_{Th} \top$, then $ec(e) = [\top]_{Th}$ and hence $e = \emptyset$, since $ec$ is one-to-one. Hence by the definition of $nc$, we have that $nc(e) = \top \top$. Hence if $nc(e)$ is Th-valid then it is a normal clause.

2. We first show that $nc$ is one-to-one. If $e$ and $e'$ are normal edits and $nc(e) = nc(e')$, then by Proposition 4.2.19 and the definition of $nc$, we have that $ec(e) = ec(e')$. Hence since $ec$ is one-to-one, we have that $e = e'$ and $nc$ is one-to-one.

We now show that $nc$ is onto. Suppose $\gamma \in NC$ and $\gamma = \bigcup_{j=1}^{N} \bigcup_{v \in S_{i}^{\gamma}} p_{j}^{v}$, where for each $j$, we have that $S_{i}^{\gamma} \subseteq A_{j}$. We consider two cases. Firstly, if there is a $k$ such that $S_{i}^{\gamma} = A_{k}$ then $\gamma$ is Th-valid and since $\gamma$ is a normal clause it must equal $\top \top$. Hence, by the definition of $nc$, $\gamma = nc(\emptyset)$ and $\gamma$ is in the range of $nc$. Secondly, if for all $k$ in $\{1, \ldots, N\}$ we have that $S_{i}^{\gamma} \neq A_{k}$, then $\gamma = nc(e)$ where $e = \prod_{j=1}^{N} S_{i}^{\gamma}$, which is a normal edit since none of its components is empty. Hence once again $\gamma$ is in the range of $nc$.

The second property of $nc$ is that it preserves record correctness and incorrectness. That is:

Proposition 4.2.28. If $v$ is a record in $D$ and $e$ is an edit of $D$, then

$v$ satisfies $e \iff f_{v}(nc(e)) = \text{true}$.

Proof. Follows from Lemma 4.2.17 and the fact that $[nc(e)]_{Th} = ec(e)$. \hfill $\dashv$

The third property of $nc$ is that the breakdown of an arbitrary edit $e$ into normal edits is consistent with the breakdown of the corresponding equivalence class $ec(e)$ into equivalence classes of the corresponding normal clauses.

Proposition 4.2.29. If $e$ is an edit and $E$ is a set of normal edits then $e = \bigcup_{n \in E} n$ if and only if $ec(e) = \bigwedge_{n \in E}[nc(n)]_{Th}$. \hfill $\dashv$
Proof. The result is proved by the following sequence of equivalent statements.

\[ e = \bigcup_{n \in E} n \]
\[ \iff ec(e) = ec\left( \bigcup_{n \in E} n \right), \quad \text{since } ec \text{ is a bijection} \]
\[ \text{(by Lemma / Definition 4.2.15)} \]
\[ \iff ec(e) = \bigwedge_{n \in E} ec(n), \quad \text{since } ec \text{ is a homomorphism} \]
\[ \text{(by Lemma / Definition 4.2.15)} \]
\[ \iff ec(e) = \bigwedge_{n \in E} [nc(n)]_{Th}, \quad \text{by Definition 4.2.26 for } nc. \hspace{1cm} \square \]

The fourth property of \( nc \) is that it preserves the ordering relation \( \supseteq \) on normal edits. The proof depends on the next lemma which gives a connection between normal clauses and the sets of propositional atoms that they contain. It is obtained by applying part 2 of Lemma 4.2.20 to normal clauses, obtaining a much simpler result:

**Lemma 4.2.30.** If \( \alpha \) and \( \beta \) are normal clauses, then \( \alpha \vdash_{Th} \beta \) if and only if \( \alpha \subseteq \beta \).

**Proof.** The reverse direction is immediate. To show the forward direction:

\[ \alpha \vdash_{Th} \beta \implies \alpha \subseteq \beta \text{ or } \beta = \top \top, \quad \text{by Lemma 4.2.20} \]
\[ \implies \alpha \subseteq \beta, \text{ since each normal clause subsumes } \top \top. \hspace{1cm} \square \]

We can now show, in the next lemma, that \( nc \) preserves the ordering relation on normal edits. The reason it preserves the ordering relation is that it depends on the Boolean isomorphism \( ec \). Note that \( ec \) converts the ordering relation \( \supseteq \) on normal edits into the ordering relation \( \subseteq \) on normal clauses, and so does \( nc \).

**Lemma 4.2.31.** Let \( e \) and \( e' \) be normal edits. Then \( e \supseteq e' \) if and only if \( nc(e) \subseteq nc(e') \).

**Proof.** The proof follows from the following sequence of equivalent statements.

\[ e \supseteq e' \]
\[ \iff e \subseteq e', \quad \text{by Definition 4.2.12 for } \mathcal{E} \]
\[ \iff ec(e) \leq_{\mathcal{E}_{Th}} ec(e'), \quad \text{since } ec \text{ is a Boolean isomorphism (Lemma / Definition 4.2.15)} \]
\[ \iff ec(e) \vdash_{Th} ec(e'), \quad \text{by Definition 3.3.4 for } \mathcal{E}_{Th} \]
\[ \iff nc(e) \vdash_{Th} nc(e'), \quad \text{because for each normal edit } n, nc(n) \in ec(n) \text{ (Lemma 4.2.24)} \]
\[ \iff nc(e) \subseteq nc(e'), \quad \text{by Lemma 4.2.30}. \hspace{1cm} \square \]

Thus we have shown that the function \( nc \) does in fact give a formalisation for normal edits, because

1. it is a bijection;
2. it preserves the correctness of records;
3. it preserves the breakdown of arbitrary edits into normal edits; and
4. it preserves the ordering relation on normal edits.

Before giving a summary of this section we now mention one more property of the function norm that will be useful later. Much like a congruence relation, the function norm preserves the connective $\lor$ in most cases.

**Lemma 4.2.32.** Suppose $\alpha$ and $\beta$ are clauses, and $(\alpha \lor \beta \not\equiv_{\text{Th}} \top$ or $\alpha \equiv_{\text{Th}} \top$ or $\beta \equiv_{\text{Th}} \top$). Then $\text{norm}(\alpha \lor \beta) = \text{norm}(\alpha) \lor \text{norm}(\beta)$.

**Proof.** We consider two cases.

**Case 1.** $\alpha \lor \beta \not\equiv_{\text{Th}} \top$.

$$\begin{align*}
\text{norm}(\alpha \lor \beta)|_{\text{Th}} &= \left[\alpha \lor \beta\right]|_{\text{Th}}, \text{ since for each clause } \xi, \text{ norm}(\xi) \equiv_{\text{Th}} \xi \\
&= \left[\alpha\right]|_{\text{Th}} \lor \left[\beta\right]|_{\text{Th}}, \text{ since } \equiv_{\text{Th}} \text{ is a congruence relation} \\
&= \left[\text{norm}(\alpha)\right]|_{\text{Th}} \lor \left[\text{norm}(\beta)\right]|_{\text{Th}}, \text{ since for each clause } \xi, \text{ norm}(\xi) \equiv_{\text{Th}} \xi \\
&= \left[\text{norm}(\alpha) \lor \text{norm}(\beta)\right]|_{\text{Th}}, \text{ since } \equiv_{\text{Th}} \text{ is a congruence relation}
\end{align*}$$

But by Corollary 4.2.21, since $\alpha \lor \beta \not\equiv_{\text{Th}} \top$, we have that $[\text{norm}(\alpha \lor \beta)]|_{\text{Th}}$ contains exactly one positive clause, namely $\text{norm}(\alpha \lor \beta)$ itself. Also, $\text{norm}(\alpha) \lor \text{norm}(\beta)$ is positive, so it is also the one positive clause in $[\text{norm}(\alpha \lor \beta)]|_{\text{Th}}$. Hence $\text{norm}(\alpha \lor \beta) = \text{norm}(\alpha) \lor \text{norm}(\beta)$.

**Case 2.** Without loss of generality, let $\alpha \equiv_{\text{Th}} \top$. Then $\text{norm}(\alpha) = \top\top$. Hence $\text{norm}(\alpha) \lor \text{norm}(\beta) = \top\top$ because $\text{norm}(\beta)$ is positive and thus subsumes the clause $\text{norm}(\alpha)$. Also, since $\alpha \equiv_{\text{Th}} \top$, we have that $\text{norm}(\alpha \lor \beta) = \text{norm}(\top) = \top\top = \text{norm}(\alpha) \lor \text{norm}(\beta)$, as required. $\dashv$

We now summarise the main points in this section.

**Summary: Formalisation of normal edits**

The formalisation in $\text{Prop}(\text{PAtm, Th})$ of the normal edit $e$ is the normal clause $\text{nc}(e)$, where

$$\text{nc}(e) = \bigvee_{j=1}^{N} \bigvee_{v \in A_j} p^v_j,$$

which is the unique normal clause in the equivalence class $\text{cc}(e)$. This formalisation is consistent with the breakdown of edits into normal edits in the following sense:

$$\text{if } E \text{ is a set of normal edits, then } \text{cc}(\bigcup_{e \in E} e) = \bigwedge_{e \in E} \text{nc}(e)|_{\text{Th}}.$$
The formalisation also preserves the ordering relation on normal edits in the following sense:

if $e$ and $e'$ are normal edits, then $e \succeq e'$ if and only if $\text{nc}(e) \subseteq \text{nc}(e')$.

The formalisation is consistent with the concept of the correctness of records in the sense that:

if $e$ is a normal edit, then the record $v$ satisfies $e$ if and only if $f_v(\text{nc}(e)) = \text{true}$.

### 4.2.6 Formalisation of the Fellegi-Holt edit generation function

We will formalise within $\text{Prop}(\text{PAtm, Th})$ the edit generation function FH by way of a logic function. Indeed, we will formalise any arbitrary edit generation function. In particular, we will also formalise the edit generation functions MFH and $\text{FCF}_\omega$ (for $\omega$ an ordering of the fields).

Since edit generation functions map from the power set of normal edits to the power set of normal edits, they will be formalised as deduction functions from the power set of normal clauses to the power set of normal clauses. The deduction function corresponding to FH will be written $\mathcal{F}$; the deduction function corresponding to MFH will be written $\mathcal{MF}$; and the deduction function corresponding to $\text{FCF}_\omega$ will be written $\mathcal{FCF}_\omega$.

If we were to follow the style of the previous sections, we would define each of $\mathcal{F}$, $\mathcal{MF}$ and $\mathcal{FCF}_\omega$ directly in terms of normal clauses, and then prove that the formalisations are consistent with the function $\text{nc}$. However, in this section we will go the other way and first define the three functions in terms of $\text{nc}$ in a way that forces them to be consistent with $\text{nc}$, as per the next definition. Afterwards, we will find the three functions directly in terms of normal clauses.

In general, we formalise an arbitrary edit generation function $G$ as the deduction function $\text{nc} \circ G \circ \text{nc}^{-1}$, in order to ensure consistency with the function $\text{nc}$. Applying this formula gives us the next definition.

**Definition 4.2.33.** We define the deduction functions

\[
\begin{align*}
\mathcal{F} & : \mathcal{P}(\text{NC}) \to \mathcal{P}(\text{NC}) ; \\
\mathcal{MF} & : \mathcal{P}(\text{NC}) \to \mathcal{P}(\text{NC}) ; \text{ and} \\
\mathcal{FCF}_\omega & : \mathcal{P}(\text{NC}) \to \mathcal{P}(\text{NC}) \\
\end{align*}
\]

(\text{where } \omega \text{ is an ordering of the fields})

by

\[
\begin{align*}
\mathcal{F} & = \text{nc} \circ \text{FH} \circ \text{nc}^{-1} ; \\
\mathcal{MF} & = \text{nc} \circ \text{MFH} \circ \text{nc}^{-1} ; \text{ and} \\
\mathcal{FCF}_\omega & = \text{nc} \circ \text{FCF}_\omega \circ \text{nc}^{-1}. \\
\end{align*}
\]
The above definitions of the deduction functions $\mathcal{F}$, $\mathcal{M}\mathcal{F}$, and $\mathcal{F}\mathcal{C}\mathcal{F}_\omega$ are in terms of the edit generation functions $FH$, $MFH$, and $FCF_\omega$. Yet for an independent formalisation we want the three functions $\mathcal{F}$, $\mathcal{M}\mathcal{F}$, and $\mathcal{F}\mathcal{C}\mathcal{F}_\omega$ directly in terms of their input variables, without reference to the corresponding edit functions.

We will see that each of the three functions can be expressed via a direct translation of the definitions of the parallel functions for edits. That is, $\mathcal{F}$ can be defined via an inductive definition which parallels the inductive definition for $FH$; $\mathcal{M}\mathcal{F}$ can be defined as a set of minimal clauses paralleling the definition of $MFH$ as a set of maximal edits; and $\mathcal{F}\mathcal{C}\mathcal{F}_\omega$ can be defined by an algorithm paralleling the algorithm for $FCF_\omega$. We will consider $\mathcal{F}$ and $\mathcal{M}\mathcal{F}$ in this section, while we will leave $\mathcal{F}\mathcal{C}\mathcal{F}_\omega$ to Chapter 6 about the Field Code Forest Algorithm.

We first develop an inductive definition of the function $\mathcal{F}$. Just as the function $FH$ was defined in terms of the edit function $FHG$, so the function $\mathcal{F}$ will be defined in terms of a logic function which we will call $FHD$ (standing for “Fellegi-Holt deduction”). We will ensure that $FHD$ is consistent with $FHG$ via the following definition.

**Definition 4.2.34.** The function

$$FHD : \{1, \ldots, N\} \times \mathcal{P}(NC) \to NC$$

is defined for each field $i$ and each set $\Sigma$ of normal clauses by

$$FHD(i, \Sigma) = nc \circ FHG(i, nc^{-1}(\Sigma)).$$

We will shortly show that the function $\mathcal{F}$ has an inductive definition in terms of $FHD$ which exactly parallels the inductive definition of $FH$ in terms of $FHG$. However, before proceeding, we will use the above definition to calculate the expression $FHD(i, \Sigma)$ in terms of its input variables, namely the field $i$ and normal clause set $\Sigma$, and without reference to $FHG$. We do the calculation using Definition 4.2.26 of $nc$, and Definition 2.4.1 of $FHG$.

In using the function $FHG$, we need to remember that the calculation of $FHG$ is a two-step process. The first step is to calculate the product of $N$ sets as specified in Definition 2.4.1. The second step applies only if the output of the calculation is the empty set: if any one component of the product is empty then the output is written as a product of $N$ empty sets, in order to be in normal form.

We apply the two-step process for $FHG$ by defining an interim function $FHD1$ upon which the function $FHD$ will depend. The function $FHD1$ is defined analogously to the first step of the calculation of the function $FHG$ as follows:

**Definition 4.2.35.** Let $i$ be a field and let $\Sigma$ be a set of positive clauses, where each element $\sigma$ of $\Sigma$ is written

$$\sigma = \bigvee_{j=1}^{N} \bigvee_{p_{j}^{v} \in S_{j}^{\sigma}} p_{j}^{v}, \text{ where } S_{j}^{\sigma} \subseteq A_{j}.$$
§4.2  The formalisation of FH edit generation in terms of propositional logic

Then
\[
\text{FHD1}(i, \Sigma) = \bigvee_{j=1, \ldots, N} \bigvee_{\begin{subarray}{c} j \neq i \\vspace{1pt} \end{subarray}} \bigvee_{v \in \bigcup_{\sigma \in \Sigma} S^\sigma_j} p^v_j \lor \bigvee_{v \in \bigcap_{\sigma \in \Sigma} S^\sigma_i} p^v_i.
\]

Note: \( \text{FHD1}(i, \emptyset) = \bigvee_{v \in A_i} p^v_i \in \text{Th2} \).

The function \( \text{FHD} \) can now be written directly in terms of its input variables via the function \( \text{FHD1} \), and without reference to the function \( \text{FHG} \):

Lemma 4.2.36. Let \( i \) be a field and let \( \Sigma \) be a set of normal clauses. Then
\[
\text{FHD}(i, \Sigma) = \begin{cases} 
\text{FHD1}(i, \Sigma), & \text{if } \text{FHD1}(i, \Sigma) \not\equiv_{\text{Th}} \top \\
\top \top, & \text{if } \text{FHD1}(i, \Sigma) \equiv_{\text{Th}} \top.
\end{cases}
\]

Proof. We first note that, since \( nc \) is a bijection, there is a set \( E \) of normal edits such that \( \Sigma = nc(E) \). Also, for each \( e \) in \( E \), we can use Definition 4.2.26 to write
\[
nc(e) = \bigvee_{j=1}^N \bigvee_{v \in \overline{A^e_j}} p^v_j.
\]

If we convert this to the variables in Definition 4.2.35 with \( \sigma = nc(e) \), we get, for each \( j = 1, \ldots, N \), that
\[
S^\sigma_j^{nc(e)} = \overline{A^e_j},
\]
and thus, using the fact that \( nc \) is a bijection, we get that
\[
\bigcup_{\sigma \in \Sigma} S^\sigma_j = \bigcap_{e \in E} A^e_j \text{ and that } \bigcap_{\sigma \in \Sigma} S^\sigma_j = \bigcup_{e \in E} A^e_j. \tag{4.2.4}
\]

We now consider the two cases of the lemma.

Case 1: \( \text{FHD1}(i, \Sigma) \not\equiv_{\text{Th}} \top \).
\[
\text{FHD}(i, \Sigma) = nc \circ \text{FHG}(i, E), \text{ by Definition 4.2.34, using the fact that } E = nc^{-1}(\Sigma)
\]
\[
= nc \left( \prod_{j=1}^{i-1} \bigcap_{e \in E} A^e_j \times \bigcup_{e \in E} A^e_i \times \prod_{j=i+1}^N \bigcap_{e \in E} A^e_j \right), \text{ by Definition 2.4.1}
\]
\[
= \bigvee_{j=1, \ldots, N} \bigvee_{\begin{subarray}{c} j \neq i \\vspace{1pt} \end{subarray}} \bigvee_{v \in \bigcap_{e \in E} A^e_j} p^v_j \lor \bigvee_{v \in \bigcup_{e \in E} \overline{A^e_i}} p^v_i, \text{ by Definition 4.2.26}
\]
\[
\begin{align*}
&= \bigvee_{j=1,\ldots,N} \bigvee_{j \neq i} \bigvee_{v \in \bigcup_{\sigma \in \Sigma} S^j_{\sigma}} p^j_{v} \quad \bigvee_{v \in \bigcap_{\sigma \in \Sigma} S^j_{\sigma}} p^i_{v}, \text{ by Equations 4.2.4} \\
&= \text{FHD1}(i, \Sigma), \text{ by Definition 4.2.35},
\end{align*}
\]

as required.

Case 2: \(\text{FHD1}(i, \Sigma) \equiv_{\text{Th}} \top\). Using Definition 4.2.35 and Lemma 4.2.20, part 1, we have that:

\[
\bigcap_{\sigma \in \Sigma} S^i_{\sigma} = A_i, \text{ or there exists } j \neq i \text{ such that } \bigcup_{\sigma \in \Sigma} S^j_{\sigma} = A_j.
\]

Thus, by Equations 4.2.4, we have that

\[
\bigcup_{e \in E} A^e_j = \emptyset, \text{ or there exists } j \neq i \text{ such that } \bigcap_{e \in E} A^e_j = \emptyset.
\]

Then, by Definition 2.4.1 of FHG, we have that \(\text{FHG}(i, E) = \emptyset\), and so by Definition 4.2.34, we have that \(\text{FHD}(i, \Sigma) = \top \top\), as required. \(\dashv\)

Having written the function FHD in terms of its input variables, we can now write an inductive definition of the function \(F\) in terms of its input variables.

**Lemma 4.2.37.** Let \(\Sigma\) be a set of normal clauses. Then \(F(\Sigma)\) is inductively defined as follows:

1. \(\Sigma \subseteq F(\Sigma)\).

2. If \(\Gamma \subseteq F(\Sigma)\) and \(i\) is a field, then \(\text{FHD}(i, \Gamma) \in F(\Sigma)\).

**Proof.** The proof converts the inductive definition of FH (Definition 2.4.3) to logic using the formalisation function \(nc\). Since the function \(nc\) is bijective, there is a unique normal edit set \(E\) such that \(\Sigma = nc(E)\). We consider the two parts of the lemma separately:

1. \[
\Sigma = nc(E), \quad \text{by the definition of } E \subseteq nc \circ FH(E), \quad \text{by part 1 of Definition 2.4.3} = nc \circ FH \circ nc^{-1} \circ nc(E) = F(\Sigma), \quad \text{by Definition 4.2.33}
\]
2. Suppose that $\Gamma$ is a set of normal clauses, that $i$ is a field, and that $\Gamma \subseteq F(\Sigma)$. Then

$$FHD(i, \Gamma) = \text{nc} \circ \text{FHG}(i, \text{nc}^{-1}(\Gamma)), \text{ by Definition 4.2.34}$$

$$\in \text{nc} \circ \text{FH} \circ \text{nc}^{-1}(\Gamma), \text{ since FHG}(i, \text{nc}^{-1}(\Gamma)) \in \text{FH} \circ \text{nc}^{-1}(\Gamma), \text{ by part 2 of Definition 2.4.3}$$

$$= F(\Gamma), \text{ by Definition 4.2.33}. \quad \square$$

Just as for resolution and FH edit generation, we can also give a stepwise or sequential definition of $F$.

**Proposition 4.2.38.** Let $\Sigma$ be a set of normal clauses and let $\sigma$ be a clause. Then $\sigma \in F(\Sigma)$ if and only if there exists a finite sequence of clauses $\sigma_1, \ldots, \sigma_n = \sigma$ such that for each $j = 1, \ldots, n$

1. $\sigma_j \in \Sigma$, or
2. there is a subset $X$ of $\{\sigma_1, \ldots, \sigma_{j-1}\}$ and a field $i$ such that $\sigma_j = FHD(i, X)$.

**Proof.** Apply Theorem 1.9.7 with:

1. $S$ = the set of normal clauses NC;
2. $B = \Sigma$; and
3. the relation $R$ on $P(\text{NC}) \times \text{NC}$ defined as follows: $(X, \beta) \in R$ if and only if there is a field $i$ such that $\beta = FHD(i, X)$.

$\square$

**Terminology.** The function $F$ is a deduction function, and we will refer to it as the **FH-deduction function**. When we have a set $\Sigma$ of clauses and a clause $\sigma$ with the properties given in Proposition 4.2.38, then we say that the sequence $\sigma_1, \ldots, \sigma_n = \sigma$ of Proposition 4.2.38 is an **FH-deduction of $\sigma$ from $\Sigma$**. For each $j \in \{1, \ldots, n\}$, we say that $\sigma_j$ is obtained by **one step of FH-deduction**. We also say that $\Sigma$ is the **input set of clauses** or the **starting set of clauses** for the FH-deduction. We call the clause $\sigma$ the **output clause**, or the **final clause**, or the **resultant clause**, or the **end result** of the FH-deduction.

We have now written the function $F$ in terms of its input variables in two ways, via an inductive definition, and via a sequential definition. We are now ready to consider the function $M F$, as foreshadowed.

Just as the edit generation function $M F H$ can be defined in terms of maximal edits, so the deduction function $M F$ can be defined in terms of minimal clauses. That is, on the one hand, for the normal edit set $E$, we have that $M F H(E) = \text{Max} \circ \text{FH}(E)$; while on the other hand, for the normal clause set $\Sigma$, we have $M F(\Sigma) = \text{Min} \circ \text{F}(\Sigma)$, as per the next lemma.
Lemma 4.2.39. $\mathcal{MF} = \operatorname{Min} \circ \mathcal{F}$.

Proof. First note that if $E$ is a set of normal edits then

$$\operatorname{nc} \circ \operatorname{Max}(E) = \operatorname{Min} \circ \operatorname{nc}(E),$$

since if $e_1$ and $e_2$ are normal clauses then $e_1 \supseteq e_2$ if and only if $\operatorname{nc}(e_1) \subseteq \operatorname{nc}(e_2)$, by Lemma 4.2.31.

Let $\Sigma$ be a set of normal clauses. Then

$$\mathcal{MF}(\Sigma) = \operatorname{nc} \circ \mathcal{MFH} \circ \operatorname{nc}^{-1}(\Sigma), \quad \text{by Definition 4.2.33}$$

$$= \operatorname{nc} \circ \operatorname{Max} \circ \mathcal{F} \circ \operatorname{nc}^{-1}(\Sigma), \quad \text{by Definition 2.5.4 of MFH}$$

$$= \operatorname{Min} \circ \operatorname{nc} \circ \mathcal{F} \circ \operatorname{nc}^{-1}(\Sigma), \quad \text{by above}$$

$$= \operatorname{Min} \circ \mathcal{F}(\Sigma), \quad \text{by Definition 4.2.33.}$$

Before giving a summary of this section I present the following lemma about the function FHD applied to subsumed normal clauses. This lemma is the logical equivalent of Lemma 2.4.10, and a logical generalisation of Kunnathur’s Lemma 3.1 (Kunnathur 1982, page 38).

Lemma 4.2.40. If $Y$ and $Y'$ are two sets of normal clauses; and $g : Y \rightarrow Y'$ is surjective; and $\gamma \in Y \Rightarrow g(\gamma) \subseteq \gamma$; then for any field $i$, $\operatorname{FHD}(i, Y') \subseteq \operatorname{FHD}(i, Y)$.

Proof. The proof uses the fact that the lemma is a logical formalisation of Lemma 2.4.10. We will define edit sets $X$ and $X'$ in terms of $Y$ and $Y'$, and the function $f$ in terms of $g$, and show that they satisfy the conditions of Lemma 2.4.10. We will then be able to conclude the result of Lemma 2.4.10 which we will then convert back to the required statement in terms of $Y$, $Y'$ and $g$.

Let $X = \operatorname{nc}^{-1}(Y)$,

$$X' = \operatorname{nc}^{-1}(Y'), \quad \text{and}$$

$$f = \operatorname{nc}^{-1} \circ g \circ \operatorname{nc} : X \rightarrow X'.$$

The function $f$ is well-defined because $\operatorname{nc}$ is a bijection.

We now show that $X$, $X'$ and $f$ satisfy the two conditions of Lemma 2.4.10.

The first condition, that $f$ is surjective, holds because $\operatorname{nc}$ is bijective and $g$ is surjective.

We now show the second condition, that if $\xi \in X$ then $\xi \subseteq f(\xi)$. Suppose that $\xi \in X$. Then by the definition of $X$, we have that $\operatorname{nc}(\xi) \in Y$. Replacing $\gamma$ by $\operatorname{nc}(\xi)$ in the assumed property of $g$ gives us that $g \circ \operatorname{nc}(\xi) \subseteq \operatorname{nc}(\xi)$. Since $\operatorname{nc}$ is a bijection this last expression is equivalent to $\operatorname{nc} \circ \operatorname{nc}^{-1} \circ g \circ \operatorname{nc}(\xi) \subseteq \operatorname{nc}(\xi)$. Hence by Lemma 4.2.31, $\operatorname{nc}^{-1} \circ g \circ \operatorname{nc}(\xi) \supseteq \xi$, that is $f(\xi) \supseteq \xi$, which is the second condition.
Having satisfied both conditions of Lemma 2.4.10, we can conclude its result, namely that $\text{FHG}(i, X) \subseteq \text{FHG}(i, X')$, that is $\text{FHG}(i, \text{nc}^{-1}(Y)) \subseteq \text{FHG}(i, \text{nc}^{-1}(Y'))$. By Lemma 4.2.31, this means that $\text{nc}(\text{FHG}(i, \text{nc}^{-1}(Y))) \supseteq \text{nc}(\text{FHG}(i, \text{nc}^{-1}(Y')))$. By the definition of FHD, this is the same as $\text{FHD}(i, Y) \supseteq \text{FHD}(i, Y')$.

We now summarise the main points in this section.

**Summary: Formalisation of the Fellegi-Holt edit generation functions**

The formalisation in $\text{Prop}(\text{PAtm, Th})$ of the edit generation function FH is defined in terms of the two functions FHD1 and FHD, which are defined as follows. Given a set $\Sigma$ of normal clauses and a field $i$, write each element $\sigma$ of $\Sigma$ as

$$\sigma = \bigvee_{j=1}^{N} \bigvee_{v \in S^\sigma_j} p^v_j, \text{ where } S^\sigma_j \subseteq A_j.$$  

Then

$$\text{FHD1}(i, \Sigma) = \bigvee_{j=1, \ldots, N} \bigvee_{\substack{v \in \bigcup_{\sigma \in \Sigma} S^\sigma_j \setminus S^\sigma_i \sigma \in \Sigma}} p^v_j \lor \bigvee_{v \in \bigcap_{\sigma \in \Sigma} S^\sigma_i} p^v_i,$$

and

$$\text{FHD}(i, \Sigma) = \begin{cases} \text{FHD1}(i, \Sigma), & \text{if } \text{FHD1}(i, \Sigma) \not\equiv_{\text{Th}} \top \top \top, \\ \top \top, & \text{if } \text{FHD1}(i, \Sigma) \equiv_{\text{Th}} \top \top. \end{cases}$$

The formalisation of the edit generation function FH is the deduction function $\mathcal{F} : \mathcal{P}(\mathcal{NC}) \rightarrow \mathcal{P}(\mathcal{NC})$ which is defined inductively for each clause set $\Sigma$ by

1. $\Sigma \subseteq \mathcal{F}(\Sigma)$; and
2. if $X \subseteq \mathcal{F}(\Sigma)$ and $i$ is a field, then $\text{FHD}(i, X) \in \mathcal{F}(\Sigma)$.

The function $\mathcal{F}$ is also equivalently defined sequentially: if $\Sigma$ is a set of normal clauses, then $\sigma \in \mathcal{F}(\Sigma)$ if and only if there exists a finite sequence of clauses $\sigma_1, \ldots, \sigma_n = \sigma$ and a finite sequence of clause sets $X_1, \ldots, X_n$ such that for each $j = 1, \ldots, n$

1. $X_j \subseteq \Sigma \cup \{\sigma_1, \ldots, \sigma_{j-1}\}$, and
2. there is a field $i$ such that $\sigma_j = \text{FHD}(i, X_j)$.

The formalisation in $\text{Prop}(\text{PAtm, Th})$ of the edit generation function MFH is the deduction function $\mathcal{M}\mathcal{F} : \mathcal{P}(\mathcal{NC}) \rightarrow \mathcal{P}(\mathcal{NC})$, where

$$\mathcal{M}\mathcal{F} = \text{Min} \circ \mathcal{F}.$$
The formalisations are consistent with the formalisation of normal edits in the sense that
\[ F = \text{nc} \circ \text{FH} \circ \text{nc}^{-1}; \quad \text{and} \]
\[ M_F = \text{nc} \circ \text{MFH} \circ \text{nc}^{-1}. \]
The formalisation in \text{Prop}(\text{PAtm}, \text{Th}) of the edit generation function \( \text{FCF}_\omega \), for \( \omega \) an ordering of the fields, is the deduction function \( \text{FCF}_\omega : \mathcal{P}(\text{NC}) \rightarrow \mathcal{P}(\text{NC}) \) where
\[ \text{FCF}_\omega = \text{nc} \circ \text{FCF}_\omega \circ \text{nc}^{-1}. \]

4.2.7 Summary: Formalisation of FH edit generation in terms of propositional logic

We conclude this part of the chapter with a summary of its main points. We have used the logic \text{Prop}(\text{PAtm}, \text{Th}) to formalise the main concepts used in creating the function FH, namely

1. edits - the edit \( e \) is formalised as the equivalence class \( \text{ec}(e) \) of formulae;
2. records - the record \( v \) is formalised as the Th-truth function \( f_v \);
3. normal edits - the normal edit \( e \) is formalised as the normal clause \( \text{nc}(e) \);
4. FH edit generation - the function FH is formalised as the deduction function \( F \).

This formalisation preserves inter-relationships amongst the concepts. It preserves:

1. the correctness of a record with respect to an edit - the record \( v \) is correct with respect to the edit \( e \) if and only if \( f_v(\text{ec}(e)) = \text{true} \); and if \( e \) is a normal edit, then the record \( v \) is correct with respect to the edit \( e \) if and only if \( f_v(\text{nc}(e)) = \text{true} \);
2. the breakdown of an edit into a set of normal edits - if \( E \) is a set of normal edits then \( \text{ec}(\bigcup_{e \in E} e) = \bigwedge_{e \in E}[\text{nc}(e)]_{\text{Th}} \);
3. the relationship between the edits generated by the Fellegi-Holt method and the set of explicit edits - if \( E \) is a set of normal edits, then \( \text{nc} \circ \text{FH}(E) = F \circ \text{nc}(E) \).

Having set up a logical formalisation of Fellegi-Holt edit generation, we are now ready to examine the relationship between the formalisation and the resolution deduction function.

4.3 Connection between FH edit generation and Th-resolution

We have defined two deduction functions, the Th-resolution deduction function \( \mathcal{R}_\text{Th} \) (Definitions 3.2.17 and 4.2.7), and the function \( F \) (Section 4.2.6), which is the logical formalisation of the edit generation function FH. In the remainder of this chapter we
will show that $R_{Th}$ and $F$ are essentially the same, in the sense to be defined below.
The implication is that the deduction function $R_{Th}$ on clauses is essentially the same as FH edit generation on normal edits, via the logical formalisation $F$.

We first note some differences between the two functions $R_{Th}$ and $F$. Firstly, they have different domains and ranges: for $F$ the domain and range are restricted to sets of normal clauses, while for $R_{Th}$ they can be sets of any clauses. Secondly, even if we restrict the domain and range of $R_{Th}$ to only normal clauses, the two functions are not the same, as seen in the next example.

**Example 4.3.1.** Suppose that the domain has two fields, that is $N = 2$, and that $A_1 = A_2 = \{1, 2, 3, 4\}$. Let $\Sigma = \{\alpha_1, \alpha_2\}$, where

$$\alpha_1 = p_1^1 \lor p_1^2 \quad \text{and} \quad \alpha_2 = p_1^1 \lor p_3^1 \lor p_4^1.$$ 

We can calculate that $F(\Sigma) = \{\alpha_1, \alpha_2, p_1^1, \top \top\}$. We now find a normal clause in $R_{Th}(\Sigma)$ which is not in $F(\Sigma)$. We resolve $\alpha_2$ with the clause $\neg p_2^1 \lor \neg p_3^1$ of $Th1$ to obtain

$$p_1^1 \lor \neg p_2^1 \lor p_4^1,$$

which we resolve with $\alpha_1$ to obtain

$$p_1^1 \lor p_4^1.$$

Hence $R_{Th}(\Sigma)$ contains amongst others the normal clause $p_1^1 \lor p_4^1$, which is not in $F(\Sigma)$, and so $R_{Th}(\Sigma) \neq F(\Sigma)$. However, the two sets $R_{Th}(\Sigma)$ and $F(\Sigma)$ have the same set of minimal clauses with respect to subsumption, namely the singleton set $\{p_1^1\}$.

The above example demonstrates that the two functions $F$ and $R_{Th}$ are not the same even when restricted to normal clauses. However, for reasons to be explained below, in the covering set method we need consider only the minimal normal clauses (with respect to subsumption) output by the two functions. In the remainder of this chapter we will show that the two functions produce the same sets of minimal normal clauses, except in the unrealistic case when the input clause set contains only $Th$-valid clauses. We will make the statement more precise by writing it in terms of the function $\operatorname{Min} \circ F$ (which equals $\mathcal{MF}$ by Lemma 4.2.39) and the function $\mathcal{MR}_{Th}$, defined below.

**Definition 4.3.2.** Let $\Sigma$ be a set of clauses. We write $\mathcal{NR}_{Th}(\Sigma)$ for the set of normal clauses in $R_{Th}(\Sigma)$, that is

$$\mathcal{NR}_{Th}(\Sigma) = R_{Th}(\Sigma) \cap \text{NC}.$$ 

We write $\mathcal{MR}_{Th}(\Sigma)$ for the set of minimal normal clauses in $R_{Th}(\Sigma)$ using the ordering relation $\subseteq$ on clauses. That is

$$\mathcal{MR}_{Th}(\Sigma) = \operatorname{Min} \circ \mathcal{NR}_{Th}(\Sigma).$$
We can state more precisely the main result of the rest of this chapter, as follows. Given a clause set \( \Sigma \), we calculate the set \( R_{Th}(\Sigma) \). We cannot calculate \( F(\Sigma) \) because \( \Sigma \) can contain other than normal clauses, but we can calculate \( F \circ \text{norm}(\Sigma) \). We then find that the sets of minimal normal clauses are the same, that is

\[
MF \circ \text{norm}(\Sigma) = MR_{Th}(\Sigma),
\]

provided that \( \Sigma \) contains at least one non-Th-valid clause.

Why is it, that in the covering set method, we need only consider the minimal normal clauses output by \( F \) and \( R_{Th} \)? The reason lies in the second step of the covering set method, namely the finding of error localisation solutions by way of covering sets. This second step works when the edit generation function is covering set correctible. Just as the function \( MFH \) is covering set correctible, so is its logical formalisation \( MF \); we will work through the details in Chapter 5. Hence we need only consider the function \( MF \) instead of the function \( F \).

In order to show the relationship between \( MR_{Th} \) and \( MF \), we will look first at the relationship between \( NR_{Th} \) and \( F \circ \text{norm} \). In the next two sections (Sections 4.3.1 and 4.3.2), we will show that the two functions \( NR_{Th} \) and \( F \circ \text{norm} \) are mutually inferior (in the sense of Definition 1.9.4), provided that they are restricted to “realistic” sets of input clauses. We can then use the fact that mutually inferior sets have the same minimal elements (Lemma 1.9.5, part 1) to show in Section 4.3.3 that \( MF \circ \text{norm} = MR_{Th} \).

We will devote one section to each direction of the mutual inferiority relationship between \( NR_{Th} \) and \( F \circ \text{norm} \), and in the third section we will take minimal clauses to obtain the relationship between \( MR_{Th} \) and \( MF \circ \text{norm} \). That is, in the section below (Section 4.3.1), we will show that \( NR_{Th}(\Sigma) \) is inferior to \( F \circ \text{norm}(\Sigma) \), so long as the input clause set \( \Sigma \) contains a non-Th-valid clause. In the subsequent section (Section 4.3.2), we will show a reverse result: that the set \( F \circ \text{norm}(\Sigma) \) is inferior to \( NR_{Th}(\Sigma) \) for any clause set \( \Sigma \). Finally, in the third section (Section 4.3.3), we will use the previous two results to show that the relevant minimal normal clauses are the same, that is that \( MR_{Th}(\Sigma) = MF \circ \text{norm}(\Sigma) \), provided that \( \Sigma \) contains at least one non-Th-valid clause.

### 4.3.1 Inferiority of \( R_{Th} \) relative to FH

In this section we show that, when restricted to “realistic” clauses, the function \( NR_{Th} \) is inferior to the function \( F \circ \text{norm} \), where by “realistic” we mean that the input clause set contains at least one non-Th-valid clause. In more detail, the final result of the section, Proposition 4.3.8, gives us that

if \( \Sigma \) is a set of clauses containing at least one non-Th-valid clause and \( \gamma \in F \circ \text{norm}(\Sigma) \), then there is a clause \( \gamma' \) in \( NR_{Th}(\Sigma) \) such that \( \gamma' \subseteq \gamma \).

In other words, we will convert any realistic FH-deduction to a related \( R_{Th} \)-deduction. Figure 4.1 represents the result: if we start with a “realistic” set \( \Sigma \) of clauses and apply
§4.3 Connection between FH edit generation and Th-resolution

\[
\begin{align*}
\Sigma \xrightarrow{\text{Th-resolution deduction}} \gamma' \in \mathcal{N}R_{\text{Th}}(\Sigma) \\
\subseteq \\
\subseteq \\
\subseteq \\
norm(\Sigma) \xrightarrow{\text{FH-deduction}} \gamma \in \mathcal{F}_{\text{norm}}(\Sigma)
\end{align*}
\]

**Figure 4.1:** Diagrammatic representation of the main result of this section, namely Proposition 4.3.8, part 2. We start with a clause set \( \Sigma \) that contains at least one non-Th-valid clause. At the bottom of the figure, we apply a sequence of FH-deduction steps to the clause set \( \text{norm}(\Sigma) \) to obtain \( \gamma \). Then the top of the figure represents the fact that we could have obtained a normal subclause \( \gamma' \) of \( \gamma \) by a sequence of Th-resolution deduction steps from \( \Sigma \).

(at the bottom of the Figure) a sequence of FH-deduction steps to \( \text{norm}(\Sigma) \) to obtain the clause \( \gamma \), then we could have obtained (at the top of the Figure) a normal subclause of \( \gamma \) by a suitable sequence of \( \mathcal{R}_{\text{Th}} \)-deduction steps also starting from the set \( \Sigma \).

The proof of Proposition 4.3.8 depends on many preliminary steps. But the underlying idea of the proof is the connection between Th-resolution deduction and the function FHD, or more precisely its preliminary function FHD1. The resolvent of two clauses is the union of the two clauses except that two of the literals are “cancelled out”. Similarly the function FHD1 of a set of clauses gives the union of the clauses except on one field where the intersection is taken. The similarity between cancelling out and taking the intersection is spelt out in detail in the proof of Theorem 4.3.3, which is the essence of this section.

Thus the main result of this section is Theorem 4.3.3, although it deals only with the simple situation of just two input clauses and the preliminary function FHD1. Nonetheless it tells us that the result of a single application of the function FHD1 to just two clauses is essentially the same as the end result of a certain sequence of Th-resolution steps.

In order to move from Theorem 4.3.3 to the main proposition, we note that there are two sources of complexity in any FH-deduction. The first is the number of input clauses in any one FH-deduction step and the second is the number of FH-deduction steps. We handle each of the two sources of complexity in a separate step, each involving a proposition.

The first step, Proposition 4.3.7, extends Theorem 4.3.3 to any number of input clauses and also is about the function FHD, rather than FHD1. Thus Proposition 4.3.7 tells us that the result of a single application of the function FHD to a “realistic” set of clauses of any size is essentially the same as the result of a certain sequence of Th-resolution steps. The proposition is proved by induction on the number of clauses in the input set.

The second step is the above-mentioned final proposition, Proposition 4.3.8, of this section. It extends the previous proposition to any number of FH-deduction steps. Once again the proof is by induction, using the inductive definition of the function \( \mathcal{F} \).
An additional source of complexity in the proof of the main proposition is that, while $F$ is a function of only normal clauses, the function $R_{\text{Th}}$ is a function of both normal and other clauses. Therefore we will need two technical lemmas: these will be Lemmas 4.3.5 and 4.3.6, which will be presented after the first theorem.

We start with Theorem 4.3.3, where we address just one step of FH-deduction, indeed just the function $\text{FHD1}$, on two positive clauses. We will show that, up to subsumption, the result of a single application of $\text{FHD1}$ to two clauses is the same as the end result of a certain sequence of Th-resolution steps. In the proof of the theorem, the set intersection in the calculation of $\text{FHD1}$ is replicated by several “cancelling out” Th-resolution steps. The theorem is followed by a simple illustrative example.

**Theorem 4.3.3.** Suppose that $\alpha_1$ and $\alpha_2$ are positive clauses, $i$ is a field and $\gamma = \text{FHD1}(i, \{\alpha_1, \alpha_2\})$. Then $\gamma \in R_{\text{Th}}(\{\alpha_1, \alpha_2\})$ or $\alpha_1 \subseteq \gamma$ or $\alpha_2 \subseteq \gamma$.

**Proof.** Let

$$\alpha_1 = \bigvee_{j=1}^{N} \bigvee_{v \in S^i_j} p^v_j, \text{ where } S^i_j \subseteq A_j$$

and

$$\alpha_2 = \bigvee_{j=1}^{N} \bigvee_{v \in S^i_j} p^v_j, \text{ where } S^i_j \subseteq A_j.$$  

By the definition of $\text{FHD1}$,

$$\gamma = \bigvee_{v \in S^i_j \cap S^i_j} p^v_j \lor \bigvee_{j \neq i} \bigvee_{v \in S^i_j \cup S^i_j} p^v_j.$$  

We will consider two cases. In Case 1 we will assume, without loss of generality, that $S^i_j \subseteq S^i_j$; while in Case 2 we will assume that $S^i_j \not\subseteq S^i_j$ and $S^i_j \not\subseteq S^i_j$.

**Case 1:** Suppose, without loss of generality, that $S^i_j \subseteq S^i_j$. Then, by the formula for $\gamma$, we have that $\alpha_1$ subsumes $\gamma$, as required.

**Case 2:** Suppose that $S^i_j \not\subseteq S^i_j$ and $S^i_j \not\subseteq S^i_j$. Since neither $S^i_j \setminus S^i_j$ nor $S^i_j \setminus S^i_j$ is empty, let

$$S^i_j \setminus S^i_j = \{g_1, \ldots, g_m\} \text{ (where } m \geq 1),$$

and let

$$S^i_j \setminus S^i_j = \{h_1, \ldots, h_n\} \text{ (where } n \geq 1).$$

We will construct an $R_{\text{Th}}$-deduction of $\gamma$ from $\{\alpha_1, \alpha_2\}$. The $R_{\text{Th}}$-deduction consists of two phases: we will first construct, for each $s = 1, \ldots, m$ and for each $k = 1, \ldots, n$, a clause $\delta_{(s,k)}$. We will then construct, for each $t = 1, \ldots, n$, a clause $\epsilon_t$. The last clause we construct, namely $\epsilon_n$, will equal $\gamma$.

We first construct the clauses of form $\delta_{(s,k)}$. We will use the fact that for each $s = 1, \ldots, m$ and for each $k = 1, \ldots, n$, the clause $\neg p^s_i \lor \neg p^k_i \in \text{Th1}$ because $g_s \neq h_k$. 

For each $s = 1, \ldots, m$ and for each $k = 1, \ldots, n$, we define the clause $\delta_{(s,k)}$ inductively on $s$ in steps (δa) and (δb):

(δa) For each $k = 1, \ldots, n$, calculate $\delta_{(1,k)}$ by resolving the clause $\neg p_i^{h_k} \lor \neg p_i^{b_k}$ with $\alpha_1$ on $p_i^{g_1}$ to get:

$$\neg p_i^{h_k} \lor \bigvee_{v \in S_1^1 \setminus \{g_1\}} p_i^v \lor \bigvee_{j \neq i} \bigvee_{v \in S_1^1} p_j^v,$$

and then, since $S_1^1 = \{g_1, \ldots, g_m\} \cup (S_1^1 \cap S_2^1)$, rewriting the above resolvent as

$$\delta_{(1,k)} = \neg p_i^{h_k} \lor \bigvee_{l=2}^{m} p_i^{g_l} \lor \bigvee_{v \in S_1^1 \cap S_2^1} p_i^v \lor \bigvee_{j \neq i} \bigvee_{v \in S_1^1} p_j^v.$$

(δb) For each $k = 1, \ldots, n$, and for each $s = 2, \ldots, m$, calculate $\delta_{(s,k)}$ by resolving $\delta_{(s-1,k)}$ with $\neg p_i^{g_s} \lor \neg p_i^{b_k}$ on $p_i^{g_s}$ to get:

$$\delta_{(s,k)} = \neg p_i^{h_k} \lor \bigvee_{l=s+1}^{m} p_i^{g_l} \lor \bigvee_{v \in S_1^1 \cap S_2^1} p_i^v \lor \bigvee_{j \neq i} \bigvee_{v \in S_1^1} p_j^v.$$

In particular, when $s = m$, we have the clause $\delta_{(m,k)}$:

$$\delta_{(m,k)} = \neg p_i^{h_k} \lor \bigvee_{v \in S_1^1 \cap S_2^1} p_i^v \lor \bigvee_{j \neq i} \bigvee_{v \in S_1^1} p_j^v.$$

We now construct the clauses of the form $\epsilon_t$. For each $t = 1, \ldots, n$, we define the clause $\epsilon_t$ inductively on $t$ in steps (εa) and (εb):

(εa) Calculate $\epsilon_1$ by resolving the clause $\delta_{(m,1)}$ with $\alpha_2$ on $p_i^{h_1}$, and use the fact that $S_2^1 = \{h_1, \ldots, h_n\} \cup (S_1^1 \cap S_2^1)$, to get:

$$\epsilon_1 = \bigvee_{l=2}^{n} p_i^{h_l} \lor \bigvee_{v \in S_1^1 \cap S_2^1} p_i^v \lor \bigvee_{j \neq i} \bigvee_{v \in S_1^1 \cup S_2^1} p_j^v.$$

(εb) For each $t = 2, \ldots, n$, calculate $\epsilon_t$ by resolving $\delta_{(m,t)}$ with $\epsilon_{t-1}$ on $p_i^{h_t}$ to get:

$$\epsilon_t = \bigvee_{l=t+1}^{n} p_i^{h_l} \lor \bigvee_{v \in S_1^1 \cap S_2^1} p_i^v \lor \bigvee_{j \neq i} \bigvee_{v \in S_1^1 \cup S_2^1} p_j^v.$$

Then, when $t = n$, we have the clause $\epsilon_n$:

$$\epsilon_n = \bigvee_{v \in S_1^1 \cap S_2^1} p_i^v \lor \bigvee_{j \neq i} \bigvee_{v \in S_1^1 \cup S_2^1} p_j^v,$$
which equals $\gamma$. Thus the sequence $(\alpha_1, \alpha_2, \delta_{(1,1)}, \ldots, \delta_{(1,n)}, \delta_{(2,1)}, \ldots, \delta_{(m,n)}, \epsilon_1, \ldots, \epsilon_n)$ is an $R_{Th}$-deduction of $\gamma$ from the set $\{\alpha_1, \alpha_2\}$ and thus $\gamma \in R_{Th}(\{\alpha_1, \alpha_2\})$, as required.

Example 4.3.4. Suppose that $N = 2$, that $A_1 = \{1, 2, 3, 4, 5\}$, and that $A_2 = A_3 = \{1, 2\}$. We will use the same variables as in the above proof, with

$$\alpha_1 = p_1^1 \lor p_2^2 \lor p_3^5 \lor p_2^1,$$
$$\alpha_2 = p_1^3 \lor p_1^4 \lor p_1^5 \lor p_3^1,$$
and $i = 1$, so that
$$\gamma = FHD1(1, \{\alpha_1, \alpha_2\}).$$

By the definition of FHD1,
$$\gamma = p_1^5 \lor p_2^1 \lor p_3^1.$$

We will follow the sequence of Th-resolution steps given in the above proof to eventually obtain the clause $\gamma$.

Continuing to use the notation of the above proof, we write

$$\alpha_1 = \bigvee_{j=1}^2 \bigvee_{v \in S_j^1} p_v^j,$$
where $S_j^1 \subseteq A_j$, and $\alpha_2 = \bigvee_{j=1}^2 \bigvee_{v \in S_j^2} p_v^j$,

where

$$S_1^1 = \{1, 2, 5\} \quad S_2^1 = \{1\} \quad S_3^1 = \emptyset$$
$$S_1^2 = \{3, 4, 5\} \quad S_2^2 = \emptyset \quad S_3^2 = \{1\}$$

We follow Case 2 of the above proof, and use its notation for $S_1^1 \setminus S_1^2$ and $S_1^2 \setminus S_1^1$:

$$S_1^1 \setminus S_1^2 = \{1, 2\},$$
and thus $m = 2$, $g_1 = 1$, $g_2 = 2$

and

$$S_1^2 \setminus S_1^1 = \{3, 4\},$$
and thus $n = 2$, $h_1 = 3$, $h_2 = 4$.

For each $k = 1, 2$, we calculate the clause $\delta_{(1,k)}$ by resolving the clause $\neg p_1^1 \lor \neg p_1^{h_k}$ with $\alpha_1$ on $p_1^1$ to get

for $k = 1$ : 
$$\delta_{(1,1)} = \neg p_1^2 \lor p_2^2 \lor p_3^5 \lor p_2^1,$$
for $k = 2$ : 
$$\delta_{(1,2)} = \neg p_1^4 \lor p_2^2 \lor p_1^5 \lor p_2^1.$$

For each $k = 1, 2$, we now calculate the clause $\delta_{(2,k)}$ by resolving $\delta_{(1,k)}$ with the clause $\neg p_1^2 \lor \neg p_1^{h_k}$ on $p_1^2$ to get

for $k = 1$ : 
$$\delta_{(2,1)} = \neg p_1^4 \lor p_2^2 \lor p_1^3,$$
for $k = 2$ : 
$$\delta_{(2,2)} = \neg p_1^4 \lor p_1^5 \lor p_2^1.$$
We now calculate $\epsilon_1$ by resolving $\delta_{(2,1)}$ with $\alpha_2$ on $p_1^3$ to get

$$\epsilon_1 = p_1^4 \lor p_2^5 \lor p_4^1 \lor p_3^1.$$  

Finally, we calculate $\epsilon_2$ by resolving $\delta_{(2,2)}$ with $\epsilon_1$ on $p_4^1$ to get

$$\epsilon_2 = p_1^5 \lor p_1^1 \lor p_1^3,$$

which equals $\gamma$. Thus the sequence $(\alpha_1, \alpha_2, \delta_{(1,1)}, \delta_{(1,2)}, \delta_{(2,1)}, \delta_{(2,2)}, \epsilon_1, \epsilon_2)$ is an $R_{Th}$-deduction of $\gamma$ from the set $\{\alpha_1, \alpha_2\}$.

We have shown that a single application of FHD1 to two clauses can essentially be replicated by Th-resolution. We will shortly show a similar result for the function FHD applied to any number of clauses, but first we need the two foreshadowed technical lemmas about the relationship between normal clauses, the function $R_{Th}$, and function $F$.

The detailed reason for needing the two technical lemmas is as follows. Given a set $\Sigma$ of clauses we will want to relate $F \circ \text{norm}(\Sigma)$ to $R_{Th}(\Sigma)$. We would want to do this by way of the intermediate set $R_{Th} \circ \text{norm}(\Sigma)$, and the first technical lemma, Lemma 4.3.5, states that $R_{Th} \circ \text{norm}(\Sigma) \subseteq R_{Th}(\Sigma)$, but we must assume that the set $\Sigma$ contains no Th-valid clauses. So we are forced to use the subset $\Sigma^*$ of non-Th-valid clauses of $\Sigma$ and then we can say that $R_{Th} \circ \text{norm}(\Sigma^*) \subseteq R_{Th}(\Sigma^*)$. We will be able to relate $F \circ \text{norm}(\Sigma^*)$ to $R_{Th} \circ \text{norm}(\Sigma^*)$ but then we still need to relate $F \circ \text{norm}(\Sigma^*)$ to $F \circ \text{norm}(\Sigma^*)$. This relationship is the subject of Lemma 4.3.6, which states that to any clause $\gamma$ in $F \circ \text{norm}(\Sigma)$ there is a subsuming clause $\gamma^*$ in $F \circ \text{norm}(\Sigma^*)$.

The first technical lemma, Lemma 4.3.5, is in two parts. The result described above is the second part, which follows easily from the first part. The first part tells us that if a non-Th-valid clause $\alpha \in R_{Th}(\Sigma)$ then its normal equivalent $\text{norm}(\alpha)$ is also in $R_{Th}(\Sigma)$: it is proved by constructing a Th-resolution deduction of $\text{norm}(\alpha)$ from $\alpha$.

**Lemma 4.3.5.** Suppose that $\alpha$ is a clause, and that $\Sigma$ is a set of clauses. Then:

1. if $\alpha$ is not Th-valid and $\alpha \in R_{Th}(\Sigma)$, then $\text{norm}(\alpha) \in R_{Th}(\Sigma)$;
2. if $\Sigma$ contains no Th-valid clause, then $R_{Th} \circ \text{norm}(\Sigma) \subseteq R_{Th}(\Sigma)$.

**Proof.** The proof of the second part of this lemma depends on the first part.

1. Let

$$\alpha = \bigvee_{j=1}^N \left( \bigvee_{v \in S^+_j} p^v_j \lor \bigvee_{v \in S^-_j} \neg p^v_j \right), \text{ where for each } j \in \{1, \ldots, N\}, \ S^+_j, S^-_j \subseteq A_j.$$

Then for $j \in \{1, \ldots, N\}$ the set $S^-_j$ is either empty or contains exactly one element, because otherwise the clause $\alpha$ would include an element of Th1 and would be Th-valid, contrary to assumption. If $S^-_j$ contains exactly one element,
then we will write $S_j^- = \{w_j\}$. Let $J$ be the set of those values of $j$ for which $S_j^-$ is non-empty, that is

$$J = \{ j \mid S_j^- \neq \emptyset \},$$

and for future convenience we will write

$$J = \{ j_1, \ldots, j_m \}, \text{ where } m \geq 0.$$

Then $\alpha$ can be written directly in terms of $J$:

$$\alpha = \bigvee_{j=1}^{N} \bigvee_{v \in S_j^+} p_j^v \lor \bigvee_{j \in J} \neg p_{w_j}^j.$$

We obtain the clause $\text{norm}(\alpha)$ by resolution from $\alpha$ and clauses of Th2 by the following steps:

Write $\alpha_0 = \alpha$.

For each $k = 1, \ldots, m$ and given the clause $\alpha_{k-1}$ we will find the clause $\alpha_k$ by resolving the clause $\alpha_{k-1}$ with the clause $\bigvee_{v \in A_{j_k}} p_{w_k}^v$ of Th2 on $p_{w_k}^j$:

$$\alpha_k = \bigvee_{j=1}^{N} \bigvee_{v \in S_j^+} p_j^v \lor \bigvee_{j \in J \setminus \{ j_1, \ldots, j_k \}} \neg p_j^w \lor \bigvee_{j \in \{ j_1, \ldots, j_k \}} \bigvee_{v \in A_j \setminus \{ w_j \}} p_j^v.$$

The only difference between clauses $\alpha_{k-1}$ and $\alpha_k$ is that term $\neg p_{w_k}^j$ in the second disjunction of $\alpha_{k-1}$ is replaced in $\alpha_k$ by:

$$\bigvee_{v \in A_{j_k} \setminus \{ w_{j_k} \}} p_{j_k}^v,$$

which appears in the third disjunction of $\alpha_k$. Hence by Lemma 4.2.8, the two clauses $\alpha_{k-1}$ and $\alpha_k$ are Th-logically equivalent, and hence $\alpha_0$ and $\alpha_m$ are Th-logically equivalent.

When $k = m$ we get

$$\alpha_m = \bigvee_{j=1}^{N} \bigvee_{v \in S_j^+} p_j^v \lor \bigvee_{j \in J} \bigvee_{v \in A_j \setminus \{ w_j \}} p_j^v.$$

Since $\alpha_m$ is Th-logically equivalent to $\alpha_0 = \alpha$, it is not Th-valid; and since it is also positive, it is also normal. As a normal clause that is Th-logically equivalent to $\alpha$, the clause $\alpha_m = \text{norm}(\alpha)$.

Hence we have obtained $\text{norm}(\alpha)$ from $\alpha$ by way of Th-resolution steps. Since $\alpha \in \mathcal{R}_{\text{Th}}(\Sigma)$, we have also that $\text{norm}(\alpha) \in \mathcal{R}_{\text{Th}}(\Sigma)$.

2. We will show that $\text{norm}(\Sigma) \subseteq \mathcal{R}_{\text{Th}}(\Sigma)$, from which we deduce that $\mathcal{R}_{\text{Th}} \circ \text{norm}(\Sigma)$
Let \( \beta \in \text{norm}(\Sigma) \). Then \( \beta = \text{norm}(\sigma) \) for some \( \sigma \) in \( \Sigma \). By the definition of \( R_{\text{Th}} \), we have that \( \sigma \in R_{\text{Th}}(\Sigma) \). Then by the first part of this lemma, \( \beta = \text{norm}(\sigma) \in R_{\text{Th}}(\Sigma) \), since all elements of \( \Sigma \), including \( \sigma \), are non-Th-valid. Hence \( \text{norm}(\Sigma) \subseteq R_{\text{Th}}(\Sigma) \) and hence \( R_{\text{Th}} \circ \text{norm}(\Sigma) \subseteq R_{\text{Th}}(\Sigma) \), as required.

Thus the above technical lemma tells us that there is a Th-resolution deduction for \( \text{norm}(\alpha) \) if there is a Th-resolution deduction for \( \alpha \), provided that \( \alpha \) is not Th-valid. The lemma also tells us that \( R_{\text{Th}} \circ \text{norm}(\Sigma) \subseteq R_{\text{Th}}(\Sigma) \), so long as \( \Sigma \) contains only non-Th-valid clauses.

We now move to the second technical lemma. As foreshadowed, given a set \( \Sigma \) of clauses and its subset \( \Sigma^* \) of non-Th-valid clauses, this second lemma gives a relationship between \( F \circ \text{norm}(\Sigma) \) and \( F \circ \text{norm}(\Sigma^*) \). Since \( \text{norm}(\Sigma^*) = \text{norm}(\Sigma) \setminus \{ \top \top \} \), we will replace reference to \( \text{norm}(\Sigma) \) by a set \( X \) of normal clauses. The lemma is proved by using the inductive definition of the function \( F \).

**Lemma 4.3.6.** Suppose that \( X \) is a set of normal clauses and \( \gamma \in F(X) \). Then there exists a clause \( \gamma^\dagger \) in \( F(X \setminus \{ \top \top \}) \) such that \( \gamma^\dagger \subseteq \gamma \).

**Proof.** The proof is by structural induction using the inductive definition of \( F \) of Lemma 4.2.37. Let \( Z \) be the set of clauses that satisfy the properties of \( \gamma \) in the proposition statement, that is:

\[
Z = \{ \gamma \in F(X) \mid \text{there exists a } \gamma^\dagger \text{ in } F(X \setminus \{ \top \top \}) \text{ such that } \gamma^\dagger \subseteq \gamma \}.
\]

We will show that \( Z = F(X) \) using the two parts of the inductive definition of \( F \).

1. We will show that \( X \subseteq Z \). Suppose that \( \xi \in X \). If \( \xi \neq \top \top \) then \( \xi \in F(X \setminus \{ \top \top \}) \) and setting \( \xi^\dagger = \xi \) gives us that \( \xi \in Z \). On the other hand, if \( \xi = \top \top \), then any clause in \( F(X \setminus \{ \top \top \}) \), which contains only positive clauses, subsumes \( \xi \). Since \( F(X \setminus \{ \top \top \}) \) is non-empty, we can choose any one of its clauses as \( \xi^\dagger \), thus giving us that \( \xi \in Z \).

2. Suppose that \( Y \subseteq Z \) and that \( i \) is a field. Let \( \gamma = \text{FHD}(i,Y) \). We will show that \( \gamma \in Z \), by finding a clause \( \gamma^\dagger \) that satisfies the conditions for \( \gamma \) to be in \( Z \).

Firstly note that since \( Y \subseteq Z \), then for each \( \alpha \in Y \), there is a clause \( \alpha^\dagger \in F(X \setminus \{ \top \top \}) \) such that \( \alpha^\dagger \subseteq \alpha \). Let \( Y^\dagger = \{ \alpha^\dagger \mid \alpha \in Y \} \) and let \( \gamma^\dagger = \text{FHD}(i,Y^\dagger) \). This definition of \( \gamma^\dagger \) is well-defined because by definition each clause \( \alpha^\dagger \) is normal. We will show that \( \gamma^\dagger \) satisfies the conditions for \( \gamma \) to be in \( Z \). Firstly, \( \gamma^\dagger \in F(X \setminus \{ \top \top \}) \) by the definition of \( F \), using the facts that \( \gamma^\dagger = \text{FHD}(i,Y^\dagger) \) and \( Y^\dagger \subseteq F(X \setminus \{ \top \top \}) \). Secondly, \( \gamma^\dagger \subseteq \gamma \) by Lemma 4.2.40, which can be used because the function \( Y \rightarrow Y^\dagger \) that takes \( \alpha \) to \( \alpha^\dagger \) is surjective, and \( \alpha^\dagger \subseteq \alpha \). Hence \( \gamma \in Z \), as required.
Having completed the two technical lemmas we can now return to the main theme of this section, which is the relationship between FH-deduction and \( \mathcal{R}_{Th} \)-deduction. Theorem 4.3.3 told us that the application of FHD1 to two clauses can essentially be replicated by \( \mathcal{R}_{Th} \)-deduction. We now present a similar result for the function FHD applied to any number of clauses. The next proposition gives us that the result of a single application of the function FHD to any number of clauses is the same, up to subsumption, as the end result of a sequence of \( \mathcal{R}_{Th} \)-deduction steps, and the result of that sequence of \( \mathcal{R}_{Th} \)-deduction steps is a normal clause, provided that the input clause set contains at least one non-Th-valid clause.

The proposition is in two parts. The first part deals with the preliminary function FHD1, and is proved by induction on the number of input clauses. The second part is about the function FHD itself and follows from the first part.

**Proposition 4.3.7.**

1. Suppose that \( \Sigma \) is a set of positive clauses, \( i \) is a field and \( \gamma = \text{FHD1}(i, \Sigma) \). Then there exists a clause \( \gamma' \) in \( \mathcal{R}_{Th} (\Sigma) \) such that \( \gamma' \subseteq \gamma \).

2. Suppose that \( \Sigma \) is a set of normal clauses at least one of which is non-Th-valid, \( i \) is a field and \( \gamma = \text{FHD}(i, \Sigma) \). Then there exists a normal clause \( \gamma' \) in \( \mathcal{R}_{Th} (\Sigma) \) such that \( \gamma' \subseteq \gamma \).

**Proof.** The proof of the second part of this proposition depends on the first part.

1. Let \( \Sigma = \{ \alpha_1, \ldots, \alpha_n \} \), where \( n \geq 0 \). The proof of this part of the proposition is by induction on \( n \).

   If \( n = 0 \), then \( \Sigma = \emptyset \) and \( \gamma = \bigvee_{v \in \mathcal{A}, p_i^v \in \text{Th2}} \subseteq \mathcal{R}_{Th} (\Sigma) \). Let \( \gamma' = \gamma \).

   If \( n = 1 \), then \( \alpha_1 = \gamma \). Let \( \gamma' = \alpha_1 \in \mathcal{R}_{Th} (\Sigma) \).

   If \( n = 2 \), then the result follows from Theorem 4.3.3: if \( \gamma \in \mathcal{R}_{Th} (\Sigma) \) then let \( \gamma' = \gamma \); if \( \alpha_1 \subseteq \gamma \) then let \( \gamma' = \alpha_1 \); and if \( \alpha_2 \subseteq \gamma \) then let \( \gamma' = \alpha_2 \).

   Suppose that \( u \geq 3 \) and that the proposition holds for \( n = u - 1 \). We will show that the proposition also holds for \( n = u \). Figure 4.2 is a diagrammatic representation of the various clauses to be used in this part of the proof.

   Let \( \gamma_{u-1} = \text{FHD1}(i, \{ \alpha_1, \ldots, \alpha_{u-1} \}) \). The clause \( \gamma_{u-1} \) is useful because the induction assumption can be applied to it and it is simply related to \( \gamma \) by:

   \[
   \gamma = \text{FHD1}(i, \{ \gamma_{u-1}, \alpha_u \}), \quad \text{since set union and intersection are associative.}
   \]

   We will now construct the required clause \( \gamma' \) from \( \gamma_{u-1} \) in the four steps below, after which we will show that \( \gamma' \) has the required properties.

   (a) By the induction hypothesis, there exists a \( \gamma'_{u-1} \) in \( \mathcal{R}_{Th} (\{ \alpha_1, \ldots, \alpha_{u-1} \}) \) such that \( \gamma'_{u-1} \subseteq \gamma_{u-1} \).

   (b) Since \( \gamma_{u-1} \) is positive, so is \( \gamma'_{u-1} \).
§4.3 Connection between FH edit generation and Th-resolution

\[ \gamma' \in \mathcal{R}_{\text{Th}}(\{\gamma'_{u-1}, \alpha_u\}) \]

\[ \gamma'_{u-1} \in \mathcal{R}_{\text{Th}}(\{\alpha_1, \ldots, \alpha_{u-1}\}) \]

\[ \gamma_{u-1} = \text{FHD1}(i, \{\alpha_1, \ldots, \alpha_{u-1}\}) \]

\[ \gamma = \text{FHD1}(i, \{\gamma'_u, \alpha_u\}) \]

\[ \gamma = \text{FHD1}(i, \{\alpha_1, \ldots, \alpha_u\}) = \text{FHD1}(i, \{\gamma_{u-1}, \alpha_u\}) \]

\[ \alpha_1, \ldots, \alpha_{u-1} \]

\[ \gamma'_{u-1} \in \mathcal{R}_{\text{Th}}(\{\alpha_1, \ldots, \alpha_{u-1}\}) \]

\[ \alpha_u \]

Th-resolution deduction

calculation using the function FHD1 and field \( i \)

subsumption: if \( \gamma_1 \) and \( \gamma_2 \) are clauses then \( \gamma_1 \rightarrow \gamma_2 \) means that \( \gamma_1 \subseteq \gamma_2 \).

Figure 4.2: Diagrammatic representation of the clauses used in the proof of part 1 of Proposition 4.3.7 in the case \( n = u \). We are given the clauses \( \alpha_1, \ldots, \alpha_u \), represented on the left and the right extremities of the figure, and given the clause \( \gamma = \text{FHD1}(i, \{\alpha_1, \ldots, \alpha_u\}) \), represented at the bottom of the figure. We work upwards through the figure from \( \gamma \) to \( \gamma' \), as follows. We assume that the proposition holds for \( \alpha_1, \ldots, \alpha_{u-1} \) and \( \gamma_{u-1} = \text{FHD1}(i, \{\alpha_1, \ldots, \alpha_{u-1}\}) \), so that we can construct \( \gamma'_{u-1} \) in \( \mathcal{R}_{\text{Th}}(\{\alpha_1, \ldots, \alpha_{u-1}\}) \) with \( \gamma'_{u-1} \subseteq \gamma_{u-1} \). We then construct \( \gamma^* \) and \( \gamma' \) so that \( \gamma' \subseteq \gamma^* \subseteq \gamma \).

(c) Let \( \gamma^* = \text{FHD1}(i, \{\gamma'_{u-1}, \alpha_u\}) \).

(d) Then by the case \( n = 2 \), there exists a \( \gamma' \) in \( \mathcal{R}_{\text{Th}}(\{\gamma'_{u-1}, \alpha_u\}) \) such that \( \gamma' \subseteq \gamma^* \).

The first required property of \( \gamma' \) is that \( \gamma' \in \mathcal{R}_{\text{Th}}(\{\alpha_1, \ldots, \alpha_u\}) \) which follows from:

(a) \( \gamma'_{u-1} \in \mathcal{R}_{\text{Th}}(\{\alpha_1, \ldots, \alpha_{u-1}\}) \); and

(b) \( \gamma' \in \mathcal{R}_{\text{Th}}(\{\gamma'_{u-1}, \alpha_u\}) \).

The second required property, that \( \gamma' \subseteq \gamma \), follows from:

(a) \( \gamma' \subseteq \gamma^* \), from Step (d) above; and
(b) \( \gamma^* \subseteq \gamma \) because \( \gamma'_{u-1} \subseteq \gamma_{u-1} \), from the first Step (a) above, and hence

\( \gamma^* = \text{FHD}1(i, \{\gamma'_{u-1}, \alpha_u\}) \subseteq \text{FHD}1(i, \{\gamma_{u-1}, \alpha_u\}) = \gamma \).

We have now proved this part of the proposition, since \( \gamma' \in \mathcal{R}_{\text{Th}}(\{\alpha_1, \ldots, \alpha_u\}) \) and \( \gamma' \subseteq \gamma \).

2. We consider two cases: \( \gamma \neq \top \top \) and \( \gamma = \top \top \).

Case a. Suppose that \( \gamma \neq \top \top \). Then by the definition of FHD in Lemma 4.2.36, we have that \( \gamma = \text{FHD}1(i, \Sigma) \). Then by the first part of this proposition, there is a clause \( \gamma' \) in \( \mathcal{R}_{\text{Th}}(\Sigma) \) such that \( \gamma' \subseteq \gamma \). Since \( \gamma \neq \top \top \), we also have that \( \gamma' \neq \text{Th} \top \), and since \( \gamma' \) is positive it is also normal, as required.

Case b. Now suppose that \( \gamma = \top \top \). Then \( \gamma \) is subsumed by every normal clause in the set \( \mathcal{R}_{\text{Th}}(\Sigma) \), which contains normal clauses because, by assumption, some element \( \sigma \) of \( \Sigma \) is not Th-valid and so by Lemma 4.3.5, \( \text{norm}(\sigma) \in \mathcal{R}_{\text{Th}}(\Sigma) \). Hence we let \( \gamma' \) be some normal clause, such as \( \text{norm}(\sigma) \), in \( \mathcal{R}_{\text{Th}}(\Sigma) \).

\( \dashv \)

We have shown that a single step of FH-deduction on any number of clauses can, up to subsumption, be replicated by a sequence of Th-resolution steps resulting in a normal clause, provided that the input set of clauses contains at least one clause that is not Th-valid.

We can now apply the preceding results to any FH-deduction. We will show that, up to subsumption, any FH-deduction can be replicated by some \( \mathcal{R}_{\text{Th}} \)-deduction resulting in a normal clause, with the proviso, once again, that the input set of clauses contains at least one clause that is not Th-valid. The next proposition is in two parts, where the proof of the second part depends on the first part. The two parts are very similar: in the first part the input set of clauses contains only normal clauses, simplifying the proof somewhat; in the second part the input set of clauses can be any set \( \Sigma \) of clauses and we apply the function \( F \) to the set \( \text{norm}(\Sigma) \) of normal equivalents.

Once again, the proof of the first part of the proposition is by induction; in this case we use structural induction based on the inductive definition of \( F \). The second part of the proposition follows from the first part, using the two technical lemmas, Lemma 4.3.5 and Lemma 4.3.6.

**Proposition 4.3.8.**

1. **Suppose that** \( \Sigma \) **is a non-empty set of normal clauses and** \( \gamma \in F(\Sigma) \). **Then there exists a normal clause** \( \gamma' \) **in** \( \mathcal{R}_{\text{Th}}(\Sigma) \) **such that** \( \gamma' \subseteq \gamma \). **That is,** \( \mathcal{N} \mathcal{R}_{\text{Th}}(\Sigma) \) **is inferior to** \( F(\Sigma) \).

2. **Suppose that** \( \Sigma \) **is a set of clauses that contains at least one non-Th-valid clause and** \( \gamma \in F \circ \text{norm}(\Sigma) \). **Then there exists a normal clause** \( \gamma' \) **in** \( \mathcal{R}_{\text{Th}}(\Sigma) \) **such that** \( \gamma' \subseteq \gamma \). **That is,** \( \mathcal{N} \mathcal{R}_{\text{Th}}(\Sigma) \) **is inferior to** \( F \circ \text{norm}(\Sigma) \).

**Proof.** The proof of the second part of this proposition depends on the first part.
1. The proof of this part is by structural induction using the inductive definition of $F$ given in Lemma 4.2.37.

Let $Z$ be the set of clauses that satisfy the properties of $\gamma$ in the proposition statement, that is:

$Z = \{ \gamma \in F(\Sigma) \mid \text{there exists a normal clause } \gamma' \text{ in } R_{Th}(\Sigma) \text{ such that } \gamma' \subseteq \gamma \}.$

We will show that $Z = F(\Sigma)$, using the two parts of the definition of $F$.

(a) We have that $\Sigma \subseteq Z$, as follows. If $\sigma \in \Sigma$ then $\sigma$ is normal and $\sigma \in F(\Sigma)$ by the definition of $F$. Also $\sigma \in R_{Th}(\Sigma)$ by the definition of $R_{Th}$. We set $\sigma' = \sigma$. Hence $\sigma \in Z$. 

(b) Suppose that $X \subseteq Z$, and that $i$ is a field. Let $\gamma = FHD(i, X)$. We will show that $\gamma \in Z$, by finding a normal clause $\gamma'$ in $R_{Th}(\Sigma)$ such that $\gamma' \subseteq \gamma$. We first consider the case when $X = \{ \top \top \}$ or $X = \emptyset$. Then $\gamma = \top \top$ which is subsumed by each normal clause. Since, by assumption $\Sigma$ is non-empty and contains only normal clauses, we choose $\gamma'$ to be some clause in $\Sigma$. Hence $\gamma' \subseteq \gamma$. Also, since $\Sigma \subseteq R_{Th}(\Sigma)$, we have that $\gamma' \in R_{Th}(\Sigma)$.

From now on we assume that $X \neq \{ \top \top \}$ and $X \neq \emptyset$. Figure 4.3 is a diagrammatic representation of the various clauses to appear in this part of the proof.

We will construct $\gamma'$ in the following three steps, after which we will show $\gamma' \in R_{Th}(X')$.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure4.3.png}
\caption{Diagrammatic representation of the clauses in the proof of Proposition 4.3.8, part 1, case (b). We are given a clause set $X \subseteq Z$ and a clause $\gamma = \text{FHD}(i, X)$, represented at the bottom of the figure. We work upwards through the figure from $\gamma$ to $\gamma'$ as follows. We use the definition of $Z$ to construct the clause set $X'$, from which we construct the clause $\gamma^*$, using the function FHD, and the clause $\gamma'$, using resolution, such that $\gamma' \subseteq \gamma^* \subseteq \gamma$.}
\end{figure}
that $\gamma'$ has the required properties.

i. By the definition of $Z$, for each $\xi$ in $X$ there is a normal clause $\xi'$ in $R_{Th}(\Sigma)$ such that $\xi' \subseteq \xi$. Let $X' = \{\xi' \mid \xi \in X\}$.

ii. Let $\gamma^* = \text{FHD}(i, X')$, which is well-defined since all clauses in $X'$ are normal.

iii. We can use Proposition 4.3.7 part 2, because $X'$ consists of normal clauses and contains at least one non-Th-valid clause since $X \neq \{\top\top\}$ and $X \neq \emptyset$. Hence by Proposition 4.3.7, part 2, there is a normal clause $\gamma'$ such that $\gamma' \in R_{Th}(X')$ and $\gamma' \subseteq \gamma^*$.

We now show that $\gamma'$ satisfies the conditions for $\gamma$ to be in the set $Z$, as follows:

i. The clause $\gamma'$ is normal, from above.

ii. $\gamma' \in R_{Th}(\Sigma)$, because $\gamma' \in R_{Th}(X')$ and $R_{Th}(X') \subseteq R_{Th}(\Sigma)$ (since $X' \subseteq R_{Th}(\Sigma)$).

iii. $\gamma' \subseteq \gamma$ because:

A. $\gamma' \subseteq \gamma^*$ from above.

B. That $\gamma^* \subseteq \gamma$ follows from Lemma 4.2.40, which we can use because the function $X \rightarrow X'$ that takes $\xi$ to $\xi'$ is surjective, and $\xi' \subseteq \xi$. Hence $\text{FHD}(i, X') \subseteq \text{FHD}(i, X)$, that is, $\gamma^* \subseteq \gamma$.

We have now proved this part of the proposition, since $\gamma \in Z$.

2. Let $\Sigma^*$ be the set of clauses in $\Sigma$ that are not Th-valid. That is

$$\Sigma^* = \{\sigma \in \Sigma \mid \sigma \text{ is not Th-valid}\}.$$ 

Then by assumption $\Sigma^*$ is not empty.

We will find a normal clause $\gamma'$ that satisfies the required properties.

To find $\gamma'$, consider the set $\text{norm}(\Sigma^*) = \text{norm}(\Sigma) \setminus \{\top\top\}$. By Lemma 4.3.6, since $\gamma \in \mathcal{F} \circ \text{norm}(\Sigma)$, there is a clause $\gamma^\dagger$ in $\mathcal{F} \circ \text{norm}(\Sigma^*)$ such that $\gamma^\dagger \subseteq \gamma$. Then, from the first part of this proposition, using the fact that $\text{norm}(\Sigma^*)$ is not empty, we have that there is a normal clause $\gamma'$ in $R_{Th} \circ \text{norm}(\Sigma^*)$ such that $\gamma' \subseteq \gamma^\dagger$.

We now show that $\gamma'$ satisfies the required properties. Firstly, from above $\gamma'$ is normal. Secondly, $\gamma' \subseteq \gamma$ from the above results that $\gamma' \subseteq \gamma^\dagger$ and $\gamma^\dagger \subseteq \gamma$. Thirdly, $\gamma' \in R_{Th}(\Sigma)$ by the following argument. By Lemma 4.3.5, part 2, since by definition $\Sigma^*$ contains only non-Th-valid clauses, we have that $R_{Th} \circ \text{norm}(\Sigma^*) \subseteq R_{Th}(\Sigma^*)$ and hence $\gamma' \in R_{Th}(\Sigma^*)$. But $R_{Th}(\Sigma^*) \subseteq R_{Th}(\Sigma)$ since $\Sigma^* \subseteq \Sigma$. Hence $\gamma' \in R_{Th}(\Sigma)$.

Hence $\gamma'$ has the required properties.
In this section we have shown that $\mathcal{F} \circ \text{norm}$ is inferior to $\mathcal{N}\mathcal{R}_{Th}$; that is, we have shown that any FH-deduction can be replaced by an $\mathcal{R}_{Th}$-deduction with its end result a subsuming normal clause, although we do have to make the assumption that the input set of clauses is “realistic” in the sense that it contains at least one clause that is not Th-valid. In order to show that FH-deduction and $\mathcal{R}_{Th}$-deduction are essentially the same, we need to consider the reverse direction: we want to show that $\mathcal{N}\mathcal{R}_{Th}$ is inferior to $\mathcal{F} \circ \text{norm}$, that is we want to be able to start with an $\mathcal{R}_{Th}$-deduction and find a corresponding FH-deduction. This reverse direction is the subject of the next section.

### 4.3.2 Inferiority of FH relative to $\mathcal{R}_{Th}$

In this section we show that the function $\mathcal{F} \circ \text{norm}$ is inferior to the function $\mathcal{N}\mathcal{R}_{Th}$. In more detail, the final result of the section, Proposition 4.3.10, gives us that

if $\Sigma$ is a set of clauses and $\gamma \in \mathcal{N}\mathcal{R}_{Th}(\Sigma)$ then there is a clause $\gamma'$ in $\mathcal{F} \circ \text{norm}(\Sigma)$ such that $\gamma' \subseteq \gamma$.

In other words, we will convert any $\mathcal{R}_{Th}$-deduction that results in a normal clause to a related FH-deduction. Figure 4.4 represents the result: we start at the bottom of the figure with a set $\Sigma$ of clauses and apply some Th-resolution steps to it to obtain a normal clause $\gamma$; the top of the figure indicates that we could have obtained a subclause $\gamma'$ of $\gamma$ by a suitable FH-deduction starting from $\text{norm}(\Sigma)$.

Although the main result of the section is in part 2 of Proposition 4.3.10, the first part of the Proposition gives a slightly more general result, displayed in Figure 4.5. In this more general result, we allow $\gamma$ to be any clause rather than insisting that it be a normal clause; then the obtained clause $\gamma'$ is not necessarily a subclause of $\gamma$ but it does Th-logically imply $\gamma$.

The first part of Proposition 4.3.10 is proved by structural induction using the inductive definition of $\mathcal{R}_{Th}$. The induction step depends on considering the case when $\gamma$ is the resolvent of two clauses $\alpha$ and $\beta$ where the proposition already holds for $\alpha$ and $\beta$. This case is the subject of the first proposition, Proposition 4.3.9, of this section.

Thus, in Proposition 4.3.9, we convert one step of resolution to a corresponding step of FH-deduction. In this proposition, displayed as Figure 4.6, we start with two

\[
\begin{align*}
\text{norm}(\Sigma) & \xrightarrow{\text{FH-deduction}} \gamma' \in \mathcal{F} \circ \text{norm}(\Sigma) \\
\Sigma & \xrightarrow{\text{Th-resolution deduction}} \gamma \in \mathcal{N}\mathcal{R}_{Th}(\Sigma)
\end{align*}
\]

**Figure 4.4:** Diagrammatic representation of the main result of this section, namely Proposition 4.3.10, part 2. We start at the bottom of the diagram. If $\Sigma$ is a set of clauses and $\gamma \in \mathcal{N}\mathcal{R}_{Th}(\Sigma)$, then there exists a clause $\gamma'$ (at the top of the figure) in $\mathcal{F} \circ \text{norm}(\Sigma)$ such that $\gamma' \subseteq \gamma$. 

Figure 4.5: Diagrammatic representation of Proposition 4.3.10, part 1, which is a generalisation of the main result of this section. We start at the bottom of the diagram. If $\Sigma$ is a set of clauses and $\gamma \in R_{\text{Th}}(\Sigma)$, then there exists a clause $\gamma'$ (at the top of the figure) in $F \circ \text{norm}(\Sigma)$ such that $\gamma' \models_{\text{Th}} \gamma$.

Figure 4.6: Diagrammatic representation of Proposition 4.3.9. The clause $\gamma$, at the bottom of the figure, is a resolvent of the clauses $\alpha$ and $\beta$ on the literal $p_i^{\mu}$. The clause $\gamma'$, in the centre of the figure, equals $\text{FHD}(i, \{\alpha', \beta'\})$. If $\alpha' \models_{\text{Th}} \alpha$ and $\beta' \models_{\text{Th}} \beta$, then $\gamma' \models_{\text{Th}} \gamma$. 
§4.3 Connection between FH edit generation and Th-resolution

clauses, $\alpha$ and $\beta$, and apply one step of resolution to them to obtain the clause $\gamma$ at the bottom of the figure. We then start with two normal clauses, $\alpha'$ and $\beta'$, at the top of the figure, that Th-logically imply $\alpha$ and $\beta$ respectively, and apply one step of FH-deduction to $\alpha'$ and $\beta'$ to obtain the normal clause $\gamma'$. We find that the second obtained clause $\gamma'$ Th-logically implies the first obtained clause $\gamma$.

Proposition 4.3.9 is proved by writing $\alpha'$, $\beta'$ and $\alpha$, $\beta$ in terms of propositional atoms and then calculating $\gamma'$.

**Proposition 4.3.9.** Suppose that $\alpha$ and $\beta$ are clauses, neither of which is Th-valid, and that $i$ is a field and that $w \in A_i$. Suppose also that the clauses $\alpha'$, $\beta'$, $\gamma$ and $\gamma'$ are as represented in Figure 4.6 and defined as follows:

- $\alpha'$ is a normal clause with $\alpha' \models_{Th} \alpha$;
- $\beta'$ is a normal clause with $\beta' \models_{Th} \beta$;
- $\gamma$ is a resolvent of $\alpha$ and $\beta$, obtained by resolving on the literals $p_i^w$ in $\alpha$ and $-p_i^w$ in $\beta$; and
- $\gamma' = FHD(i, \{\alpha', \beta'\})$.

Then $\gamma' \models_{Th} \gamma$.

**Proof.** If $\gamma \equiv_{Th} \top$, then each clause Th-logically implies $\gamma$, and in particular, $\gamma' \models_{Th} \gamma$, as required.

From now on, assume that $\gamma \not\equiv_{Th} \top$.

We will consider two clauses that are Th-logically equivalent to $\gamma'$ and $\gamma$. The first clause, Th-logically equivalent to $\gamma'$, is $\gamma^* = FHD1(i, \{\alpha', \beta'\})$. The second clause, Th-logically equivalent to $\gamma$, is $\text{norm}(\gamma)$.

We will show that $\gamma^* \subseteq \text{norm}(\gamma)$, from which we will deduce that $\gamma' \models_{Th} \gamma$, as required.

In order to relate $\gamma^*$ to $\text{norm}(\gamma)$, we consider the literals in their building blocks $\alpha'$, $\beta'$, and $\alpha$, $\beta$, respectively. Since $\alpha'$ and $\beta'$ are normal, we will write

$$\alpha' = \bigvee_{j=1}^{N} \bigvee_{v \in a_j'} p_j^v,$$
$$\beta' = \bigvee_{j=1}^{N} \bigvee_{v \in b_j'} p_j^v,$$

where $a_j' \subseteq A_j$, and $b_j' \subseteq A_j$.

Then

$$\gamma^* = \bigvee_{j \neq i} \bigvee_{v \in a_j' \cup b_j'} p_j^v \vee \bigvee_{v \in a_i' \cap b_i'} p_i^v. \quad (4.3.1)$$
Since $\alpha$ contains $p^w_i$, and $\beta$ contains $\neg p^w_i$, we will write

$$\alpha = p^w_i \lor \delta_\alpha,$$

where $\delta_\alpha$ is a clause, and

$$\beta = \neg p^w_i \lor \delta_\beta,$$

where $\delta_\beta$ is a clause.

Then, by definition, $\gamma = \delta_\alpha \lor \delta_\beta$.

In order to consider $\text{norm}(\gamma)$, we note that $\text{norm}(\gamma) = \text{norm}(\delta_\alpha) \lor \text{norm}(\delta_\beta)$, by Lemma 4.2.32, since $\gamma \not\equiv_{\text{Th}} \top$. We will write

$$\text{norm}(\delta_\alpha) = \bigvee_{j=1}^N \bigvee_{v \in a_j} p^v_j,$$

where $a_j \subseteq A_j$, and

$$\text{norm}(\delta_\beta) = \bigvee_{j=1}^N \bigvee_{v \in b_j} p^v_j,$$

where $b_j \subseteq A_j$.

Then $\text{norm}(\gamma) = \bigvee_{j=1}^N \bigvee_{v \in a_j \cup b_j} p^v_j$. (4.3.2)

In order to relate $\gamma^*$ to $\text{norm}(\gamma)$, we will relate $a'_j$ to $a_j$, and $b'_j$ to $b_j$ (for $j = 1, \ldots, N$), by using the facts that $\alpha' \models_{\text{Th}} \alpha$ and $\beta' \models_{\text{Th}} \beta$, and hence that $\alpha' \models_{\text{Th}} \text{norm}(\alpha)$ and $\beta' \models_{\text{Th}} \text{norm}(\beta)$. Note that

$$\text{norm}(\alpha) = \text{norm}(p_i^w) \lor \text{norm}(\delta_\alpha),$$

by Lemma 4.2.32, since $\alpha \not\equiv_{\text{Th}} \top$ by assumption

$$= \bigvee_{j \neq i} \bigvee_{v \in a_j} p^v_j \lor \bigvee_{v \in a_i \cup \{w\}} p^w_i,$$

and that

$$\text{norm}(\beta) = \text{norm}(\neg p_i^w) \lor \text{norm}(\delta_\beta),$$

by Lemma 4.2.32, since $\beta \not\equiv_{\text{Th}} \top$ by assumption

$$= \bigvee_{j \neq i} \bigvee_{v \in b_j} p^v_j \lor \bigvee_{v \in b_i \cup (A_i \setminus \{w\})} p^v_i,$$

since $\text{norm}(\neg p_i^w) = \bigvee_{v \in A_i \setminus \{w\}} p^v_i$ by Lemma 4.2.8

$$= \bigvee_{j \neq i} \bigvee_{v \in b_j} p^v_j \lor \bigvee_{v \in A_i \setminus \{w\}} p^v_i,$$

since $b_i \subseteq A_i \setminus \{w\}$ - because otherwise $\beta \equiv_{\text{Th}} \top$, contrary to assumption.

Applying Lemma 4.2.30 to the expression $\alpha' \models_{\text{Th}} \text{norm}(\alpha)$ gives us that $\alpha' \subseteq \text{norm}(\alpha)$ and hence

$$j \neq i \Rightarrow a'_j \subseteq a_j$$

and

$$a'_i \subseteq a_i \cup \{w\}.$$
\[\beta' \subseteq \text{norm}(\beta)\] and hence
\[j \neq i \Rightarrow b'_j \subseteq b_j\]
and \[b'_i \subseteq A_i \setminus \{w\} \].

Hence
\[j \neq i \Rightarrow a'_j \cup b'_j \subseteq a_j \cup b_j\]
and \[a'_i \cap b'_i \subseteq (a_i \cup \{w\}) \cap (A_i \setminus \{w\}) = a_i \setminus \{w\}, \text{ since } a_i \subseteq A_i \]
\[\subseteq a_i \cup b_i.\]

Hence, using Equations 4.3.1 and 4.3.2, we have that \(\gamma^* \subseteq \text{norm}(\gamma)\), and hence that \(\gamma^* \models \text{norm}(\gamma)\). But \(\gamma^* \equiv_{\text{Th}} \gamma'\) and \(\text{norm}(\gamma) \equiv_{\text{Th}} \gamma\). Hence \(\gamma' \models_{\text{Th}} \gamma\), as required. ⊣

We have shown that a single step of resolution is, up to Th-logical implication, the same as a single step of FH-deduction. As foreshadowed at the beginning of this section, we now consider multiple steps of resolution. In the next proposition we start with a set of clauses, rather than just two clauses, and undertake several steps of resolution to find a clause \(\gamma\). As displayed on page 130 in Figure 4.5, we then find that had we applied a suitable FH-deduction to the normal equivalents of our starting set of clauses we could have obtained a clause that Th-logically implies \(\gamma\). The second part of this proposition, displayed in Figure 4.4 on page 129, assumes that \(\gamma\) is a normal clause instead of a general clause, and finds that the output clause of the suitable FH-deduction subsumes \(\gamma\).

**Proposition 4.3.10.**

1. Suppose that \(\Sigma\) is a set of clauses, and \(\gamma \in R_{\text{Th}}(\Sigma)\). Then there is a clause \(\gamma'\) in \(\mathcal{F} \circ \text{norm}(\Sigma)\) such that \(\gamma' \models_{\text{Th}} \gamma\).

2. Suppose that \(\Sigma\) is a set of clauses, and \(\gamma \in NR_{\text{Th}}(\Sigma)\). Then there is a clause \(\gamma'\) in \(\mathcal{F} \circ \text{norm}(\Sigma)\) such that \(\gamma' \subseteq \gamma\). That is, \(\mathcal{F} \circ \text{norm}(\Sigma)\) is inferior to \(NR_{\text{Th}}(\Sigma)\).

**Proof.** The proof of the second part of this proposition depends on the first part.

1. The proof of this part is by structural induction using the inductive definition of \(R_{\text{Th}}\).

Let \(Z\) be the set of clauses that satisfy the properties of \(\gamma\) in the proposition, that is:
\[Z = \{\gamma \in R_{\text{Th}}(\Sigma) \mid \text{there is a } \gamma' \text{ in } \mathcal{F} \circ \text{norm}(\Sigma) \text{ such that } \gamma' \models_{\text{Th}} \gamma\}.\]

We will show that \(Z = R_{\text{Th}}(\Sigma)\) using the two parts of the inductive definition of \(R_{\text{Th}}\).
We show that $\Sigma \cup \Theta \subseteq Z$.

We first show that if $\sigma \in \Sigma$ then $\sigma \in Z$, by showing that $\sigma$ satisfies the criteria in the definition of $Z$. Since $\sigma \in \Sigma$, we have that $\sigma \in R_{Th}(\Sigma)$. Also $\text{norm}(\sigma) \in \text{norm}(\Sigma) \subseteq F \circ \text{norm}(\Sigma)$; and $\text{norm}(\sigma) \models_{Th} \sigma$ by the definition of norm. Let $\sigma' = \text{norm}(\sigma)$, and hence $\sigma \in Z$.

We now show that if $\tau \in \Theta$ then $\tau \in Z$, by showing that $\tau$ satisfies the criteria in the definition of $Z$. Since $\tau \in \Theta$, we have that $\tau \in R_{Th}(\Sigma)$. Also, every clause $\Theta$-logically implies $\tau$, and, since $F \circ \text{norm}(\Sigma) \neq \emptyset$, we let $\tau' \in F \circ \text{norm}(\Sigma)$. Hence $\tau \in Z$.

(b) Suppose that $\alpha, \beta \in Z$, and that $\gamma$ is a resolvent of $\alpha$ and $\beta$. We will show that $\gamma \in Z$, by showing that $\gamma$ satisfies the criteria in the definition of $Z$.

We have that $\gamma \in R_{Th}(\Sigma)$, by the definition of $\gamma$, since $\alpha, \beta \in R_{Th}(\Sigma)$.

We now construct a clause $\gamma'$ that satisfies the criteria in the definition of $Z$. We use the fact that, by the definition of $Z$, there are clauses $\alpha'$ and $\beta'$ in $F \circ \text{norm}(\Sigma)$ such that $\alpha' \models_{Th} \alpha$ and $\beta' \models_{Th} \beta$. We consider two cases.

Case i. Suppose that neither $\alpha$ nor $\beta$ is $\Theta$-valid. Also, without loss of generality, suppose that the resolvent $\gamma$ is on $p_i^w$ of $\alpha$ and $\neg p_i^w$ of $\beta$. Let $\gamma' = \text{FHD}(i, \{\alpha', \beta'\})$, which is well-defined because $\alpha'$ and $\beta'$ are normal. Then $\gamma' \in F \circ \text{norm}(\Sigma)$ since $\alpha', \beta' \in F \circ \text{norm}(\Sigma)$. Also, since the conditions of Proposition 4.3.9 are satisfied, we have that $\gamma' \models_{Th} \gamma$, as required.

Case ii. Suppose that at least one of $\alpha$ and $\beta$ is $\Theta$-valid, and without loss of generality assume that $\alpha$ is $\Theta$-valid. Then, since $\alpha, \beta \models \gamma$, we have that $\beta \models \gamma$. But since $\beta' \models_{Th} \beta$, we have that $\beta' \models_{Th} \gamma$. Also $\beta' \in F \circ \text{norm}(\Sigma)$, from above. Let $\gamma' = \beta'$.

Hence in either case, $\gamma' \in F \circ \text{norm}(\Sigma)$ and $\gamma' \models_{Th} \gamma$, and hence $\gamma \in Z$.

2. This part of proposition follows from the first part because, if $\gamma'$ and $\gamma$ are normal clauses and $\gamma' \models_{Th} \gamma$, then $\gamma' \subseteq \gamma$, by Lemma 4.2.30.

In this section we have shown that $NR_{Th}$ is inferior to $F \circ \text{norm}$: that is, we have shown that any sequence of resolution deduction steps ending with a normal clause can be replaced by a sequence of FH-deduction steps ending in a subsuming clause. We have also shown, previously, a converse, namely that $F \circ \text{norm}$ is inferior to $NR_{Th}$, so long as the two functions are restricted to a “realistic” set of input clauses. We will now use these two complementary results to show that FH-deduction and $R_{Th}$-deduction are the same up to minimal normal clauses.

4.3.3 The minimal normal clauses produced by $R_{Th}$-deduction and FH-deduction

In this section we prove that the minimal normal clauses output by $R_{Th}$-deduction and by FH-deduction are the same, provided that the input set of clauses contains
Proposition 4.3.11. Let $\Sigma$ be a set of clauses containing at least one non-$\text{Th}$-valid clause. Then $\mathcal{M}R\text{Th}(\Sigma) = \mathcal{M}F \circ \text{norm}(\Sigma)$.

Proof. We will use Lemma 1.9.5, part 1, applied to the partially ordered set $\text{NC}$ of normal clauses with the ordering relation $\subseteq$, and the two sets $\mathcal{N}R\text{Th}(\Sigma)$ and $\mathcal{F} \circ \text{norm}(\Sigma)$.

By part 2 of Proposition 4.3.8, $\mathcal{N}R\text{Th}(\Sigma)$ is inferior to $\mathcal{F} \circ \text{norm}(\Sigma)$. By part 2 of Proposition 4.3.10, $\mathcal{F} \circ \text{norm}(\Sigma)$ is inferior to $\mathcal{N}R\text{Th}(\Sigma)$. Hence by Lemma 1.9.5, part 1,

$$\text{Min} \circ \mathcal{N}R\text{Th}(\Sigma) = \text{Min} \circ \mathcal{F} \circ \text{norm}(\Sigma).$$

Hence by the definitions of $\mathcal{M}R\text{Th}$ and $\mathcal{M}F$, we have that

$$\mathcal{M}R\text{Th}(\Sigma) = \mathcal{M}F \circ \text{norm}(\Sigma).$$

\[\top\]

4.3.4 Summary: Connection between FH-edit generation and $\text{Th}$-resolution

We conclude this part of the chapter with a summary of its main points. The two deduction functions $\mathcal{R}_{\text{Th}}$ and $\mathcal{F}$ are similar because of the similarity between the cancelling out of resolution and the set intersections of Fellegi-Holt deduction. The two functions are not identical, but they do produce the same sets of minimal normal clauses, except in the unrealistic case where all of the input clauses are $\text{Th}$-valid. That is, in the realistic case when the clause set $\Sigma$ contains at least one non-$\text{Th}$-valid clause, then $\mathcal{M}R\text{Th}(\Sigma) = \mathcal{M}F \circ \text{norm}(\Sigma)$.

4.4 Conclusion

I have given a formalisation of FH-edit generation in terms of classical propositional logic. We formalised the edit generation function FH as the deduction function $\mathcal{F}$. In order to do the formalisation, we formalised edits as formulae, normal edits as certain positive clauses called normal clauses, and records as $\text{Th}$-truth functions.

The formalisation was in terms of Boolean algebras. The Boolean algebra for the set of edits is the dual power set algebra of subsets of the domain: we need the dual rather than the primal algebra because edits are expressed in terms of failure regions rather than acceptance regions. The Boolean algebra for formulae is the Lindenbaum algebra of $\text{Th}$-equivalence classes of formulae.

The formalisation preserves the main interrelationships between records, edits and edit generation. In particular, it preserves the correctness of a record with respect to an edit, and it preserves the relationship between FH-generated edits and explicit edits.
With this formalisation, Fellegi-Holt edit generation is seen to be essentially the same as propositional resolution deduction with theory Th. The implication is that we might be able to use resolution consequence finders to implement the Fellegi-Holt edit generation process. Another implication is that we might be able to use the insights of logic to better understand the editing process.

Fellegi-Holt edit generation is just one step of the Fellegi-Holt editing method. The other step is the use of covering sets to find error localisation solutions. This other step can also be formalised in terms of logic and will be the topic of the next chapter.
Chapter 5

Covering set correctibility from the perspective of logic: categorical edits

5.1 Introduction

In the previous chapter we formalised edit generation functions as logical deduction functions. In particular, we formalised the edit generation functions FH and MFH as the deduction functions $F$ and $MF$ respectively. We found that $MF$ is essentially the same as $MR_{Th}$.

However, formalising edit generation functions does not formalise the whole of the covering set method, which depends on aspects other than edit generation. Most importantly, the covering set method depends on the property of covering set correctibility.

If the formalisation of the covering set method is to be used for automation, then it is likely to be more powerful if the full method, including the property of covering set correctibility, is formalised in terms of logic. Otherwise, if only the edit generation functions are formalised, then any method based on logic will merely replicate the methods using sets. In this chapter I present a logical formalisation of the property of covering set correctibility. Summaries of the main results presented here were published in my previous papers (Boskovitz and Goré (2005) and Boskovitz, Goré, and Wong (2005)).

The property of covering set correctibility can be formalised by directly translating from the language of sets to the language of logic, using the formalisation function nc and the formalisation of each record $v$ as the function $f_v$. I present a formalisation via such a direct translation in Section 5.2. However, such a direct formalisation is not in terms of any constructs of logic that might be of use in any automation.

We can, however, approach the formalisation in a different way. We will consider the meaning of covering set correctibility via the two properties it links together, namely (a) that of yielding a correction, and (b) that of being a covering set of failed clauses. We will find in Section 5.3 that property (a) is a type of Th-satisfiability, while property (b) can be expressed in terms of whether the empty clause is in a particular clause set. But refutation completeness and soundness also link satisfiability with whether the
empty clause is in a certain set. We will see in Proposition 5.3.7 that covering set correctibility turns out to be an extension of refutation completeness and soundness.

Indeed refutation completeness turns out to be related to one direction of covering set correctibility, namely the error correction guarantee; while soundness turns out to be related to the other direction of covering set correctibility, namely error correction totality. Section 5.4 will elucidate exactly how the error correction guarantee and error correction totality are related to Th-completeness and Th-soundness. In effect the error correction guarantee and error correction totality can be seen as new types of logical completeness and soundness respectively.

We return in Section 5.5 to the original problem addressed by covering set correctibility, namely the problem of error localisation. Covering set correctibility is related to the error localisation problem just as soundness and completeness are related to the propositional satisfiability problem. The relationship extends beyond the problem statements themselves to the underlying properties in their solutions.

The results of this chapter have potential for possible automated implementations of the covering set method. The implementations could be based on the automated logical implementations for the propositional satisfiability problem but modified to take account of the covering set correctibility property.

5.2 Direct logical formalisation of covering set correctibility

Chapter 4 gave a logical formalisation for normal edits (as normal clauses), records (as Th-truth functions), fields (as the subscripts of propositional atoms), and edit generation functions (as deduction functions). In this section we will find a formalisation of the remaining aspect of the covering set method, namely of covering set correctibility.

In order to formalise covering set correctibility we will first formalise the concepts used in its definition. Therefore the section starts with a formalisation of the concepts of error localisation, covering sets, and involved fields. We will then formalise covering set correctibility, and its two directions, namely error correction guarantee and error correction totality.

We will use the same name to each of the formalised concepts as to the original concept. That is, within logic the formalised concepts will also be called error localisation, covering sets, involved fields, covering set correctibility, error correction guarantee, and error correction totality. We will also use the same symbols. The ambiguity should not cause any confusion, because the context will be clear.

After defining all the formalised concepts, we will conclude this section with a lemma that tells us that the formalisations are consistent with each other. We will also be able to show that the deduction functions $F$ and $M_F$ are covering set correctible.

We start with the problem that covering set correctibility addresses, namely error localisation. The following definition is a translation to logical formulae of Definition 2.2.8 for error localisation using edits.
Definition 5.2.1. Let $v = (v_1, \ldots, v_N)$ be a record and $\Sigma$ be a set of logical formulae. (In general, we will require $\Sigma$ to be a set of normal clauses.) \textbf{The error localisation problem} is the problem of deciding which sets of fields can be changed to correct $f_v$ with respect to $\Sigma$. That is, it is the following problem: given a set $C$ of fields and a record $v$, decide whether there is a record $w = (w_1, \ldots, w_N)$ such that

1. $C \supseteq \{j \mid w_j \neq v_j\}$, and
2. $f_w$ satisfies $\Sigma$.

If there exists a record $w$ such that a and b are satisfied, then we say that \textbf{the set $C$ yields a correction $w$ to $v$ with respect to $\Sigma$} or that \textbf{the set $C$ yields a correction $f_w$ to $f_v$ with respect to $\Sigma$} and we shall also say that that $C$ is \textbf{a solution to the error localisation problem for $v$ (or $f_v$) with respect to $\Sigma$}.

The set of solutions to the error localisation problem for the record $v$ and the set $\Sigma$ of normal clauses will be written $EL(\Sigma, v)$. This means that $C$ yields a correction to $v$ with respect to $\Sigma$ if and only if $C \in EL(\Sigma, v)$.

Note 1: $f_v$ satisfies $\Sigma$ if and only if $\emptyset \in EL(\Sigma, v)$.

Note 2: If $\Box \in \Sigma$ then $EL(\Sigma, v) = \emptyset$.

Before we can give a logical formalisation of covering set correctibility, we must formalise the concept used in its definition, namely covering sets. Since covering sets are defined in terms of involved fields, we first give a logical formalisation of the term “involved”.

Definition 5.2.2. \textbf{The field $j$ is involved} in the normal clause $\sigma$ when $\sigma$ contains a propositional atom of form $p^v_j$, where $v \in A_j$.

Note: Every field is involved in $\top \top$.

We will show below, in Lemma 5.2.7, that the above definition is consistent with the definition of “involved” for edits, that is, that the field $j$ is involved in the edit $e$ if and only if it is involved in its logical formalisation, namely the clause $nc(e)$.

We use the term “involved” to give the logical formalisation of covering sets, in the next definition. We define covering sets in terms of involved fields in the same way as we did for edits in Definition 2.3.3.

Definition 5.2.3. \textbf{A covering set $C$ of the set $\Sigma$ of normal clauses is a set of fields such that, for each clause $\sigma$ in $\Sigma$, there exists a field in $C$ that is involved in $\sigma$.} The set of all covering sets of $\Sigma$ is written $C(\Sigma)$.

Note 1: $\Sigma = \emptyset$ if and only if $\emptyset \in C(\Sigma)$.

Note 2: $\Box \in \Sigma$ if and only if $C(\Sigma) = \emptyset$.

In order to give the logical formalisation of covering set correctibility, we will use the same notation, $X$, as we used for edits, for the set of clauses failed by a Th-truth function, as per the next definition, which follows the definition used for edits (Definition 2.3.5).
Definition 5.2.4. Define $X(\Sigma, v)$ to be the set of clauses in the clause set $\Sigma$ that are failed by $f_v$. That is, $X(\Sigma, v) = \{ \sigma \in \Sigma \mid f_v(\sigma) = \text{false} \}$.

We will also use the same notation, $C_X$, as we used for edits (Definition 2.3.6), for the set of covering sets of failed clauses, as follows.

Definition 5.2.5. Define $C_X(\Sigma, v)$ to be the set of covering sets of the clauses of $\Sigma$ failed by $f_v$. That is, $C_X(\Sigma, v) = C(\{ \sigma \in \Sigma \mid f_v(\sigma) = \text{false} \})$.

We can now give a logical formalisation of covering set correctibility, and its two directions, error correction totality and error correction guarantee. Once again we follow the definitions used for edits (Definitions 2.3.14 and 2.3.16).

Definition 5.2.6. The deduction function $D$ has **error correction totality** if for any set $\Sigma$ of normal clauses and any record $v$,

$$\mathcal{E}L(\Sigma, v) \subseteq C_X(D(\Sigma), v).$$

The deduction function $D$ has the **error correction guarantee** if for any set $\Sigma$ of normal clauses and any record $v$,

$$C_X(D(\Sigma), v) \subseteq \mathcal{E}L(\Sigma, v).$$

The deduction function $D$ is **covering set correctible** if for any set $\Sigma$ of normal clauses and any record $v$,

$$C_X(D(\Sigma), v) = \mathcal{E}L(\Sigma, v).$$

The next lemma confirms that the above formalisations are consistent with the concepts that they formalise and with the formalisation function $\text{nc}$ introduced in Chapter 4, Section 4.2.5.

Lemma 5.2.7.

1. If $e$ is a normal edit, then the field $j$ is an involved field of the normal clause $\text{nc}(e)$ if and only if $j$ is an involved field of $e$.

2. If $E$ is a set of normal edits, then the field set $C$ is a covering set of $\text{nc}(E)$ if and only if $C$ is a covering set of $E$. That is, $C(\text{nc}(E)) = C(E)$.

3. Let $v$ be a record and let $E$ be a set of normal edits. Then $X(\text{nc}(E), v) = X(E, v)$.

4. Let $v$ be a record and let $E$ be a set of normal edits. Then the field set $C$ yields a correction to $f_v$ with respect to the set $\text{nc}(E)$ if and only if the set $C$ yields a correction to $v$ with respect to the set $E$. That is, $\mathcal{E}L(\text{nc}(E), v) = \mathcal{E}L(E, v)$.

5. The deduction function $D$ has error correction totality if and only if the edit generation function $\text{nc}^{-1} \circ D \circ \text{nc}$ has error correction totality.

6. The deduction function $D$ has the error correction guarantee if and only if the edit generation function $\text{nc}^{-1} \circ D \circ \text{nc}$ has the error correction guarantee.
7. The deduction function \( D \) is covering set correctible if and only if the edit generation function \( \text{nc}^{-1} \circ D \circ \text{nc} \) is covering set correctible.

Proof.

1. By definition, the statement that \( j \) is an involved field of the clause \( \text{nc}(e) \) means that there is a field value \( v \) in \( A_j \) such that \( p_j^v \subseteq \text{nc}(e) \). Also, by the definition of \( \text{nc} \) (Definition 4.2.26),

\[
p_j^v = \text{nc}(n_v),
\]

where \( n_v \) is the normal edit \( A_1 \times \cdots \times A_{j-1} \times (A_j \setminus \{v\}) \times A_{j+1} \times \cdots \times A_N \) (and this way of writing \( n_v \) is in normal form because \( A_j \) has more than one element).

Hence the statement that \( j \) is an involved field of the clause \( \text{nc}(e) \) is equivalent to saying that there is a field value \( v \) in \( A_j \) such that \( \text{nc}(n_v) \subseteq \text{nc}(e) \). The required result then follows from the following sequence of equivalent statements:

\[
\exists v \in A_j \quad \text{nc}(n_v) \subseteq \text{nc}(e) \iff \exists v \in A_j \quad \text{nc}(n_v) \models_{\text{Th}} \text{nc}(e), \quad \text{by Lemma 4.2.30}
\]

\[
\iff \exists v \in A_j \quad \text{ec}(n_v) \models_{\text{Th}} \text{ec}(e), \quad \text{since for each normal edit } x, \text{ the clause } \text{nc}(x) \text{ is in the equivalence class } \text{ec}(x), \quad \text{by Proposition 4.2.19 and Definition 4.2.26}
\]

\[
\iff \exists v \in A_j \quad n_v \supseteq e, \quad \text{since } \text{ec} \text{ is a Boolean isomorphism, by Lemma / Definition 4.2.15}
\]

\[
\iff \exists v \in A_j \quad A_j \setminus \{v\} \supseteq A'_j, \quad \text{by the definition of } n_v, \text{ and Lemma 2.2.15}
\]

\[
\iff \text{the field } j \text{ is involved in the edit } e.
\]

2. Direct consequence of part 1 of this lemma, since covering sets are defined in terms of involved fields.

3. Suppose \( e \in E \). The result follows from the property that \( \text{nc} \) preserves record correctness and incorrectness (Proposition 4.2.28); that is: \( \text{nc}(e) \in \mathcal{X} \) (\( \text{nc}(E), v \) \( \Rightarrow f_v(\text{nc}(e)) = \text{false} \Leftrightarrow v \text{ fails } e \Leftrightarrow e \in \mathcal{X}(E, v) \).

4. By definition, \( C \in \mathcal{EL}(\text{nc}(E), v) \) if and only if there is a record \( w \) that differs from \( v \) only on \( C \) such that \( f_w \) satisfies \( \text{nc}(E) \). But by Proposition 4.2.28, \( f_w \) satisfies \( \text{nc}(E) \) if and only if \( w \) satisfies \( E \). Thus \( C \in \mathcal{EL}(\text{nc}(E), v) \) if and only if there is a record \( w \) that differs from \( v \) only on \( C \) such that \( w \) satisfies \( E \), that is \( C \in \mathcal{EL}(E, v) \).

5. Let \( v \) be a record, and let \( \Sigma \) be a set of normal clauses. Since \( \text{nc} \) is a bijection (by Lemma 4.2.27), let \( E \) be the unique edit set such that \( \Sigma = \text{nc}(E) \). The error correction totality of \( D \) means that for each record \( v \) and for each normal clause set \( \Sigma \),

\[
\mathcal{EL}(\Sigma, v) \subseteq \mathcal{CX}(D(\Sigma), v).
\]
We will find expressions for each side of this equation in terms of normal edits rather than in terms of normal clauses. Firstly, the left-hand side:

$$\mathcal{EL}(\Sigma, v) = \mathcal{EL}(E, v),$$ by part 4 of this lemma.

Secondly, the right-hand side:

$$\mathcal{CX}(D(\Sigma), v) = \mathcal{CX}(nc^{-1} \circ D(\Sigma), v),$$ by part 3 of this lemma, since $nc$ is a bijection

$$= \mathcal{CX}(nc^{-1} \circ D \circ nc(E), v),$$ by the definition of $E$.

Hence since $nc$ is a bijection, error correction totality for clauses is equivalent to saying that for each record $v$ and for each set $E$ of normal edits,

$$\mathcal{EL}(E, v) \subseteq \mathcal{CX}(nc^{-1} \circ D \circ nc(E), v).$$

This is the error correction totality of $nc^{-1} \circ D \circ nc$, as required.

6. The proof is identical to the proof of part 5, except with the subset relation reversed.

7. Follows from parts 5 and 6.

An immediate corollary is that the deduction functions $\mathcal{F}$, $\mathcal{MF}$ and $\mathcal{FCF}_\omega$ (for $\omega$ an ordering of the fields) are covering set correctible, just as are the edit generation functions $FH$, $MFH$ and $FCF_\omega$ that they formalise.

**Corollary 5.2.8.** The deduction functions $\mathcal{F}$, $\mathcal{MF}$ and $\mathcal{FCF}_\omega$ (where $\omega$ is an ordering of the fields) are covering set correctible.

**Proof.** By Chapter 4, Definition 4.2.33, we have that $FH = nc^{-1} \circ \mathcal{F} \circ nc$, that $MFH = nc^{-1} \circ \mathcal{MF} \circ nc$, and that $FCF_\omega = nc^{-1} \circ \mathcal{FCF}_\omega \circ nc$. But $FH$ is covering set correctible (by Theorem 2.4.6 and Proposition 2.4.9), and $MFH$ is covering set correctible (by Corollary 2.5.9 and Corollary 2.5.14). The function $FCF_\omega$ is also covering set correctible, although we will not prove this until Chapter 6 (Lemma 6.3.1 and Proposition 6.8.5). Hence by part 7 of Lemma 5.2.7, we have that $\mathcal{F}$, $\mathcal{MF}$ and $\mathcal{FCF}_\omega$ are covering set correctible.

In this section, we have formalised covering set correctibility and the main concepts used in its construction. We have also shown that the formalisation is consistent with the definitions used for edits. The formalisation is in terms of new logical constructs, namely $\mathcal{EL}$, $\mathcal{C}$, $\mathcal{X}$, and involved fields, which are obtained via a direct translation from sets. It would be useful to instead have a formalisation in terms of some natural constructs of logic. This is the topic of the next section.
5.3 Covering set correctibility in terms of some natural constructs of logic

In this section we will find a logical expression for covering set correctibility in terms of some constructs that arise naturally in logic. Indeed we will see that the property of covering set correctibility is an extension of the properties of completeness and soundness.

Our method will be to consider separately the two properties linked together by covering set correctibility, namely that of yielding a correction and that of being a covering set of failed clauses. Theorems 5.3.5 and 5.3.6 express each of the latter two properties in terms of some naturally arising constructs of logic. From the two theorems we can derive a logical expression for covering set correctibility, presented in Proposition 5.3.7.

We will first consider the property of yielding a correction. Theorem 5.3.5 will give us a logical expression for the statement that \( C \) yields a correction to the record \( v \) with respect to the set \( \Sigma \) of normal clauses. The logical expression will be in terms of the Th-satisfiability of a related clause set, the set \( \Sigma[v, \overline{C}] \) of "reduced clauses", to be defined in Definition 5.3.1.

We will secondly consider the property of being a covering set of failed clauses. The relevant covering sets are the covering sets of the set \( \mathcal{X}(D(\Sigma), v) \), where \( D \) is a deduction function. Theorem 5.3.6 will give us a logical expression for the statement that \( C \) is a covering set of \( \mathcal{X}(D(\Sigma), v) \), although in the theorem we generalise and replace \( D(\Sigma) \) by \( \Gamma \). The logical expression will again use the reduced clauses of Definition 5.3.1: in this case it will be in terms of whether the empty clause \( \Box \) is in the reduced clause set \( (D(\Sigma))[v, \overline{C}] \).

Finally, in Proposition 5.3.7, we will link our two logical expressions, one for yielding a correction and the other for being a covering set, to produce a logical expression for covering set correctibility. Thus the covering set correctibility of \( D \) will link the Th-satisfiability of \( \Sigma[v, \overline{C}] \) with whether the empty clause is in \( (D(\Sigma))[v, \overline{C}] \). But completeness and soundness of \( D \) link similar properties: they link the Th-satisfiability of \( \Sigma \) with whether the empty clause is in \( D(\Sigma) \). Indeed covering set correctibility turns out to be an extension of completeness and soundness.

We now consider the property of yielding a correction. When we say that \( C \) yields a correction to the record \( v = (v_1, \ldots, v_N) \) with respect to the set \( \Sigma \) of normal clauses, we mean that \( \Sigma \) is Th-satisfiable subject to the restriction that each propositional atom \( p_j^x \) whose field \( j \) is in \( C \) must take the truth value \( f_{v}(p_j^x) \). Thus we should first assign the truth value \( f_{v}(p_j^x) \) for each propositional atom \( p_j^x \) whose field is in \( \overline{C} \), and then recalculate each clause \( \sigma \) in \( \Sigma \) to get a "reduced clause" \( \sigma[v, \overline{C}] \), to be defined formally below in Definition 5.3.1. It turns out that \( C \) yields a correction to the record \( v \) with respect to \( \Sigma \) if and only if the set \( \Sigma[v, \overline{C}] \) of reduced clauses is Th-satisfiable - a result given below in the Theorem 5.3.5.

Before giving a formal definition of the reduced clause \( \sigma[v, \overline{C}] \), we describe it in words as follows. If \( \sigma \) contains a propositional atom of the form \( p_j^x \) for some \( j \) in \( \overline{C} \), then the reduced clause always takes the truth value \( \text{true} \), and we make the reduced
clause equal to \( \top \) in order to make it a normal clause. On the other hand, if \( \sigma \) contains no atom of the form \( p_{y_j} \) for any \( j \) in \( \mathcal{C} \), then the reduced clause will always take the same truth value as the clause \( \sigma \setminus \{ p_{y_j} \mid j \in \mathcal{C}, y \in A_j \} \), which we will define to be the reduced clause. More formally we make the following definition, where we replace \( \mathcal{C} \) by \( Z \):

**Definition 5.3.1.** If \( \sigma \) is a normal clause, \( v = (v_1, \ldots, v_N) \) is a record, and \( Z \) is a field set, then the **reduction of \( \sigma \) by \( v \) and \( Z \)** is

\[
\sigma[v, Z] = \begin{cases} 
\sigma \setminus \{ p_{y_j} \mid j \in Z, y \in A_j \}, & \text{if for each } j \text{ in } Z \text{ we have that } p_{y_j} \notin \sigma \\
\top \top, & \text{if there is a } j \text{ in } Z \text{ such that } p_{y_j} \in \sigma.
\end{cases}
\]

Where the context is clear, we will call \( \sigma[v, Z] \) the **reduction of \( \sigma \)** or simply the **reduced clause**.

If \( \Sigma \) is a set of normal clauses and \( Z \) is a field set then the **reduction of \( \Sigma \) by \( v \) and \( Z \)** is

\[
\Sigma[v, Z] = \{ \sigma[v, Z] \mid \sigma \in \Sigma \}.
\]

Where the context is clear, we will call \( \Sigma[v, Z] \) the **reduction of \( \Sigma \)** or simply the **set of reduced clauses**.

Note 1: An equivalent statement for \( \sigma[v, Z] \) is

\[
\sigma[v, Z] = \begin{cases} 
\lor \{ p_{y_j} \in \sigma \mid j \in Z \}, & \text{if for each } j \text{ in } Z \text{ we have that } p_{y_j} \notin \sigma \\
\top \top, & \text{if there is a } j \text{ in } Z \text{ such that } p_{y_j} \in \sigma.
\end{cases}
\]

Note 2: \( \Sigma[v, Z] \) contains only normal clauses.

Note 3: In the previous papers (Boskovitz, Goré, and Wong 2005; Boskovitz and Goré 2005), the set \( \Sigma[v, Z] \) was defined to be smaller than here: I dropped \( \sigma[v, Z] \) from \( \Sigma[v, Z] \) when \( \sigma[v, Z] = \top \top \). In the latter paper, I used a different notation: instead of \( \Sigma[v, Z] \) I used \( \Sigma[v_i \mid i \notin Z] \).

Note 4: The method of obtaining the set \( \Sigma[v, Z] \) from the set \( \Sigma \) is based on the same idea as the DPLL splitting rule (Davis et al.1962) described in Chapter 3, Section 3.2.6. The main difference between the method of obtaining the set \( \Sigma[v, Z] \) and the DPLL splitting rule is that in calculating \( \Sigma[v, Z] \) we simultaneously assign truth values to all the propositional atoms in the set \( \{ p_{y_j} \mid j \in Z, y \in A_j \} \) rather than to just a single propositional atom. Also, \( \Sigma[v, Z] \) retains \( \sigma[v, Z] = \top \top \) under the second condition of the definition of \( \sigma[v, Z] \), whereas the DPLL splitting rule deletes \( \sigma[v, Z] \) under this second condition.

**Example 5.3.2.** Suppose that \( N = 3 \); that \( A_1 = A_2 = A_3 = \{1, 2, 3\} \); that \( Z = \{1\} \);
and that \( v = (3, 3, 3) \). Then

\[
\begin{align*}
(p_1^1 \lor p_2^3)[v, Z] &= p_2^3; \\
p_1^1[v, Z] &= \Box; \\
p_2^3[v, Z] &= p_2^3; \\
(p_1^3 \lor p_2^0)[v, Z] &= \top\top; \\
p_3^0[v, Z] &= \top\top.
\end{align*}
\]

Having defined reduced clauses, we are ready to express the property of yielding a correction in terms of the Th-satisfiability of reduced clauses. We will show below, in Theorem 5.3.5, that \( C \) yields a correction to the record \( v \) with respect to \( \Sigma \) if and only if \( \Sigma[v, C] \) is Th-satisfiable.

The proof of Theorem 5.3.5 depends on the following lemma and corollary comparing Th-truth functions that satisfy \( \Sigma \) to Th-truth functions that satisfy the reduced set \( \Sigma[v, Z] \), for \( Z \) a set of fields. Rather than being in terms of a set \( \Sigma \) of clauses, the lemma and corollary are in terms of an individual clause \( \sigma \).

**Lemma 5.3.3.** Let \( Z \) be a set of fields; and let \( \sigma \) be a normal clause. Let \( v = (v_1, \ldots, v_N) \), \( x = (x_1, \ldots, x_N) \) and \( w = (w_1, \ldots, w_N) \) be records such that \( w \) is the same as \( v \) on the fields in \( Z \), and \( w \) is the same as \( x \) on the fields in \( \overline{Z} \). That is, for each \( j \) in \( \{1, \ldots, N\} \),

\[
\begin{align*}
j \in Z &\implies w_j = v_j, \quad \text{and} \\
j \in \overline{Z} &\implies w_j = x_j.
\end{align*}
\]

Then \( f_w \) satisfies \( \sigma \) if and only if \( f_x \) satisfies \( \sigma[v, Z] \).

**Proof.** We consider the two cases in the definition of \( \sigma[v, Z] \).

**Case 1.** Suppose that for each \( j \) in \( Z \), we have that \( p_j^{v_j} \not\in \sigma \). Then we can deduce the required result from the following sequence of equivalent statements:

\[
f_w(\sigma) = \text{true}.
\]

\[
\iff \text{there is a field } k \text{ in } \{1, \ldots, N\} \text{ such that } p_k^{w_k} \in \sigma, \quad \text{by Lemma 4.2.18, since } \sigma \text{ is a positive clause}
\]

\[
\iff \text{there is a field } k \text{ in } Z \text{ such that } p_k^{w_k} \in \sigma, \quad \text{since if } k \in Z, \text{ then } p_k^{w_k} = p_k^{v_k} \text{ which is not in } \sigma \text{ by assumption}
\]

\[\iff \text{there is a field } k \text{ in } \overline{Z} \text{ such that } p_k^{w_k} \in \sigma[v, Z], \quad \text{by Equation 5.3.1 above}
\]

\[\iff \text{there is a field } k \text{ in } \overline{Z} \text{ such that } p_k^{w_k} \in \sigma[v, Z], \quad \text{since if } k \in \overline{Z} \text{ then } w_k = x_k
\]

\[\iff f_x(\sigma[v, Z]) = \text{true}, \quad \text{by Lemma 4.2.18, since } \sigma[v, Z] \text{ is a positive clause}.
\]

**Case 2.** Suppose that there is a field \( j \) in \( Z \) such that \( p_j^{v_j} \in \sigma \). Then by assumption \( w_j = v_j \), so that \( p_j^{w_j} \in \sigma \). Then \( f_w(\sigma) = \text{true} \). Also, since by its definition, \( \sigma[v, Z] = \top\top \), we have that \( f_w(\sigma[v, Z]) = \text{true} \), and hence we have the required result. \( \square \)
Corollary 5.3.4. Let \( Z \) be a set of fields; and let \( \sigma \) be a normal clause. Let \( v = (v_1, \ldots, v_N) \) and \( w = (w_1, \ldots, w_N) \) be records such that if \( j \in Z \) then \( w_j = v_j \). Then \( f_w \) satisfies \( \sigma \) if and only if \( f_w \) satisfies \( \sigma[v, Z] \).

\[ \text{Proof.} \quad \text{In Lemma 5.3.3, let } x = w. \quad \]

We now use the above lemma and corollary in the proof of the next theorem, giving a logical expression for the property of yielding a correction.

Theorem 5.3.5. Let \( C \) be a set of fields, let \( v \) be a record, and let \( \Sigma \) be a set of normal clauses. Then the field set \( C \) yields a correction to the Th-truth function \( f_v \) with respect to the clause set \( \Sigma \) if and only if \( \Sigma[v, C] \) is Th-satisfiable.

\[ \text{Proof.} \quad \text{Forward direction.} \quad \text{Suppose that the field set } C \text{ yields the correction } f_w \text{ to the Th-truth function } f_v \text{ with respect to the clause set } \Sigma. \text{ We write } v = (v_1, \ldots, v_N) \text{ and } w = (w_1, \ldots, w_N). \text{ Then by Definition 5.2.1,} \]

\begin{enumerate}
  \item if \( j \in C \) then \( w_j = v_j \); and
  \item \( f_w \) satisfies \( \Sigma \).
\end{enumerate}

Then, by Corollary 5.3.4 (with \( Z = C \)), the truth function \( f_w \) satisfies each clause of \( \Sigma[v, C] \), which is thereby Th-satisfiable, as required.

\textit{Backward direction.} Suppose that \( \Sigma[v, C] \) is satisfied by the Th-truth function \( f_x \), where \( v = (v_1, \ldots, v_N) \) and \( x = (x_1, \ldots, x_N) \). Define \( w = (w_1, \ldots, w_N) \) so that it differs from \( v \) only on the fields of \( C \), where it is defined to be the same as \( x \):

\[ w_j = \begin{cases} v_j, & j \in C \\ x_j, & j \in C. \end{cases} \]

Then, by Lemma 5.3.3 (with \( Z = C \)), we have that \( f_w \) Th-satisfies \( \Sigma \). Hence \( C \) yields a correction \( f_w \) to \( f_x \) with respect to \( \Sigma \), as required.

Having found a logical expression for the property of yielding a correction, we will now find a logical expression for the other component of covering set correctibility, namely the property of being a covering set of the set \( \mathcal{X}(D(\Sigma), v) \), where \( D \) is a deduction function, \( \Sigma \) is a set of normal clauses, and \( v = (v_1, \ldots, v_N) \) is a record. To simplify notation and generalise the question, we will replace \( D(\Sigma) \) by \( \Gamma \), and seek a logical expression for the property of being a covering set of \( \mathcal{X}(\Gamma, v) \).

When we say that \( C \) is a covering set of \( \mathcal{X}(\Gamma, v) \), we mean that if the propositional atoms with fields in \( \overline{C} \) are removed from any clause \( \gamma \) of \( \mathcal{X}(\Gamma, v) \), then the new clause will not be empty. But the new clause is just the clause \( \gamma[v, \overline{C}] \), as follows: since \( \gamma \in \mathcal{X}(\Gamma, v) \), we have that \( \gamma \) contains no propositional atom of form \( p_{v,j} \), and hence \( \gamma[v, \overline{C}] \) is the clause obtained by removing from \( \gamma \) the propositional atoms with fields in \( \overline{C} \). Thus the statement that \( C \) is a covering set of \( \mathcal{X}(\Gamma, v) \) means that for each \( \gamma \) in
\(\mathcal{X}(\Gamma, v)\), the clause \(\gamma[v, \overline{C}]\) is non-empty. In fact we will see in Theorem 5.3.6 that even for those clauses \(\gamma\) not in \(\mathcal{X}(\Gamma, v)\) (but still in \(\Gamma\)), the clause \(\gamma[v, C]\) is not empty. Thus the property of be a covering set can also be expressed in terms of reduced clauses: the next theorem states that \(C\) is a covering set of the set \(\mathcal{X}(\Gamma, v)\) if and only if \(\Gamma[v, C]\) does not contain the empty clause.

**Theorem 5.3.6.** If \(C\) is a set of fields and \(\Gamma\) is a set of normal clauses, then \(C\) is a covering set of \(\mathcal{X}(\Gamma, v)\) if and only if \(\square \notin \Gamma[v, C]\).

*Proof.* We partition the set \(\Gamma\) into \(\mathcal{X}(\Gamma, v)\) and its complement \(\Gamma \setminus \mathcal{X}(\Gamma, v)\) which we call \(\Delta\). We will prove a statement for each partition:

1. if \(\gamma \in \mathcal{X}(\Gamma, v)\), then \(C\) covers \(\{\gamma\}\) if and only if \(\gamma[v, C] \neq \square\);
2. \(\square \notin \Delta[v, C]\), where \(\Delta = \Gamma \setminus \mathcal{X}(\Gamma, v)\).

We will then combine the above two statements to obtain the required result.

*Proof of statement 1.* Since \(\gamma \in \mathcal{X}(\Gamma, v)\), then \(f_v(\gamma) = \text{false}\) so that for each field \(j\) we have that \(p^v_j \notin \gamma\) (by Lemma 4.2.18). Hence by Equation 5.3.1, we have that \(\gamma[v, C] = \bigvee p^v_j \in \gamma \mid j \in C\). Hence the statement that \(\gamma[v, C] \neq \square\) is equivalent to saying that there is some field \(k\) in \(C\) and some field value \(w\) in \(A_k\) such that \(p^w_k \in \gamma\), which is the same as saying that \(C\) covers \(\{\gamma\}\).

*Proof of statement 2.* Let \(\delta \in \Delta\). Then by the definition of \(\Delta\), we have that \(f_v(\delta) = \text{true}\) and thus there is a field \(k\) such that \(p^v_k \notin \delta\). We now consider the two cases in the definition of \(\delta[v, C]\):

- **Case a.** Suppose that for each \(j\) in \(C\), we have that \(p^v_j \notin \delta\). Hence \(k \in C\). Also, by Equation 5.3.1, we have that \(\delta[v, C] = \bigvee p^v_j \in \delta \mid j \in C\), which contains \(p^v_k\) since \(k \in C\). Hence \(\delta[v, C]\) is non-empty.

- **Case b.** Suppose that there is a field \(j\) in \(C\) such that \(p^v_j \notin \delta\). Then by definition \(\delta[v, C] = \top\top\), which is non-empty.

In either case \(\delta[v, C]\) is non-empty, as required.

*Combining statements 1 and 2.* The required result follows from the following sequence of equivalent statements:

- \(C\) is a covering set of \(\mathcal{X}(\Gamma, v)\)
- \(\iff\) for each clause \(\gamma\) in \(\mathcal{X}(\Gamma, v)\), the set \(C\) is a covering set of \(\{\gamma\}\)
- \(\iff\) for each clause \(\gamma\) in \(\mathcal{X}(\Gamma, v)\), the clause \(\gamma[v, C] \neq \square\), by Statement 1 above
- \(\iff\) \(\square \notin (\mathcal{X}(\Gamma, v))[v, C]\)
- \(\iff\) \(\square \notin (\mathcal{X}(\Gamma, v))[v, C] \cup \Delta[v, C]\), by Statement 2 above
- \(\iff\) \(\square \notin \Gamma[v, C]\), by the definition of \(\Delta\).
Covering set correctibility from the perspective of logic: categorical edits

With the above theorem we now have a logical expression for each of the two properties linked by covering set correctibility. We bring the two theorems together in the next proposition to give a logical expression for covering set correctibility, as well as of its two directions, error correction totality and error correction guarantee.

**Proposition 5.3.7.** Let \( D \) be a deduction function.

1. The deduction function \( D \) has error correction totality if and only if the following statement holds: For each set \( \Sigma \) of normal clauses, each field set \( Z \), and each record \( v \)

\[
\Sigma[v, Z] \text{ is Th-satisfiable } \Rightarrow \Box \notin (D(\Sigma))[v, Z].
\]

2. The deduction function \( D \) has the error correction guarantee if and only if the following statement holds: For each set \( \Sigma \) of normal clauses, each field set \( Z \), and each record \( v \)

\[
\Box \notin (D(\Sigma))[v, Z] \Rightarrow \Sigma[v, Z] \text{ is Th-satisfiable}.
\]

3. The deduction function \( D \) is covering set correctible if and only if the following statement holds: For each set \( \Sigma \) of normal clauses, each field set \( Z \), and each record \( v \)

\[
\Box \notin (D(\Sigma))[v, Z] \iff \Sigma[v, Z] \text{ is Th-satisfiable}.
\]

**Proof.** For the proof of part 1, we use the definition of error correction totality given in Definition 5.2.6. We start with the formula used in that definition and derive the following sequence of equivalent statements:

\[
E\mathcal{L}(\Sigma, v) \subseteq C\mathcal{X}(D(\Sigma), v)
\]

\[
\Leftrightarrow \text{for each field set } C, \text{ if } C \text{ yields a correction to } v \text{ with respect to } \Sigma, \text{ then } C \text{ is a covering set of } X(D(\Sigma), v) \quad \text{(by the definitions of } C \text{ and } E\mathcal{L})
\]

\[
\Leftrightarrow \text{for each field set } C, \text{ if } \Sigma[v, \overline{C}] \text{ is Th-satisfiable, then } \Box \notin (D(\Sigma))[v, \overline{C}] \quad \text{(by Theorems 5.3.5 and 5.3.6)}
\]

\[
\Leftrightarrow \text{for each field set } Z, \text{ if } \Sigma[v, Z] \text{ is Th-satisfiable, then } \Box \notin (D(\Sigma))[v, Z] \quad \text{(since there is a one-to-one correspondence between field sets and their complements)}.
\]

Since the above sequence of statements holds for any \( \Sigma \) and \( v \), then part 1 follows.

The proof of part 2 is identical in structure to the proof of part 1. In the first line of the sequence of equivalent statements, the subset relation is reversed to obtain the
definition of the error correction guarantee. In the remaining lines of the sequence of equivalent statements, the implication direction is reversed.

Part 3 follows directly from parts 1 and 2.

The expressions in the above proposition are strengthenings of refutation Th-soundness and refutation Th-completeness (here we distinguish between refutation and strong Th-soundness as we did in Chapter 3, to maintain the symmetry with refutation and strong Th-completeness). Indeed, if Z is empty, then the above proposition reduces to a statement of the refutation Th-soundness of those deduction functions that have error correction totality, and a statement of the refutation Th-completeness of those deduction functions that have the error correction guarantee. However refutation Th-soundness and refutation Th-completeness on their own are not enough to ensure error correction totality or the error correction guarantee, as will be seen in the next section.

5.4 Soundness, completeness, and covering set correctibility

In this section we look at the relationships between soundness, completeness and covering set correctibility. We will consider separately the two directions of covering set correctibility, namely error correction totality and the error correction guarantee.

We already have, as an immediate consequence of Proposition 5.3.7, that each function with error correction totality is refutation Th-sound, and that each function with the error correction guarantee is refutation Th-complete. In this section we will see that both error correction totality and the error correction guarantee have stronger, but not parallel, properties. We first deal with error correction totality: we will see in Proposition 5.4.1 that error correction totality is equivalent to strong Th-soundness. However, we will find that the parallel property does not apply to the error correction guarantee: it is not equivalent to strong Th-completeness. Instead we will show that the error correction guarantee lies between strong Th-completeness and refutation Th-completeness - in Proposition 5.4.2, Example 5.4.3, Lemma 5.4.4 and Example 5.4.5.

Just as we related soundness and completeness to satisfiability in Lemma 3.2.26, so we here will relate error correction totality and the error correction guarantee to satisfiability. We will see in Proposition 5.4.6 and Example 5.4.7 that error correction totality has an equivalent definition in terms of satisfiability, but for the error correction guarantee the corresponding property is implied rather than equivalent.

We start with the equivalence of strong Th-soundness and error correction totality.

**Proposition 5.4.1.** The deduction function \( \mathcal{D} \) has error correction totality if and only if it is strongly Th-sound.

*Proof.*

*Forward direction.* Suppose that \( \mathcal{D} \) has error correction totality. Let \( \Sigma \) be a set of normal clauses and let \( v \) be a record such that \( f_v \) satisfies \( \Sigma \), so that \( \emptyset \in \mathcal{E}(\Sigma, v) \). Then, since \( \mathcal{D} \) has error correction totality, we have that \( \emptyset \in \mathcal{E}(\Sigma, v) \), which means that
\( \mathcal{X}(D(\Sigma), v) = \emptyset \). Hence \( f_v \) satisfies \( D(\Sigma) \). This gives us that \( D \) is strongly Th-sound.

**Backward direction.** Suppose that \( D \) is strongly Th-sound. We will use the characterisation of error correction totality given by Proposition 5.3.7, part 1. Let \( \Sigma \) be a set of normal clauses; let \( v = (v_1, \ldots, v_N) \) be a record; and let \( Z \) be a set of fields. We suppose that \( \Sigma[v, Z] \) is Th-satisfiable, and we will show that \( \Box \notin (D(\Sigma))[v, Z] \).

Suppose that a Th-truth function that satisfies \( \Sigma[v, Z] \) is \( f_x \), with \( x = (x_1, \ldots, x_N) \). Let \( w = (w_1, \ldots, w_N) \) be defined for each \( j = 1, \ldots, N \) by

\[
w_j = \begin{cases} v_j, & j \in Z \\ x_j, & j \notin Z. \end{cases}
\]

Then by Lemma 5.3.3, we have that \( f_w \) satisfies \( \Sigma \). Then, by the strong Th-soundness of \( D \), we also have that \( f_w \) satisfies \( D(\Sigma) \). Then using Corollary 5.3.4, we have that \( f_w \) satisfies \( (D(\Sigma))[v, Z] \), so that \( \Box \notin (D(\Sigma))[v, Z] \), as required.

Although error correction totality is equivalent to strong Th-soundness, the error correction guarantee is not equivalent to strong Th-completeness. However, each function with strong Th-completeness does have the error correction guarantee, as seen in the next proposition.

**Proposition 5.4.2.** If \( D \) is a strongly Th-complete deduction function, then it has the error correction guarantee.

**Proof.** We will use the characterisation of the error correction guarantee given by Proposition 5.3.7, part 2. Let \( \Sigma \) be a set of normal clauses; let \( v = (v_1, \ldots, v_N) \) be a record; and let \( Z \) be a set of fields. We suppose that \( \Sigma[v, Z] \) is not Th-satisfiable, and we will show that \( \Box \notin (D(\Sigma))[v, Z] \).

Since \( \Sigma[v, Z] \) is not Th-satisfiable, then for each vector \( w \) in \( D \) there is a normal clause \( \alpha_w \) in \( \Sigma[v, Z] \) such that \( f_w(\alpha_w) = \text{false} \). Hence \( \alpha_w \neq \top \top \) and by Definition 5.3.1, there is a normal clause \( \sigma_w \) in \( \Sigma \) such that \( \alpha_w = \sigma_w \setminus \{p^y_j \mid j \in Z, y \in A_j\} \) and for each \( j \in Z \) we have that \( p^y_j \notin \sigma_w \).

Let \( \delta \) be the disjunction of those propositional atoms that are in some \( \sigma_w \) and that have their field in the set \( Z \), that is:

\[
\delta = \bigvee \{p^y_j \mid j \in Z, y \in A_j\}, \text{ and there is a } w \text{ in } D \text{ such that } p^y_j \in \sigma_w \}.
\]

We will prove two things. Firstly, we will show that \( \delta \in D(\Sigma) \). Secondly, we will show that \( \delta[v, Z] = \Box \). We will then have the required result that \( \Box \notin (D(\Sigma))[v, Z] \).

We first show that \( \delta \in D(\Sigma) \), which we obtain by showing that \( \Sigma \models_{\text{Th}} \delta \) and then using strong Th-completeness. In order to show that \( \Sigma \models_{\text{Th}} \delta \), let \( f_x \) be a Th-truth function that satisfies \( \Sigma \), where \( x = (x_1, \ldots, x_N) \). By the above definition of \( \alpha_x \), we have that \( f_x(\alpha_x) = \text{false} \), so that for each \( j \in \{1, \ldots, N\} \) we have that \( p^y_j \notin \alpha_x \) (by Lemma 4.2.18). Also, since \( f_x \) satisfies \( \Sigma \), we have that \( f_x(\sigma_x) = \text{true} \), so that there is a \( k \) in \( \{1, \ldots, N\} \) such that \( p^y_k \in \sigma_x \). Hence \( p^y_k \in \sigma_x \setminus \alpha_x \), so that \( k \in Z \). Hence, by
the definition of $\delta$, we have that $p^k_k \in \delta$. Hence $f_\kappa(\delta) = \text{true}$, so that $\Sigma \models_{\text{Th}} \delta$. Then by strong Th-completeness, $\delta \in D(\Sigma)$.

We now show that $\delta[v, Z] = \Box$. We have from above that for each $j$ in $Z$ and each vector $w$, the propositional atom $p^j_j \notin \sigma_w$. Hence by the definition of $\delta$, we also have for each $j$ in $Z$ that $p^j_j \notin \delta$. Hence by Definition 5.3.1, we have that $\delta[v, Z] = \delta \setminus \{p^j_j \mid j \in Z, y \in A_j\}$, which equals $\Box$.

Since $\delta \in D(\Sigma)$, we have that $\delta[v, Z] \in (D(\Sigma))[v, Z]$. Hence, since $\delta[v, Z] = \Box$, we have that $\Box \in (D(\Sigma))[v, Z]$, as required. \(\Box\)

Although each strongly Th-complete deduction function has the error correction guarantee, the converse does not apply. The following is an example of a deduction function which has the error correction guarantee but which is not strongly Th-complete.

**Example 5.4.3.** The deduction function $F$ has the error correction guarantee (by Corollary 5.2.8) but is not strongly Th-complete. For example, suppose that $N = 2$, that $A_1 = A_2 = \{1, 2, 3\}$, and that $\Sigma = \{p^1_1\}$. Then $F(\Sigma) = \{p^1_1\}$. But $\Sigma \models_{\text{Th}} p^1_1 \lor p^1_2$, and yet $p^1_1 \lor p^1_2 \notin F(\Sigma)$.

We have seen that the error correction guarantee is a strictly weaker property than strong Th-completeness. However, it is not as weak as Th-refutation completeness. Before giving a demonstrating example, we first confirm that the error correction guarantee is at least as strong as refutation Th-completeness.

**Lemma 5.4.4.** If the deduction function $D$ has the error correction guarantee, then it is refutation Th-complete.

**Proof.** The result follows from Proposition 5.3.7, part 2, in which we replace the set $Z$ by the empty set, and note that for any set $\Gamma$ of normal clauses (including $\Gamma = \Sigma$ and $\Gamma = D(\Sigma)$), we have that $\Gamma[v, \emptyset] = \Gamma$. \(\Box\)

Although deduction functions with the error correction guarantee are refutation Th-complete, the converse does not apply. In the next example, we present a deduction function which is refutation Th-complete, but which does not have the error correction guarantee.

**Example 5.4.5.** Given a total ordering $\prec$ on the set $\text{PAtm}$ of propositional atoms (Definition 4.2.3), let the deduction function

$$NR^\prec_{\text{Th}} : \mathcal{P}($$

be defined for each set $\Sigma$ of clauses by

$$NR^\prec_{\text{Th}}(\Sigma) = R^\prec_{\text{Th}}(\Sigma) \cap \text{NC},$$

where $R^\prec_{\text{Th}}$ was defined in Definition 3.2.19.

Then $NR^\prec_{\text{Th}}$ is refutation Th-complete, because (i) $R^\prec_{\text{Th}}$ is refutation Th-complete by Theorem 3.2.30; and (ii) $\Box$ is normal.
However, although the function $\mathcal{NR}_{Th}^\prec$ is refutation Th-complete, it does not have the error correction guarantee, as seen in the situation below.

Define the ordering relation $\prec$ on propositional atoms as follows:

$p^x_i \prec p^y_j$ if $i < j$, or if $i = j$ and $x < y$.

Suppose that $N = 2$, that $A_1 = A_2 = \{1, 2, 3\}$, and that the clauses $\sigma_1, \sigma_2$, the clause set $\Sigma$, and the record $v$ are defined as follows:

$\sigma_1 = p^1_1 \lor p^1_2$,
$\sigma_2 = p^1_1 \lor p^2_2$,
$\Sigma = \{\sigma_1, \sigma_2\}$, and
$\nu = (2, 3)$.

Then the record $\nu$ fails both clauses of $\Sigma$. It cannot be corrected by changing only field 2 since $\sigma_1$ and $\sigma_2$ use different values for field 2. However it can be corrected by changing field 1, to the value 1. Hence

$\mathcal{EL}(\Sigma, \nu) = \{\{1\}, \{1, 2\}\}$.

Then we have the following sets:

$\mathcal{NR}_{Th}^\prec(\Sigma) = \Sigma$
$\mathcal{N}(\mathcal{NR}_{Th}^\prec(\Sigma), \nu) = \Sigma$
$\mathcal{CX}(\mathcal{NR}_{Th}^\prec(\Sigma), \nu) = \{\{1\}, \{2\}, \{1, 2\}\}$.

Hence $\mathcal{NR}_{Th}^\prec$ does not have the error correction guarantee, because the set of covering sets is not a subset of the set of error localisation solutions. That is, $\mathcal{CX}(\mathcal{NR}_{Th}^\prec(\Sigma), \nu) \not\subseteq \mathcal{EL}(\Sigma, \nu)$.

So far in this section, we have examined the relationship between error correction totality and Th-soundness, and the relationship between the error correction guarantee and Th-completeness. Figure 5.1 summarises the results of this section: the left-hand box represents the fact that error correction totality is equivalent to strong Th-soundness, while the right-hand box represents the fact that the error correction guarantee lies between strong Th-completeness and refutation Th-completeness.

The results of this section so far may seem odd because they are not symmetrical, even though all the underlying relationships seem to be symmetrical: firstly, error correction totality and the error correction guarantee are generalisations of soundness and completeness respectively; secondly, error correction totality and the error correction guarantee are converses of each other; and thirdly, soundness and completeness are also converses of each other. Yet error correction totality is related to soundness in a different way from the error correction guarantee to completeness.

The underlying reason for the lack of symmetry is the lack of symmetry in equivalent
§5.4 Soundness, completeness, and covering set correctibility

Figure 5.1a: Error correction totality related to soundness.

Figure 5.1b: Error correction guarantee related to completeness.

Figure 5.1: Diagrammatic representation of the relationship between error correction totality and soundness, and the relationship between the error correction guarantee and completeness. Figure 5.1a, at the top, represents the fact that error correction totality is equivalent to strong Th-soundness. Figure 5.1b, at the bottom, represents the fact that the error correction guarantee lies between strong Th-completeness and refutation Th-completeness.

The deduction function \( D \) is strongly sound if and only if for each Th-truth function \( f \) and each clause set \( \Sigma \) the following holds:

\[
f \text{satisfies } \Sigma \Rightarrow f \text{satisfies } D(\Sigma).
\]  

(5.4.1)

However, as noted in Chapter 3 (page 69), the converse statement, for each Th-truth function \( f \) and each clause set \( \Sigma \), is not equivalent to the strong completeness of \( D \), where the converse statement is

\[
f \text{satisfies } D(\Sigma) \Rightarrow f \text{satisfies } \Sigma.
\]  

(5.4.2)

The first statement, Statement 5.4.1, is used in the proof of the fact that error correction totality is equivalent to strong Th-soundness. However we cannot use the second statement, Statement 5.4.2, to prove a similar relationship between the error correction guarantee and strong Th-completeness.

The lack of symmetry persists when relating error correction totality and the error correction guarantee to Statements 5.4.1 and 5.4.2 respectively. Figure 5.2 expands
strong Th-soundness $\equiv$ error correction totality $\equiv (f \text{ satisfies } \Sigma \Rightarrow f \text{ satisfies } D(\Sigma))$

(Statement 5.4.1)

Figure 5.2a: Expansion of Figure 5.1a to display the fact that Statement 5.4.1 is equivalent to error correction totality.

strong Th-completeness

$\equiv$

error correction guarantee

$\Rightarrow$

refutation Th-completeness

(Statement 5.4.2)

Figure 5.2b: Expansion of Figure 5.1b to display the fact that Statement 5.4.2 is equivalent to the error correction guarantee.

Figure 5.2: Expansion of Figure 5.1 to incorporate the relationship of Statements 5.4.1 and 5.4.2 to error correction totality and the error correction guarantee respectively. Figure 5.2a, at the top, represents the fact that error correction totality is equivalent to Statement 5.4.1. Figure 5.2b, at the bottom, represents the fact that the error correction guarantee only implies Statement 5.4.2, which is distinct from refutation completeness.

Figure 5.1 to represent the fact that, although error correction totality is equivalent to Statement 5.4.1, the error correction guarantee only implies Statement 5.4.2. Indeed the three properties, Statement 5.4.2, the error correction guarantee, and refutation completeness, are distinct from each other. We have already seen, in Example 3.2.28 part 2, that Statement 5.4.2 is distinct from refutation completeness. The remaining relationships are presented below in Proposition 5.4.6 and Example 5.4.7.

**Proposition 5.4.6.** Let $D$ be a deduction function.

1. The deduction function $D$ has error correction totality if and only if for each Th-truth function $f$ and each clause set $\Sigma$,

$$f \text{ satisfies } \Sigma \Rightarrow f \text{ satisfies } D(\Sigma).$$

(5.4.1, repeated)
2. If the deduction function $D$ has the error correction totality, then for each Th-truth function $f$ and each clause set $\Sigma$,

$$f \text{ satisfies } D(\Sigma) \Rightarrow f \text{ satisfies } \Sigma.$$  

(5.4.2, repeated)

Proof.

1. Follows from Lemma 3.2.26 part 3 and Proposition 5.4.1.

2. Suppose that $D$ has the error correction guarantee; that $\Sigma$ is a clause set; and that $f$ is a Th-truth function that satisfies $D(\Sigma)$. We will show that $f$ satisfies $\Sigma$. Firstly, since $f$ is a Th-truth function, it can be written $f_v$ for some record $v$. Since $f_v$ satisfies $D(\Sigma)$, the empty set yields a correction to $f_v$ with respect to the clause set $D(\Sigma)$. Hence by Theorem 5.3.5, with $C = \emptyset$, we have that $(D(\Sigma))[v, \emptyset]$ is Th-satisfiable. Hence since $\Box$ is not Th-satisfiable, we have that $\Box \not\in (D(\Sigma))[v, \emptyset]$, from which we can deduce, using Proposition 5.3.7 part 2, that $\Sigma[v, \emptyset]$ is Th-satisfiable. Once again using Theorem 5.3.5 with $C = \emptyset$, we can deduce that the empty set yields a correction to the Th-truth function $f_v$ with respect to the clause set $\Sigma$, that is $f_v$ satisfies $\Sigma$, as required.

The above proposition tells us that error correction totality is equivalent to Statement 5.4.1, and that the error correction guarantee implies Statement 5.4.2. The next example demonstrates that the error correction guarantee is indeed distinct from Statement 5.4.2.

Example 5.4.7. This is an example of a deduction function for which Statement 5.4.2 is distinct from the error correction guarantee and also from refutation completeness. The identity deduction function $I$ defined for each clause set $\Sigma$ by

$$I(\Sigma) = \Sigma$$

satisfies Statement 5.4.2, but does not have the error correction guarantee, nor is it refutation complete.

We have established in this section the parallels between the properties of covering set correctibility and of soundness / completeness and displayed them in Figure 5.2. The next step is to return to the original problem of error localisation and its parallel with the propositional satisfiability problem.

5.5 Error localisation and propositional satisfiability

As discussed in Section 3.2.6 of Chapter 3, the propositional satisfiability problem (known as SAT) is the problem of deciding whether a given set $\Sigma$ of clauses is satisfiable. The error localisation problem is an extension of SAT: it is a satisfiability problem constrained by the given field set $C$ and the given record $v$. In this section, we will see
that not only is the error localisation problem an extension of the SAT problem, but the methods of solution are in parallel, and the reasons why the pure deduction method works for error localisation is an extension of the reasons why the pure deduction method works for SAT.

Table 5.1 summarises the parallels between error localisation and propositional satisfiability, each of which is allocated a separate column. The table lists several aspects for each of the two problems, and in each case the description for the error localisation problem is a strengthening of the description for the propositional satisfiability problem. We will consider each aspect in turn in the following discussion. The first aspect is the problem statement itself: as stated above, the error localisation problem is a strengthening of the SAT problem.

For both the error localisation problem and the SAT problem, most solution methods can be seen as a trade-off between search and deduction, as discussed in Chapter 2, Section 2.7, and Chapter 3, Section 3.2.6. In particular, as summarised in aspect 2 of Table 5.1, both problems can be solved by a pure deduction method, with no search component. For example, as discussed in Chapter 3, SAT can be solved using a suitable deduction function such as resolution, as seen in the work of Davis and Putnam (1960), and Rish and Dechter (Rish and Dechter 2000; Dechter and Rish 1994). On the other hand, the error localisation problem can also be solved using a suitable deduction function such as the Fellegi-Holt deduction function $F$.

Indeed, the two pure deduction methods use deduction in a similar way. For a suitable deduction function $D$, the set $\Sigma$ is satisfiable if and only if $\Box \notin D(\Sigma)$. On the other hand, the set $\Sigma$ has an error localisation solution for the record $v$ and the field set $C$ if and only if $\Box \notin (D(\Sigma))[v, C]$.

Not only can both problems be solved in a similar way using a suitable deduction function $D$, but $D$ is suitable in similar circumstances. On the one hand, SAT can be solved using $D$ if and only if $D$ is both refutation sound and refutation complete. For example, the resolution deduction function is both refutation sound and refutation complete. On the other hand, the error localisation problem can be solved using $D$ if and only if $D$ has both error correction totality and the error correction guarantee, which are strengthenings of refutation soundness and refutation completeness respectively. For example, the Fellegi-Holt deduction function has both error correction totality and the error correction guarantee.

Not only are the properties for error localisation strengthenings of the properties for SAT, but the two pairs of properties play the same roles with regard to the two problems: that is, refutation completeness plays the same role for the SAT problem as the error correction guarantee plays for the error localisation problem; and refutation soundness plays the same role for the SAT problem as error correction totality plays for the error localisation problem. In more detail, we first note that both problems have been stated as “yes / no” questions: the SAT problem as “is $\Sigma$ satisfiable?” and the error localisation problem as “does the field set $C$ yield a correction to $v$ with respect to $\Sigma$?”. We summarise the situation for the answer “yes” in aspect 3 of Table 5.1. On the one hand, for SAT, the answer “yes” from the deduction method is always correct if and only if the deduction function is refutation complete. On the other hand, for error
### Table 5.1: Comparison of error localisation and propositional satisfiability

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Error localisation</th>
<th>Propositional satisfiability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Problem statement</td>
<td>Given a set ( \Sigma ) of normal clauses, a record ( v ) and a set ( C ) of fields, is ( \Sigma[v, C] ) Th-satisfiable?</td>
<td>Given a set ( \Sigma ) of clauses, is ( \Sigma ) satisfiable?</td>
</tr>
<tr>
<td>2. Pure deduction solution method, using deduction function ( D )</td>
<td>Return “yes” if and only if ( \Box \not\in (D(\Sigma))[v, C] ).</td>
<td>Return “yes” if and only if ( \Box \not\in D(\Sigma) ).</td>
</tr>
<tr>
<td>3. The property that the output “yes” is always correct</td>
<td>Error correction guarantee: ( \Box \not\in (D(\Sigma))[v, C] ) ( \Rightarrow ) ( \Sigma[v, C] ) is Th-satisfiable</td>
<td>Refutation completeness: ( \Box \not\in D(\Sigma) ) ( \Rightarrow ) ( \Sigma ) is satisfiable</td>
</tr>
<tr>
<td>4. The property that the output “no” is always correct</td>
<td>Error correction totality: ( \Sigma[v, C] ) is Th-satisfiable ( \Rightarrow \Box \not\in (D(\Sigma))[v, C] )</td>
<td>Refutation soundness: ( \Sigma ) is satisfiable ( \Rightarrow \Box \not\in D(\Sigma) )</td>
</tr>
<tr>
<td>5. Examples of deduction function ( D )</td>
<td>Fellegi-Holt deduction ( F )</td>
<td>Th-resolution ( R_{Th} )</td>
</tr>
<tr>
<td></td>
<td>Field Code Forest ( FC_{\omega} )</td>
<td>Ordered Th-resolution ( R_{Th}^\omega )</td>
</tr>
</tbody>
</table>
localisation, the answer “yes” is always correct if and only if the deduction function has the error correction guarantee. There is a similar parallel when the answer is “no”, as summarised in aspect 4 of Table 5.1. For SAT the answer “no” is always correct if and only if the deduction function is refutation sound; for error localisation the answer “no” is always correct if and only if the deduction function has error correction totality.

Given the above parallels, it is not surprising that there are parallels between the deduction functions for which the pure deduction methods work. We give two pairs of functions in aspect 5 of Table 5.1. The first pair consists of the Fellegi-Holt deduction function $F$ and the Th-resolution deduction function $R_{\text{Th}}$: we proved in Chapter 4 that $F$ is essentially the same as $R_{\text{Th}}$. The second pair of parallel deduction functions consists of the field code forest deduc tion function $FCF_{\omega}$ and the ordered Th-resolution function $R_{\text{Th}}^{\prec}$. The function $FCF_{\omega}$ builds on $F$ in the same way as $R_{\text{Th}}^{\prec}$ builds on $R_{\text{Th}}$: in both cases deduction steps are only allowed on the lowest order propositional atoms of the input clauses.

Thus the error localisation problem and the SAT problem are strongly related, in the problem statement, pure deduction solution method, and the reasons that the pure deduction method works. This means that the automated techniques of SAT could potentially be adapted to solve the error localisation problem.

5.6 Conclusion

This chapter has presented the beginnings of a theoretical logical framework for analysing methods of solving the error localisation problem. The main aspects are listed below.

1. The error localisation problem is a strengthening of the propositional satisfiability (SAT) problem.

2. Both the error localisation problem and the propositional satisfiability problem can be solved by a mixture of deduction and search, and indeed both problems can be solved by a pure deduction method.

3. The pure deduction method for the error localisation problem, that is the covering set method, is a strengthening of the pure deduction method for the SAT problem. Each method seeks to determine whether the empty clause is obtained in a calculated set, but for the error localisation problem the calculated set is more complex to calculate.

4. The pure deduction method of solving the error localisation problem depends on covering set correctibility in just the same way as the pure deduction method of solving the SAT problem depends on refutation completeness and soundness.

In order to obtain the above results, we first formalised covering set correctibility in terms of logic by directly translating from sets to logic the key definitions such as “involved” and “covering set”. We ensured that the direct translation was consistent with sets, so that the formalisations of functions that have covering set correctibility in sets also have covering set correctibility in logic. Any method of solving the error
localisation problem based on this formalisation would also be a direct translation from the methods used in sets.

We then analysed the meaning of covering set correctibility and sought some constructions of logic that have the same meaning. Covering set correctibility links together two other properties, namely that of yielding a solution and that of being a covering set of a set of failed clauses. Both properties can be defined in terms of sets of “reduced clauses”, obtained by fixing the truth values of certain propositional atoms. When expressed in terms of reduced clauses, covering set correctibility can be seen to be a strengthening of refutation completeness and soundness. The analysis of covering set correctness in terms of logical constructs might help in converting the methods of SAT to error localisation.

Logic gives three benefits. Firstly, it gives an alternative way of analysing the problem and thus potentially gives new theoretical insights. Secondly, its collection of sophisticated automated tools could potentially be modified to use covering set correctness for solving error localisation problems. Thirdly, the strong parallels between the two subjects are of aesthetic appeal.
Covering set correctibility from the perspective of logic: categorical edits
The Field Code Forest Algorithm

6.1 Introduction

The Field Code Forest (FCF) Algorithm, developed by Garfinkel, Kunnathur, and Liepins (1986b), was stated in Chapter 2 (Section 2.6). Given a total ordering \( \omega \) of the fields, the algorithm specifies an edit generation function \( \text{FCF}_\omega \).

This chapter gives a proof that the FCF Algorithm is correct, meaning that the function \( \text{FCF}_\omega \) is covering set correctible, or equivalently that its logical formalisation \( \mathcal{FCF}_\omega \) is covering set correctible. The proof can be written equally in terms of edits (using the function \( \text{FCF}_\omega \)) or in terms of logical clauses (using \( \mathcal{FCF}_\omega \)). The presented proof is in terms of edits in order to stay consistent with other published works. For completeness, I give a statement of the function \( \mathcal{FCF}_\omega \) in terms of clauses in Appendix 1 to this chapter.

Garfinkel, Kunnathur and Liepins (1986b) - henceforth referred to as “G, K & L” - gave a proof that the function \( \text{FCF}_\omega \) is what they called “sufficient”, to be defined in Section 6.2. Sufficiency is closely related to covering set correctibility, but they are not the same, as explained in Section 6.2.

There have been questions about whether the FCF Algorithm is correct (Winkler 1997). Indeed, in Section 6.2 we will see that the proof of sufficiency given by G, K & L proves something other than the correctness of the FCF Algorithm.

This chapter is the result of my attempt to decide whether the FCF Algorithm is correct. At first I expected that the correctness could be proved by simply modifying G, K & L’s proof. However, my attempt at such a modification failed, and I sought a different approach. While the proof of error correction totality of \( \text{FCF}_\omega \) is brief (Lemma 6.3.1), the proof of the error correction guarantee of \( \text{FCF}_\omega \) takes many sections (Sections 6.4 to 6.8, with an overview in Section 6.3).

The proof of the error correction guarantee of \( \text{FCF}_\omega \) is a modification of Fellegi and Holt’s original proof (Fellegi and Holt 1973, 1976), presented in this thesis as Theorem 2.4.6, of the error correction guarantee of the function FH. As in the F & H proof, the proof for \( \text{FCF}_\omega \) uses three steps based on the lifting property. However there are some important differences.

The main difference between the proof presented here and the F & H proof is that the required lifting property is not for a single sequence of sets but for a tree of sets,
each branch of which has the lifting property. The definition of the tree of sets is in Section 6.5 (after an outline of why the sets used in the original F & H proof do not work) and its main property is in Section 6.6.

The proof presented here for the error correction guarantee of FCF\(_{\omega}\) uses a simplification of the function FCF\(_{\omega}\), called FCFS\(_{\omega}\) ("S" for "simple"), which is the same as FCF\(_{\omega}\) except without the Domination Rules, and is defined precisely in Section 6.4. After proving that the simpler function FCFS\(_{\omega}\) has the error correction guarantee (Sections 6.4 to 6.7), we use the relationship between FCFS\(_{\omega}\) and FCF\(_{\omega}\) to get the error correction guarantee of FCF\(_{\omega}\) (Section 6.8).

Once we have proved that FCF\(_{\omega}\) has both the error correction guarantee and error correction totality, we can quickly deduce that the FCF Algorithm is correct (Section 6.9).

After an explanation about notation, we start with the original work done by Garfinkel, Kunnathur and Liepins.

**Notation.** This chapter uses a large number of variables, exceeding the capacity of the Roman alphabet. Therefore we will expropriate certain Greek letters even though they have been used for other purposes in other chapters:

For edit sets, other than the Roman letter \(S\) and the bilingual letters \(X\) and \(E\), we will use the Greek letters \(\Gamma\), \(\Delta\), \(\Lambda\), and \(\Omega\) (the last symbol following Fellegi and Holt).

For edits, other than the Roman letter \(e\), we will use \(\alpha\), \(\beta\), \(\gamma\), \(\delta\), \(\epsilon\), and \(\xi\).

For nodes of a field code forest, we will use \(\nu\), \(\rho\), \(\sigma\), \(\tau\), and \(\omega\) (the last of which is used as a total ordering of all the fields).

### 6.2 The work of Garfinkel, Kunnathur and Liepins

Garfinkel, Kunnathur, and Liepins (1986b) gave a proof that the edit generation function determined by the FCF Algorithm is what they call "sufficient" - to be defined below in Section 6.2.1. The property of sufficiency is different from covering set correctibility, although the two are related. This section explains the difference between sufficiency and covering set correctibility (Section 6.2.1), and then outlines and reviews G, K & L’s proof of the sufficiency of the function FCF\(_{\omega}\), for \(\omega\) a total ordering of the fields (Section 6.2.2).

#### 6.2.1 Sufficiency

In this section we define sufficiency and explain its relationship to covering set correctibility. Sufficiency was defined by G, K & L as follows.

**Definition 6.2.1.** (Garfinkel, Kunnathur, and Liepins 1986b.) If \(b = (b_1, \ldots, b_N)\) is a sequence of positive numbers and \(G\) is an edit generation function, then \(G\) is **sufficient** for \(b\) when
(i) $G$ is a subfunction of $\text{ENFH}$ (where subfunction is defined in Definition 1.9.3 of Chapter 1, and $\text{ENFH}$ is defined in Definition 2.5.2 of Chapter 2);

(ii) for each edit set $E$,

$$\bigcup\{e \mid e \in G(E)\} = \bigcup\{e \mid e \in E\};$$

and

(iii) $G$ is smallest weighted covering set correctible for the weights $b$, that is for each edit set $E$ and record $v$,

$$\text{SCX}(G(E), v, b) = \text{SEL}(E, v, b).$$

The above definition uses slightly different language from the original used by G, K & L, as described below.

1. G, K & L applied the term “sufficient” to the set $G(E)$, rather than to the edit generation function $G$.

2. G, K & L used the expression “sufficient” rather than “sufficient for the weights $b$”. However, the weights in their paper are fixed, hence it appears that they intended that sufficiency be relative to one particular set of weights, as in the above definition.

3. G, K & L used the notation $E_S$ and $E_C$ instead of $G(E)$ and $\text{ENFH}(E)$ respectively.

4. G, K & L used the expression “$E_S$ is equivalent to $E_C$” to encompass conditions (ii) and (iii) of the above definition.

5. Instead of condition (ii) of the above definition, G, K & L used the equation

$$\bigcup\{e \mid e \in G(E)\} = \bigcup\{e \mid e \in \text{ENFH}(E)\},$$

which they wrote, using a notation “$P$”, as $P(E_S) = P(E_C)$. By Definition 2.5.2 of $\text{ENFH}$, this equation means the same as condition (ii).

6. G, K & L expressed condition (iii) of the above definition directly in terms of the covering set problem, that is:

the smallest weighted covering sets of the edits of $E_S$ ($= G(E)$) failed by $v$ are the same as the smallest weighted covering sets of the edits of $E_C$ ($= \text{ENFH}(E)$) failed by $v$,

which is the same as the following equation:

$$\text{SCX}(G(E), v, b) = \text{SCX}(\text{ENFH}(E), v, b).$$
Since ENFH is covering set correctible (Corollary 2.5.9 and Proposition 2.5.11) this last equation means the same as condition (iii).

Since condition (iii) of the definition of sufficiency is about covering set correctibility, it follows that sufficiency and covering set correctibility are closely related. Certainly, by Proposition 2.3.20, the edit generation function $G$ is covering set correctible if and only if it satisfies condition (iii) for all positive weights. Hence, if $G$ is sufficient for all positive weights, then $G$ is covering set correctible.

However, the converse does not apply. If $G$ is covering set correctible then a fortiori it satisfies condition (iii) of sufficiency, and it satisfies condition (ii) by Proposition 6.2.2 below, but it need not satisfy condition (i), as seen in the subsequent example (Example 6.2.3).

**Proposition 6.2.2.** If the edit generation function $G$ is covering set correctible, then it satisfies condition (ii) of Definition 6.2.1 of sufficiency, that is, for each edit set $E$,

$$
\bigcup \{ e \mid e \in G(E) \} = \bigcup \{ e \mid e \in E \}.
$$

**Proof.** We have already mostly proved this proposition for the logical formalisation, as Proposition 5.4.6. Therefore, in spite of doing everything else in this chapter in terms of edits rather than clauses, we use the logical formalisation in this proof.

Suppose that $G$ is a covering set correctible edit generation function. Then by Lemma 5.2.7 part 7 and since the function $nc$ is bijective, the deduction function $nc \circ G \circ nc^{-1}$ is also covering set correctible. Then by Proposition 5.4.6, we have for each Th-truth function $f$ and each set $\Sigma$ of normal clauses that

$$
f \text{ satisfies } nc \circ G \circ nc^{-1}(\Sigma) \Leftrightarrow f \text{ satisfies } \Sigma.
$$

But since $f$ is a Th-truth function, it can be written $f_v$ for some record $v$ (Lemma 4.2.17 part 1), and hence for each record $v$ and each set $\Sigma$ of normal clauses we have that

$$
f_v \text{ satisfies } nc \circ G \circ nc^{-1}(\Sigma) \Leftrightarrow f_v \text{ satisfies } \Sigma.
$$

Then by Proposition 4.2.28 and the fact that $nc$ is a bijection, we have for each record $v$ in each set $\Sigma$ of normal clauses that

$$
v \text{ does not satisfy } G \circ nc^{-1}(\Sigma) \Leftrightarrow v \text{ does not satisfy } nc^{-1}(\Sigma),
$$

that is, for each set $\Sigma$ of normal clauses,

$$
\bigcup \{ e \mid e \in G \circ nc^{-1}(\Sigma) \} = \bigcup \{ e \mid e \in nc^{-1}(\Sigma) \}.
$$

which, since $nc$ is a bijection, is equivalent to saying that for each normal edit set $E$,

$$
\bigcup \{ e \mid e \in G(E) \} = \bigcup \{ e \mid e \in E \},
$$

as required.
Thus, all covering set correctible functions satisfy condition (ii) of Definition 6.2.1 for sufficiency. However there are covering set correctible functions that do not satisfy condition (i), that is, they are not subfunctions of ENFH. The following is an example of such a function.

**Example 6.2.3.** This is an example of an edit generation function $G$ that is covering set correctible but is not a subfunction of ENFH. Define $G(E)$ for each edit set $E$ by

$$G(E) = \text{ENFH}(E) \cup \left\{ \bigcap \{ e \mid e \in X \} \mid X \subseteq E \right\}.$$ 

Then $G$ is covering set correctible because (1) since $G$ is a superfunction (Definition 1.9.3) of ENFH which has the error correction guarantee, then by Corollary 2.5.10 the function $G$ also has the error correction guarantee, and (2) since ENFH, which has error correction totality, is superior (in the sense of Definition 1.9.4) to $G$, then by Corollary 2.5.15 the function $G$ also has error correction totality.

But $G(E)$ need not be a subset of $\text{ENFH}(E)$. For example, suppose that $N = 2$, that $A_1 = A_2 = \{1, 2, 3, 4\}$, and that $E = \{e_1, e_2\}$ where

$$e_1 = \{1, 2\} \times \{4\}; \text{ and } e_2 = \{2, 3\} \times \{4\}.$$

Then

$$\text{ENFH}(E) = E; \text{ and } \quad G(E) = E \cup \{\{2\} \times \{4\}\}.$$ 

Thus $G(E) \not\subseteq \text{ENFH}(E)$ and $G$ is not a subfunction of ENFH.

The above proposition and example tell us that, although covering set correctible functions need not satisfy condition (i) of sufficiency, they do satisfy conditions (ii) and (iii). Hence if a covering set correctible function is also a subfunction of ENFH, then it is sufficient for all positive weights.

The above discussion of the relationship between covering set correctibility and sufficiency for all positive weights is summarised in the next proposition.

**Proposition 6.2.4.**

1. If $G$ is a subfunction of ENFH and $G$ is covering set correctible, then $G$ is sufficient for all positive weights $b$.

2. If $G$ is sufficient for all positive weights $b$, then $G$ is covering set correctible.

**Proof.**

1. Condition (i) of Definition 6.2.1 of sufficiency is given by assumption; condition (ii) follows from Proposition 6.2.2; and condition (iii) follows from Proposition 2.3.20.
2. By Proposition 2.3.20, the covering set correctibility of $G$ follows from condition (iii) of Definition 6.2.1 of sufficiency.

The above discussion relates covering set correctibility to the property of sufficiency for all positive weights. If $G$ has the weaker property of sufficiency for just some sets of positive weights, then it need not be covering set correctible, as seen in the next example.

**Example 6.2.5.** The function $G$ of Example 2.5.16 of Chapter 2 (and repeated below) is sufficient for a given set of weights but not covering set correctible. Suppose that $N = 2$, that $A_1 = \{1, 2, 3, 4\}$, that $A_2 = \{1, 2\}$, that the vector of weights is $(1, 2)$, that the edit set $X = \{\{2, 3\} \times \{1\}, \{3, 4\} \times \{2\}\}$, and that $G(E)$ is defined for each edit set $E$ by

$$G(E) = \begin{cases} \text{ENFH}(E), & \text{if } E \neq X \\ E, & \text{if } E = X. \end{cases}$$

We confirm that the function $G$ is sufficient for the weights $(1, 2)$ by checking each of the conditions (i), (ii) and (iii) of Definition 6.2.1 for sufficiency for $(1, 2)$:

(i) For each edit set $E$, the set $G(E) \subseteq \text{ENFH}(E)$, because $E \subseteq \text{ENFH}(E)$.

(ii) For each edit set $E$, the set $\bigcup \{e \mid e \in G(E)\} = \bigcup \{e \mid e \in E\}$, by the definition of ENFH.

(iii) As explained in Example 2.5.16 of Chapter 2, the function $G$ is smallest weighted covering set correctible for the weights $(1, 2)$.

Having confirmed that $G$ is sufficient for the weights $(1, 2)$, we note that in Example 2.5.16 of Chapter 2 we showed that $G$ is not covering set correctible.

Thus $G$ is sufficient for $(1, 2)$, but is not covering set correctible.

In this section we have seen that sufficiency and covering set correctibility are closely related, although they are not the same. Having defined sufficiency, we are ready to inspect the proof published by G, K & L about the sufficiency of the FCF Algorithm.

### 6.2.2 Proof by G, K & L of the sufficiency of $\text{FCF}_\omega$

In this section we analyse the proof of the sufficiency of the function $\text{FCF}_\omega$ published by G, K & L. Their result is about something other than the correctness of the FCF Algorithm.

The proof by G, K & L of sufficiency is not in terms of the function $\text{FCF}_\omega$, but in terms of two other functions, to be called $GKL$ and $MGKL$. G, K & L themselves used a different notation: for a given edit set $E$ they used $E_\sigma$ instead of $GKL(\sigma, E)$, and they used $N_\sigma$ instead of $MGKL(\sigma, E)$.

A precise definition of the function $GKL$ is given below in Definition 6.2.6. The function $GKL$ is the same as GenI (Definition 2.6.4), but without the restrictions on involved fields. The function $MGKL$ returns the maximal elements of those returned by $GKL$. 
Definition 6.2.6. Let $E$ be an edit set, and let $\omega$ be a total ordering of the fields. Let $0 \leq n \leq N$ and suppose that $(i_1, \ldots, i_n) \subseteq \omega$. Define the set $GKL((i_1, \ldots, i_m), E)$ inductively, for $m = 0, \ldots, n$, as follows:

1. $GKL((), E) = E$.

2. If $m \geq 1$, then

$$GKL((i_1, \ldots, i_m), E) = \left\{ \text{FHG}(i_m, X) \mid X \subseteq GKL((i_1, \ldots, i_{m-1}), E), \right.$$ \[and \ \text{FHG}(i_m, X) \neq \emptyset \}.$$ 

Define

$$MGKL((i_1, \ldots, i_m), E) = \text{Max} \circ GKL((i_1, \ldots, i_m), E).$$

A consequence of the definition of $GKL$ is that the set of edits associated with any node contains all the non-empty edits at previous nodes, that is, if $(i_1, \ldots, i_n) \subseteq \omega$ and $1 \leq m \leq n$, then

$$GKL((i_1, \ldots, i_m), E) \supseteq GKL((i_1, \ldots, i_{m-1}), E) \setminus \{\emptyset\}.$$ 

Because of an ambiguity in G, K & L’s paper, the set $GKL((i_1, \ldots, i_m), E)$ could be taken to be only the essentially new edits generated, using generating field $i_m$, from subsets of $GKL((i_1, \ldots, i_{m-1}), E)$. But then $MGKL(\omega, E)$ is either empty or $\{A_1 \times \cdots \times A_N\}$, and is not sufficient for any set of positive weights.

The proof presented by G, K & L gives us that if $\omega$ is an ordering of the fields then $MGKL(\omega, -)$ satisfies conditions (ii) and (iii) of Definition 6.2.1 for sufficiency, with the consequence that $MGKL(\omega, -)$ is covering set correctible. The proof uses three steps, as follows.

1. Theorem 1 of G, K & L. If $i$ and $j$ are distinct fields, then

$$MGKL((i, j), E) = MGKL((j, i), E),$$

which G, K & L wrote as $N_{ij} = N_{ji}$.

The proof uses the distributive laws of set union and intersection. Note that $(i, j)$ and $(j, i)$ are substrings of different field orderings.

2. Corollary 1 of G, K & L. If $\sigma$ is a total ordering of a subset of the fields, and $\delta(\sigma)$ is a permutation of $\sigma$, then

$$MGKL(\delta(\sigma), E) = MGKL(\sigma, E).$$

G, K & L wrote this equation using the notation $(i, j, \ldots, k)$ instead of $\sigma$, and using a second permutation $\gamma(\sigma)$ of $\sigma$, as $N_{\delta(i,j,\ldots,k)} = N_{\gamma(i,j,\ldots,k)}$.

The proof uses Theorem 1 of G, K & L and the fact that any permutation of the fields can be written as a sequence of transpositions of pairs of fields. It also
uses the compatibility of the MGKL function with the concatenation of fields into field orderings; that is, if $\sigma_1$ and $\sigma_2$ are orderings of disjoint subsets of the fields, then $\text{MGKL}(\sigma_1 \circ \sigma_2, E) = \text{MGKL}(\sigma_2, \text{MGKL}(\sigma_1, E))$.

3. Lemma 2 of G, K & L states that if $\omega$ is a total ordering of the fields, then the function $\text{MGKL}(\omega, -)$ is sufficient. However $\text{MGKL}(\omega, -)$ is not a subfunction of $\text{ENFH}$, so it does not satisfy condition (i) of Definition 6.2.1 for sufficiency. However, by G, K & L Corollary 1, $\text{MGKL}(\omega, E) = \text{MFH}(E)$, so that $\text{MGKL}(\omega, -)$ is covering set correctible. Hence by Propositions 2.3.20 and 6.2.2, the function $\text{MGKL}(\omega, -)$ satisfies conditions (ii) and (iii) of Definition 6.2.1 for sufficiency.

Note that although the function $\text{MGKL}(\omega, -)$ is not sufficient, it is covering set correctible. The reason is that, by G, K & L Corollary 1, the function $\text{MGKL}(\omega, -)$ equals the function $\text{MFH}$.

The advantage of $\text{MGKL}(\omega, -)$ over the function $\text{MFH}$ is that, by G, K & L Corollary 1, the function $\text{MGKL}(\omega, -)$ can be calculated by traversing the fields just once. For example, if there are only two fields, then $\text{MGKL}((1, 2), E)$ is superior (in the sense of Definition 1.9.4) to the set of edits generated from $E$ using just the field 2. Thus, when deciding whether a record can be corrected by changing only field 2, those edits where no field or field 2 has been used as the generating field are included in $\text{MGKL}(\omega, E)$.

The paper of G, K & L presents a puzzle: although they have just presented a function $\text{MGKL}(\omega, -)$ that is calculated by traversing the single branch ending with the node $\omega$, they then propose an algorithm that traverses a field code forest rather than a single branch. However, it would appear that the reason that they propose traversing a field code forest is that the FCF Algorithm uses a different edit generation function, $\text{GenI}$, which is much more restricted than the edit generation function $\text{MGKL}$ used in their proof. For example, in the FCF Algorithm, each newly generated edit must have its generating field uninvolved, and the edits used to generate any new edit also have restrictions on the involvedness of their fields.

It seems that G, K & L wished to imply that their proofs of their Theorem 1, Corollary 1 and Lemma 2 can be easily converted to the method of edit generation of the FCF Algorithm. In such a conversion, one replaces $\text{MGKL}(\sigma, E)$ of Corollary 1 by $\bigcup \{ \text{GenF}(\tau, E) \mid \tau \subseteq \sigma \}$, which equals $\text{FCF}_\omega(E)$ when $\sigma$ is a total ordering $\omega$ of all the fields.

The conversion to a proof for the FCF Algorithm fails at two points. Firstly, Corollary 1 of G, K & L does not convert, and indeed there are field orderings $\omega$ and permutations $\delta$ such that $\text{FCF}_\omega(E)$ is not the same as $\text{FCF}_{\delta(\omega)}(E)$. Secondly, even if $\text{FCF}_\omega(E)$ and $\text{FCF}_{\delta(\omega)}(E)$ had some useful relationship, the method of Lemma 2 (converted to the FCF Algorithm) does not apply because $\text{FCF}_\omega(E)$ is not in general equal to $\text{MFH}(E)$. The next example is a simple case where $\text{FCF}_\omega(E) \neq \text{FCF}_{\delta(\omega)}(E)$ and $\text{FCF}_\omega(E) \neq \text{MFH}(E)$.
Example 6.2.7. Let $A_1 = \{1, 2\}$ and $A_2 = \{1, 2, 3\}$. Let $E = \{e_1, e_2, e_3\}$ where

\[
\begin{align*}
    e_1 &= \{1\} \times \{1, 2\}; \\
    e_2 &= \{2\} \times \{1\}; \\
    e_3 &= \{2\} \times \{2, 3\}.
\end{align*}
\]

We will construct $\text{FCF}_{(1, 2)}(E)$ and $\text{FCF}_{(2, 1)}(E)$ and find that they are different. In addition $\text{FCF}_{(1, 2)}(E) \neq \text{MFH}(E)$.

Firstly, the construction of $\text{FCF}_{(1, 2)}(E)$ can be summarised in the next table, where the edits $e_4$, $e_5$, $e_6$ are defined after the table.

<table>
<thead>
<tr>
<th>$\sigma = \emptyset$</th>
<th>(1)</th>
<th>(2)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{GenI}(\sigma, E)$</td>
<td>${e_1, e_2, e_3}$</td>
<td>${e_4, e_5}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$\text{GenV}(\sigma, E, (1))$</td>
<td>${e_1, e_2, e_3}$</td>
<td>${e_4, e_5}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$\text{GenV}(\sigma, E, (2))$</td>
<td>${e_1, e_4, e_6}$</td>
<td>${e_4, e_5}$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

$\text{GenV}(\sigma, E, (2)) = \text{GenF}(\sigma, E)$

$e_4 = \text{FHG}(1, \{e_1, e_2\}) = A_1 \times \{1\}$

$e_5 = \text{FHG}(1, \{e_1, e_3\}) = A_1 \times \{2\}$

$e_6 = \text{FHG}(2, \{e_4, e_3\}) = \{2\} \times A_2$

$e_4 \supseteq e_2$

$e_6 \supseteq e_3$

Hence $\text{FCF}_{(1, 2)}(E) = \text{GenF}(\emptyset, E) \cup \text{GenF}((1), E) \cup \text{GenF}((1, 2), E) \cup \text{GenF}((2), E) = \{e_1, e_4, e_5, e_6\}$.

Secondly, the construction of $\text{FCF}_{(2, 1)}(E)$ can be summarised in the next table, where the edits $e_6$ and $e_7$ are defined after the table. Note that the edit $e_6$ was already defined above, but is generated here from different edits.

<table>
<thead>
<tr>
<th>$\sigma = \emptyset$</th>
<th>(1)</th>
<th>(2)</th>
<th>(2, 1)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{GenI}(\sigma, E)$</td>
<td>${e_1, e_2, e_3}$</td>
<td>${e_6}$</td>
<td>$\emptyset$</td>
<td>${e_7}$</td>
</tr>
<tr>
<td>$\text{GenV}(\sigma, E, (1))$</td>
<td>${e_1, e_2, e_3}$</td>
<td>${e_6}$</td>
<td>$\emptyset$</td>
<td>${e_7}$</td>
</tr>
<tr>
<td>$\text{GenV}(\sigma, E, (2))$</td>
<td>${e_1, e_6}$</td>
<td>${e_6}$</td>
<td>$\emptyset$</td>
<td>${e_7}$</td>
</tr>
<tr>
<td>$\text{GenV}(\sigma, E, (1))$</td>
<td>${e_7, e_6}$</td>
<td>${e_6}$</td>
<td>$\emptyset$</td>
<td>${e_7}$</td>
</tr>
<tr>
<td>$\text{GenV}(\sigma, E, (1))$</td>
<td>${e_7, e_6}$</td>
<td>${e_6}$</td>
<td>$\emptyset$</td>
<td>${e_7}$</td>
</tr>
</tbody>
</table>

$\text{GenV}(\sigma, E, (1)) = \text{GenF}(\sigma, E)$

$e_6 = \text{FHG}(2, \{e_2, e_3\}) = \{2\} \times A_2$

$e_7 = \text{FHG}(1, \{e_1, e_6\}) = A_1 \times \{1, 2\}$

$e_6 \supseteq e_2, e_6 \supseteq e_3$

$e_7 \supseteq e_1$
Hence
\[ FCF_{(2,1)}(E) = \text{GenF}((), E) \cup \text{GenF}((2), E) \cup \text{GenF}((2, 1), E) \cup \text{GenF}((1), E) \]
\[ = \{e_6, e_7\}. \]

Also,
\[ FH(E) = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}, \text{ and} \]
\[ MFH(E) = \{e_6, e_7\}. \]

Hence \( FCF_{(1,2)}(E) \neq FCF_{(2,1)}(E) \) and \( FCF_{(1,2)}(E) \neq MFH(E) \).

Winkler (1997) gives a different example to demonstrate that \( FCF_\omega \neq FCF_\delta(\omega) \)· However he appears to use a different interpretation of the algorithm from that given here. Appendix 2 to this chapter states his counter-example and explains the difference between his interpretation of the algorithm and the interpretation presented in this thesis in Section 2.6.

The above example shows that Corollary 1 and Lemma 2 of G, K & L cannot be simply converted to prove that \( FCF_\omega \) is covering set correctible. Although G, K & L proved that the function \( MGKL(\omega, -) \) is covering set correctible, the proof cannot be simply adapted to prove that \( FCF_\omega \) is covering set correctible.

However, \( FCF_\omega \) is indeed covering set correctible. Its covering set correctibility is the subject of the rest of this chapter. Interestingly, the proof of the covering set correctibility of \( FCF_\omega \) does not use equivalents of G, K & L’s Theorem 1 and Corollary 1. That is, there is no need to prove that, if one permutes the order of the fields, one still obtains the same set of generated edits. Instead the proof is a generalisation of the method of proof of Fellegi and Holt.

### 6.3 Overview of a correctness proof of the FCF Algorithm

This section gives an overview of a proof of the correctness of the FCF Algorithm, that is, that for any total ordering \( \omega \) of the fields, the function \( FCF_\omega \) is covering set correctible. The full proof occupies the next several sections (Sections 6.4 to 6.9). The end result, that the FCF Algorithm is correct, is given in Section 6.9 as Proposition 6.9.1.

There are two components to the proof of the covering set correctibility of \( FCF_\omega \): error correction totality and error correction guarantee. We will quickly dispense with error correction totality below in Lemma 6.3.1. In contrast, the proof of the error correction guarantee will occupy the next several sections but will be introduced in this section.

The error correction totality of \( FCF_\omega \) is a consequence of our earlier result (Corollary 2.5.8) that any subfunction (Definition 1.9.3) of a function with error correction totality also has error correction totality:

**Lemma 6.3.1 (Error correction totality of \( FCF_\omega \)).** If \( \omega \) is a total ordering of the fields then the function \( FCF_\omega \) has error correction totality.

**Proof.** The function \( FCF_\omega \) is a subfunction of the function \( FH \) which has error correction totality (Proposition 2.4.9). Hence by Corollary 2.5.8, the function \( FCF_\omega \) itself has error correction totality.

\[ \Box \]
We now turn to the proof that $FCF_\omega$ has the error correction guarantee. The proof parallels the two-part proof method, given in Chapter 2 (Corollary 2.5.14), for the error correction guarantee of MFH. The first part of the proof for MFH is that the related function $FH$ has the error correction guarantee (proved by F & H and presented here as Theorem 2.4.6), where $FH$ is the same as MFH but without the removal of dominated edits. The second part is that MFH is superior (in the sense of Definition 1.9.4) to $FH$, from which we conclude that MFH also has the error correction guarantee.

Similarly to the proof for MFH, the proof of the error correction guarantee of $FCF_\omega$ also has two parts. The first part is that the related function $FCFS_\omega$ has the error correction guarantee (Corollary 6.7.3), where $FCFS_\omega$ is the same as $FCF_\omega$ but without the removal of dominated edits - the “S” in “FCFS_\omega” stands for “simple”. The function $FCFS_\omega$ is calculated via the FCFS Algorithm which is spelt out in detail in Section 6.4. The second part of the proof of the error correction guarantee of $FCF_\omega$ is that $FCF_\omega$ is superior to $FCFS_\omega$ (Corollary 6.8.4), from which we can conclude that $FCF_\omega$ also has the error correction guarantee (Proposition 6.8.5). In the next paragraphs we give a brief overview of the two parts.

The proof of the error correction guarantee of $FCFS_\omega$ follows the same three steps as in the F & H proof of the error correction guarantee of $FH$. In this thesis the three steps are incorporated into the proof outline for Theorem 2.4.6, but F & H presented them as three distinct results: a lifting property theorem, followed by a repeated lifting corollary, from which is deduced an error correction guarantee corollary.

But the proof of the error correction guarantee of $FCFS_\omega$ is not simply a translation of the proof for $FH$ with “$FH$” replaced throughout by “$FCFS_\omega$”: the main reason is that the lifting property of the sequence $(\Omega_k)_{k=0}^N$ used in the proof for $FH$ does not translate to a lifting property of the corresponding sequence $(\Omega_k')_{k=0}^N$ for a putative proof for $FCFS_\omega$. Details are given in Section 6.5, but in essence the reason is this. The proof for $FH$ depends on using a particular subset of $\Omega_{k-1}$ to generate an edit in $\Omega_k$. However the corresponding subset of $\Omega_{k-1}'$ cannot be used to generate an edit in $\Omega_k'$ because of the restrictions on edit generation in the FCFS Algorithm.

Instead of using a sequence of edits such as $\Omega_k$, the proof of the error correction guarantee of $FCFS_\omega$ uses a tree of edit sets, each branch of which has the lifting property. Instead of the edit sets $\Omega_k$ we will construct an edit set $\Gamma_{(i_1,\ldots,i_k)}$ for each node $(i_1,\ldots,i_k)$ of the field code forest. The definition of $\Gamma_{(i_1,\ldots,i_k)}$ is in Section 6.5, but note that the set $\Gamma_{(i_1,\ldots,i_k)}$ is different from the set of edits generated by the FCFS Algorithm at the node $(i_1,\ldots,i_k)$. For each node $(i_1,\ldots,i_n)$ (where $n \geq 1$) the sequence $(\Gamma_{(i_1,\ldots,i_n)})_{k=0}^n$ does have the lifting property - this is given in Section 6.7, in Theorem 6.7.1, the proof of which follows the proof method of the lifting property theorem for the sequence $(\Omega_k)_{k=0}^N$ (F & H Theorem 1). The remainder of Section 6.7 gives us that $FCFS_\omega$ has the error correction guarantee, in Corollary 6.7.2 (repeated lifting property) and Corollary 6.7.3 which parallel F & H Corollaries 1 and 2 respectively.

Although the proof of the lifting property for edit sets of the form $\Gamma_{(i_1,\ldots,i_k)}$ follows the original F & H proof method, it depends on a preliminary result not needed in the original F & H proof. The preliminary result, called the Single Branch Theorem, is given in Section 6.6.
Once we have proved that FCFS_ω has the error correction guarantee, we need the second part of the proof of the error correction guarantee of FCF_ω, namely that FCF_ω is superior to FCFS_ω. In spite of the connection between FCFS_ω and FCF_ω, this superiority result is not immediate, because of the restrictive rules for edit generation in the FCF Algorithm. Details are given in Section 6.8, but a summary is as follows. Although a set X of edits might generate an edit α in the FCFS Algorithm, in the FCF Algorithm a set X′ superior to X might not be allowed to be used to generate an edit dominating α. However, by a suitable inductive definition of nodes, it is possible to find a dominating edit from the FCF Algorithm for each edit from the FCFS Algorithm. Theorem 6.8.3 and Corollary 6.8.4 give us that FCF_ω is superior to FCFS_ω. The error correction guarantee follows quickly in Proposition 6.8.5.

Figure 6.1 summarises the steps in the proof of the correctness of the FCF Algorithm. The bottom of the figure (node (a)) represents the end result that the FCF Al-

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure6.1}
\caption{The main results in the proof of the correctness of the Field Code Forest Algorithm.}
\end{figure}
algorithm is correct, or that for each total ordering $\omega$ of the fields the function $\text{FCF}_\omega$ is covering set correctible (Proposition 6.9.1). The two nodes above (nodes (b) and (c)) represent the two parts of the proof: error correction totality (Lemma 6.3.1) and error correction guarantee (Proposition 6.8.5). The next two nodes (nodes (d) and (e)) represent the fact that the error correction guarantee of $\text{FCF}_\omega$, in turn, also depends on two parts - the error correction guarantee of $\text{FCFS}_\omega$ (Corollary 6.7.3) and the superiority of $\text{FCF}_\omega$ over $\text{FCFS}_\omega$ (Corollary 6.8.4). The error correction guarantee of $\text{FCFS}_\omega$ is proved via the same three steps as the proof of the error correction guarantee of FH: a lifting property (node (g), Theorem 6.7.1), a repeated lifting corollary (node (f), Corollary 6.7.2) and an error correction guarantee corollary (node (d), Corollary 6.7.3). However the steps are applied to the sets of the form $\Gamma_{(i_1, \ldots, i_k)}$ rather than of the form $\Omega_k$, and use a preliminary result called the Single Branch Theorem (node (h), Theorem 6.6.1). The superiority of $\text{FCF}_\omega$ over $\text{FCFS}_\omega$ also depends on a preliminary result about the location of dominating edits (node (i), Theorem 6.8.3).

Before moving to the definition and properties of the set $\Gamma_{(i_1, \ldots, i_k)}$, we define the FCFS Algorithm, in the next section.

### 6.4 The FCFS Algorithm

The proof of the error correction guarantee of $\text{FCF}_\omega$ depends on the error correction guarantee of the function $\text{FCFS}_\omega$. In this section we define $\text{FCFS}_\omega$, and give three basic results which are used in later proofs.

The function $\text{FCFS}_\omega$ is calculated via the FCFS Algorithm, a precise statement of which is given below in Definition 6.4.1. Similarly to the FCF Algorithm (Section 2.6), the FCFS Algorithm works through a field code forest in depth-first order, generating an edit set $\text{GenS}(\sigma, E)$ at each node $\sigma$ in a similar way to $\text{GenI}(\sigma, E)$. But unlike the FCF Algorithm, the edit set associated with each node is not modified as the algorithm traverses the nodes.

**Definition 6.4.1 (Statement of the FCFS Algorithm).**

**INPUT to the algorithm:** a set $E$ of edits on fields $\{1, \ldots, N\}$, and a total ordering $\omega$ of the set $\{1, \ldots, N\}$.

**STEPS in the algorithm:** Traverse the field code forest $F(\omega)$ in depth-first order. At each node $\sigma$ of $F(\omega)$, calculate the set $\text{GenS}(\sigma, E)$, according to Definition 6.4.3 below.

**OUTPUT of the algorithm:** the set

$$\text{FCFS}_\omega(E) = \bigcup \{ \text{GenS}(\sigma, E) \mid \sigma \text{ is a node of } F(\omega) \}.$$ 

Similarly to $\text{GenI}(\sigma, E)$ of the FCF Algorithm, the set $\text{GenS}(\sigma, E)$ depends on the edits at nodes above $\sigma$ in the tree. So, similarly to the FCF Algorithm, for any node $\sigma$ we define the set $\text{BranchS}(\sigma, E)$ as the edits generated at or above $\sigma$, as follows:
Definition 6.4.2. Suppose $E$ is an edit set and $\omega$ is a total ordering of the fields. Suppose also that $\sigma$ is a node of the field code forest $F(\omega)$. Write $\sigma = (i_1, \ldots, i_k)$ where $k \geq 0$. Then

$$\text{BranchS}(\sigma, E) = \bigcup \{ \text{GenS}((i_1, \ldots, i_j), E) \mid j = 0, \ldots, k \}.$$ 

We can now define the set $\text{GenS}(\sigma, E)$.

Definition 6.4.3 (Definition of $\text{GenS}(\sigma, E)$ for Definition 6.4.1). Let $\sigma = (i_1, \ldots, i_m) \subseteq \omega$. Define the set $\text{GenS}((i_1, \ldots, i_m), E)$ inductively as follows:

a. $\text{GenS}((,), E) = E$.

b. If $m \geq 1$ and $\text{GenS}((i_1, \ldots, i_{m-1}), E) = \emptyset$, then $\text{GenS}((i_1, \ldots, i_m), E) = \emptyset$.

c. If $m \geq 1$ and $\text{GenS}((i_1, \ldots, i_{m-1}), E) \neq \emptyset$ then

$$\text{GenS}((i_1, \ldots, i_m), E) = \left\{ \text{FHG}(i_m, X) \mid X \subseteq \text{BranchS}((i_1, \ldots, i_{m-1}), E), \text{ and } \right.$$ 

$$\text{FHG}(i_m, X) \neq \emptyset, \text{ and } i_m \text{ is not involved in } \text{FHG}(i_m, X), \text{ and }$$

$$i_m \text{ is involved in each element of } X, \text{ and }$$

$$\text{each of } i_1, \ldots, i_{m-1} \text{ is uninvolved in each element of } X \right\}.$$ 

For the purposes of later proofs, as for the FCF Algorithm, it is useful to list and number the properties in part c of Definition 6.4.3 given above, as follows:

For $m \geq 1$, the edit $\alpha \in \text{GenS}((i_1, \ldots, i_m), E)$ if and only if

GS1. $\text{GenS}((i_1, \ldots, i_{m-1}), E) \neq \emptyset$;

GS2. $\alpha = \text{FHG}(i_m, X)$;

GS3. $X \subseteq \text{BranchS}((i_1, \ldots, i_{m-1}), E)$;

GS4. $\alpha \neq \emptyset$;

GS5. the field $i_m$ is not involved in $\alpha$;

GS6. the field $i_m$ is involved in each element of $X$;

GS7. each of the fields $i_1, \ldots, i_{m-1}$ is uninvolved in each element of $X$.

Having defined the FCFS Algorithm, we are ready to give some results about it, in particular about multiple steps of edit generation. In order to simplify discussion about multiple steps of edit generation, we introduce, in Definition 6.4.5, an “ancestor” of an edit: an ancestor of edit $\alpha$ is $\alpha$ itself or another edit used to generate $\alpha$, possibly in many steps of edit generation, in the FCFS Algorithm. The ancestor relation is
transitive, as is given below in Lemma 6.4.6. The succeeding results, Lemma 6.4.7 and Corollary 6.4.8, give some basic results about the interactions between uninvolved fields and ancestors.

We start with the definition of “ancestor”. We allow an edit to be its own ancestor in order to simplify some of the proofs. In order to define “ancestor” we first define, in the next definition, the edit \( \alpha \) to be a “predecessor” of edit \( \beta \) when \( \alpha \) is used in the FCFS Algorithm to generate \( \beta \) in one step of edit generation.

**Definition 6.4.4.** Let \( E \) be a set of edits and let \( \omega \) be a total ordering of the fields. Let \( b \) be a positive integer and let \( (i_1, \ldots, i_b) \) be a node of \( F(\omega) \). Let \( \beta \in \text{GenS}( (i_1, \ldots, i_b), E) \). Then the edit \( \alpha \) is a **predecessor at node** \( \sigma_\alpha \) **of** \( \beta \) **on node** \( (i_1, \ldots, i_b) \) **in** \( \text{FCFS}_\omega(E) \) if

1. \( \sigma_\alpha = (i_1, \ldots, i_a) \) where \( 0 \leq a < b \);

and \( \beta \in \text{GenS}(\sigma_\beta, E) \) and the set \( X \) used to generate \( \beta \) contains \( \alpha \), or more formally:

2. \( \alpha \in \text{GenS}(\sigma_\alpha, E) \);

3. there is a set \( X \) such that \( \alpha \in X \) and \( \beta = \text{FHG}(i_b, X) \);

4. \( X \subseteq \text{BranchS}( (i_1, \ldots, i_{b-1}), E) \);

5. \( i_b \) is involved in each element of \( X \); and

6. each of the fields \( i_1, \ldots, i_{b-1} \) is uninvolved in each element of \( X \).

We now use the definition of predecessors to define ancestors.

**Definition 6.4.5.** Let \( E \) be a set of edits and let \( \omega \) be a total ordering of the fields. Let \( b \geq 0 \) and let \( (i_1, \ldots, i_b) \) be a node of \( F(\omega) \). Let \( \beta \in \text{GenS}( (i_1, \ldots, i_b), E) \). Then the edit \( \alpha \) is an **ancestor at** \( \sigma_\alpha \) **of** \( \beta \) **on node** \( (i_1, \ldots, i_b) \) **in** \( \text{FCFS}_\omega(E) \) if

\[
\sigma_\alpha = (i_1, \ldots, i_a) \text{ where } 0 \leq a \leq b;
\]

and

if \( a = b \), then \( \alpha = \beta \);

and

if \( a < b \), there is a sequence of edits \( \alpha = \alpha_0, \ldots, \alpha_p = \beta \), where \( p > 0 \), and there is a sequence of integers \( a = n_0 < \cdots < n_p = b \) such that for each \( k = 0, \ldots, p-1 \), the edit \( \alpha_k \) is a predecessor at \( (i_1, \ldots, i_{n_k}) \) of \( \alpha_{k+1} \) on \( (i_1, \ldots, i_{n_{k+1}}) \).

Note: the above definition means that each edit is one of its own ancestors.

The next lemma tells us that the ancestor relation is transitive.
Lemma 6.4.6. If $\alpha$ is an ancestor at $\sigma_\alpha$ of $\beta$ on $\sigma_\beta$, and $\beta$ is an ancestor at $\sigma_\beta$ of $\gamma$ on $\sigma_\gamma$, then $\alpha$ is an ancestor at $\sigma_\alpha$ of $\gamma$ on $\sigma_\gamma$.

Proof. As a result of the ancestor relationships we can write

$$
\sigma_\alpha = (i_1, \ldots, i_a), \\
\sigma_\beta = (i_1, \ldots, i_b), \text{ and} \\
\sigma_\gamma = (i_1, \ldots, i_c),
$$

where $0 \leq a \leq b \leq c$.

If $a = b$ then $\alpha = \beta$. If $b = c$ then $\beta = \gamma$. In either case $\alpha$ is an ancestor at $\sigma_\alpha$ of $\gamma$ on $\sigma_\gamma$, as required.

Now suppose that $a < b < c$. Then there is a sequence of edits $\alpha = \alpha_0, \ldots, \alpha_p = \beta$, where $p > 0$, and there is a sequence of integers $a = n_0 < \cdots < n_p = b$ such that for each $k = 0, \ldots, p - 1$,

$$
\alpha_k \text{ is a predecessor at } (i_1, \ldots, i_{n_k}) \text{ of } \alpha_{k+1} \text{ on } (i_1, \ldots, i_{n_{k+1}}). \tag{6.4.1}
$$

Also there is a sequence of edits $\beta = \beta_0, \ldots, \beta_q = \gamma$, where $q > 0$, and there is a sequence of integers $b = m_0 < \cdots < m_q = c$ such that for each $k = 0, \ldots, q - 1$,

$$
\beta_k \text{ is a predecessor at } (i_1, \ldots, i_{m_k}) \text{ of } \beta_{k+1} \text{ on } (i_1, \ldots, i_{m_{k+1}}). \tag{6.4.2}
$$

Then the sequence of edits $\alpha = \alpha_0, \ldots, \alpha_p = \beta_0, \ldots, \beta_q = \gamma$ and the sequence of integers $a = n_0 < \cdots < n_p = b = m_0 < \cdots < m_q = c$ have the predecessor properties needed to make $\alpha$ an ancestor at $\sigma_\alpha$ of $\gamma$ on $\sigma_\gamma$. More formally, we define the sequence of edits $\alpha = \gamma_0, \ldots, \gamma_{p+q} = \gamma$ and the sequence of integers $a = r_0, \ldots, r_{p+q} = c$ as follows:

For $j = 0, \ldots, p + q$,

$$
\gamma_j = \begin{cases} 
\alpha_j & \text{if } 0 \leq j \leq p - 1 \\
\beta_{j-p} & \text{if } p \leq j \leq p + q;
\end{cases}
$$

$$
r_j = \begin{cases} 
n_j & \text{if } 0 \leq j \leq p - 1 \\
m_{j-p} & \text{if } p \leq j \leq p + q.
\end{cases}
$$

Then for $j = 0, \ldots, p + q - 1$, the edit $\gamma_j$ is a predecessor at $(i_{r_1}, \ldots, i_{r_j})$ of $\gamma_{j+1}$ on $(i_{r_1}, \ldots, i_{r_{j+1}})$, as follows:

1. if $j = p - 1$, then $\gamma_j = \alpha_{p-1}$, $i_{r_j} = i_{n_{p-1}}$, $\gamma_{j+1} = \beta_0 = \beta = \alpha_p$, $i_{r_{j+1}} = i_{m_0} = i_b = i_{n_p}$: apply Statement 6.4.1 with $k = p - 1$;

2. if $0 \leq j \leq p - 2$, apply Statement 6.4.1 with $k = j$; and

3. if $p \leq j \leq p + q - 1$, apply Statement 6.4.2 with $k = j - p$.

Hence $\alpha$ is an ancestor at $\sigma_\alpha$ of $\gamma$ on $\sigma_\gamma$. \hfill $\square$
The next lemma and corollary tell us about the provenance of uninvolved fields in single or multiple steps of edit generation respectively. Lemma 6.4.7, dealing with a single step of edit generation, tells us that if the field $j$ is uninvolved in an FH-generated edit and $j$ is not the generating field, then all of the generating edits must also have that $j$ uninvolved.

**Lemma 6.4.7.** Suppose $i$ and $j$ are distinct fields and $X$ is an edit set such that $j$ is uninvolved in $\text{FHG}(i, X)$. Then $j$ is uninvolved in each edit of $X$.

**Proof.** Since $j$ is uninvolved in $\text{FHG}(i, X)$, we have that $\text{FHG}(i, X)$ is non-empty. Then the $j$ component of $\text{FHG}(i, X) = A_j = \bigcap \{ A_j^\xi \mid \xi \in X \}$. Hence for each $\xi$ in $X$, we have that $A_j^\xi = A_j$, and $j$ is uninvolved in $\xi$. 

The next corollary extends the above lemma to many steps of edit generation. If there are several consecutive steps of edit generation, none of which use the field $j$ as the generating edit, and the final edit has $j$ uninvolved, then all of the ancestors of the final edit must have had $j$ uninvolved.

**Corollary 6.4.8.** Let $\sigma_\beta \subseteq \{1, \ldots, N\}$ and suppose that $\beta \in \text{GenS}(\sigma_\beta, E)$. Let $\sigma_\alpha \subseteq \{1, \ldots, N\}$ and suppose that $\alpha$ is an ancestor at node $\sigma_\alpha$ of $\beta$ on node $\sigma_\beta$ in $\text{FCFS}_\omega(E)$. Suppose also that $\rho$ is a set of fields each of which is uninvolved in $\beta$. Then the fields in the set $\rho \ \bigcap (\sigma_\beta \setminus \sigma_\alpha)$ are uninvolved in $\alpha$.

**Proof.** Suppose that $j \in \rho \ \bigcap (\sigma_\beta \setminus \sigma_\alpha)$. We will show that $j$ is uninvolved in $\alpha$.

Write $\sigma_\beta = (i_1, \ldots, i_b)$ where $0 \leq b \leq N$. Since $\alpha$ is an ancestor at $\sigma_\alpha$ of $\beta$ on $\sigma_\beta$ we may write $\sigma_\alpha = (i_1, \ldots, i_a)$ where $0 \leq a \leq b$.

We consider two cases. In the first case $a = b$, while in the second case $a < b$.

**Case (i):** Suppose that $a = b$. Then, by the definition of ancestor, $\alpha = \beta$. Since $j \in \rho \ \bigcap (\sigma_\beta \setminus \sigma_\alpha)$, we have that $j \in \rho$, so that by assumption $j$ is uninvolved in $\alpha$.

**Case (ii):** Suppose that $a < b$. Then there is a sequence of edits $\alpha = \alpha_0, \ldots, \alpha_p = \beta$, where $p > 0$, and a sequence of integers $a = n_0 < \cdots < n_p = b$ such that, for all $k = 0, \ldots, p - 1$, the edit $\alpha_k$ is a predecessor at $(i_1, \ldots, i_{n_k})$ of $\alpha_{k+1}$ on $(i_1, \ldots, i_{n_{k+1}})$.

We will prove by induction on $l = p, \ldots, 0$, that $j$ is uninvolved in $\alpha_l$. Setting $l = 0$ will then give the required result.

Firstly, if $l = p$ then $j$ is uninvolved in $\alpha_p = \beta$ because $j \in \rho$.

Now suppose that $0 \leq l < p$ and $j$ is uninvolved in $\alpha_{l+1}$. We will show that $j$ is uninvolved in $\alpha_l$.

By assumption $\alpha_l$ is a predecessor at $(i_1, \ldots, i_{n_l})$ of $\alpha_{l+1}$ on $(i_1, \ldots, i_{n_{l+1}})$. By the definition of predecessor there is a set $X$ containing $\alpha_l$ such that $\alpha_{l+1} = \text{FHG}(i_{n_{l+1}}, X)$.

We can now apply Lemma 6.4.7, because $j \neq i_{n_{l+1}}$ by the following:

1. $i_{n_{l+1}} \in (i_{a+1}, \ldots, i_b)$, because $0 < l + 1 \leq p$ so that $a = n_0 < n_{l+1} \leq n_p = b$; but
2. \( j \notin (i_{a+1}, \ldots, i_b) \) because \( j \in \rho \setminus (\sigma_\beta \setminus \sigma_\alpha) = \rho \setminus (i_{a+1}, \ldots, i_b) \).

Hence by Lemma 6.4.7, \( j \) is uninvolved in \( \alpha_l \).

Setting \( l = 0 \) gives us that \( j \) is uninvolved in \( \alpha_0 = \alpha \), and hence all elements of \( \rho \setminus (\sigma_\beta \setminus \sigma_\alpha) = \rho \setminus (i_{a+1}, \ldots, i_b) \).

\[ \therefore \]

This section has given the definition of the function \( \text{FCFS}_\omega \) via the FCFS Algorithm, and given some basic properties. In particular the underlying function \( \text{GenS} \) has seven defining properties, GS1 to GS7. We have also seen three basic results which will be needed (in Section 6.6) in the proof of the single branch property of \( \Gamma_{(i_1, \ldots, i_k)} \), which itself is introduced in the next section.

6.5 The set \( \Gamma_{(i_1, \ldots, i_k)} \)

The proof of the error correction guarantee of the function \( \text{FCFS}_\omega \) follows the same steps as the proof by Fellegi and Holt for the error correction guarantee of the function \( \text{FH} \), summarised in the proof of Theorem 2.4.6. But we will see in this section that the F & H proof cannot be simply reworked by replacing the function \( \text{FH} \) by the function \( \text{FCFS}_\omega \). The sequence of edits of the form \( \Omega_k \) used in the F & H proof must be replaced by a tree of edit sets of the form \( \Gamma_{(i_1, \ldots, i_k)} \) - one set for each node \( (i_1, \ldots, i_k) \) of the field code forest - defined below in Definition 6.5.1. Before giving the definition of \( \Gamma_{(i_1, \ldots, i_k)} \) we will trace through various attempts to modify Fellegi and Holt’s proof, leading to the definition of \( \Gamma_{(i_1, \ldots, i_k)} \).

Fellegi and Holt’s proof was outlined for Theorem 2.4.6, but we repeat the three main steps here. The first step is the Lifting Property Theorem (F & H Theorem 1) for the sequence \( (\Omega_N, \ldots, \Omega_0) \) of edit sets. The Lifting Property Theorem states that if the record \( v \) satisfies the edits of \( \Omega_k \), then \( v \) can be changed on field \( k \) to satisfy \( \Omega_{k-1} \). The second step is the Repeated Lifting Corollary (F & H Corollary 2) for \( (\Omega_N, \ldots, \Omega_0) \). The Repeated Lifting Corollary states that if \( v \) satisfies \( \Omega_n \) for some \( n \), then \( v \) can be changed on fields \( n, \ldots, 1 \) to satisfy \( \Omega_0 \) which equals \( \text{FH}(E) \). The third step is the Error Correction Guarantee Corollary (F & H Corollary 2), which states that \( \text{FH} \) has the error correction guarantee. The main ideas in the proofs are as follows.

a. For each \( k = 0, \ldots, N \) the set \( \Omega_k \) is defined by

\[ \Omega_k = \{ \alpha \in \text{FH}(E) \mid \text{fields } 1, \ldots, k \text{ are uninvolved in } \alpha \}. \]

b. For each \( k = 1, \ldots, N \), the proof of the Lifting Property Theorem uses a particular subset \( S(\Omega_{k-1}) \) of \( \Omega_{k-1} \) to generate an edit \( \gamma(\Omega_k) = \text{FHG}(k, S(\Omega_{k-1})) \) in the adjacent set \( \Omega_k \). We will write \( S \) and \( \gamma \) instead of \( S(\Omega_{k-1}) \) and \( \gamma(\Omega_k) \) respectively,
to improve legibility where the context is clear. That is,
\[ S \subseteq \Omega_{k-1}, \]
\[ \gamma = \text{FHG}(k, S), \]
with the result that \( \gamma \in \Omega_k. \)

c. The proof of the Repeated Lifting Corollary starts at some set \( \Omega_n \) that is satisfied by the record \( v \) and proceeds stepwise through the sets \( \Omega_n, \ldots, \Omega_1 \). At each step \( k \) from \( n \) down to 1, the originally failing record \( v \) is successively “improved” to satisfy \( \Omega_{k-1} \), by changing the value of the field \( k \). When finally \( k = 1 \), the “improved” record satisfies all edits in \( \Omega_0 \), which is the set \( \text{FH}(E) \) of all edits generated by the FH edit generation process.

d. In order to prove the Error Correction Guarantee Corollary we choose a covering set of the edits of \( \text{FH}(E) \) failed by \( v \) and renumber the fields to ensure that the chosen covering set can be written \( \{1, \ldots, n\} \) for some \( n \). We show that \( v \) satisfies \( \Omega_n \), and then use the Repeated Lifting Corollary to show that \( v \) can be changed on fields \( n, \ldots, 1 \) to satisfy \( \Omega_0 \) which equals \( \text{FH}(E) \).

We now trace through five successive attempts to modify F & H’s proof, leading to the definition of \( \Gamma_{(i_1, \ldots, i_k)} \). The successful attempt is the fourth, with the fifth being an unsuccessful attempt at simplification.

**Attempt 1.** It might seem that in order to prove that the function \( \text{FCFS}_\omega \) has the error correction guarantee, we should be able to use the same proof as the Fellegi-Holt proof except with FH replaced by \( \text{FCFS}_\omega \) throughout the proof. That is, we would replace each edit set \( \Omega_k \) by a new set \( \Omega'_k \), which is the same as \( \Omega_k \) except with FH replaced by \( \text{FCFS}_\omega \):
\[ \Omega'_k = \{ \alpha \in \text{FCFS}_\omega(E) \mid \text{fields } 1, \ldots, k \text{ are uninvolved in } \alpha \}. \]

Then the new set \( S \) and the new edit \( \gamma \) have the following characteristics:
\[ S \subseteq \Omega'_{k-1}, \]
\[ \text{and } \gamma = \text{FHG}(k, S). \]

However such a reworking of the F & H proof fails at two points. Firstly, the edit \( \gamma \) is not necessarily in \( \Omega'_k \). The reason is that the edits of the subset \( S \) are not necessarily all in the one branch of the field code forest, and so might not be able to be used in the FCFS Algorithm to generate the edit \( \gamma \). Secondly, we cannot use a renumbering of the fields to force the covering set to be \( \{1, \ldots, n\} \) for some \( n \) because the FCFS Algorithm depends on a pre-decided ordering of the fields.

**Attempt 2.** We first address the second problem, of not being able to renumber fields in the FCFS Algorithm, by modifying the original Fellegi-Holt proof for the function FH. We note that the Fellegi-Holt proof of the error correction guarantee of FH
still works if the sequence \((\Omega_k)_{k=1}^N\) is replaced by a tree of edit sets corresponding to the field code forest, so that there is no need to renumber fields. For each node \((i_1, \ldots, i_k)\) of the field code forest, we construct an edit set \(\Lambda(i_1, \ldots, i_k)\) defined similarly to \(\Omega_k\):

\[
\Lambda(i_1, \ldots, i_k) = \{ \alpha \in FH(E) | \text{ fields } i_1, \ldots, i_k \text{ are uninvolved in } \alpha \}.
\]

The proof is the same as the original Fellegi-Holt proof with

\[
S \subseteq \Lambda(i_1, \ldots, i_{k-1}),
\]

and \(\gamma = FHG(i_k, S)\),

with the result that \(\gamma \in \Lambda(i_1, \ldots, i_k)\).

In this proof there is no need to renumber fields in the proof of the Error Correction Guarantee Corollary for FH, because each possible covering set is represented by some node \((i_1, \ldots, i_n)\) of the field code forest.

It might seem that in order to prove that the function FCFS\(_\omega\) has the error correction guarantee, we should be able to use the above modified Fellegi-Holt proof, except with FH replaced by FCFS\(_\omega\) throughout the proof. That is, we would replace each edit set \(\Lambda(i_1, \ldots, i_k)\) by a set \(\Lambda'(i_1, \ldots, i_k)\), which is the same as \(\Lambda(i_1, \ldots, i_k)\) except with FH replaced by FCFS\(_\omega\):

\[
\Lambda'(i_1, \ldots, i_k) = \{ \alpha \in FCFS\_\omega(E) | \text{ fields } i_1, \ldots, i_k \text{ are uninvolved in } \alpha \}.
\]

Then following the same method as the original Fellegi-Holt proof gives us

\[
S \subseteq \Lambda'(i_1, \ldots, i_{k-1}),
\]

and \(\gamma = FHG(i_k, S)\).

However, although there is now no need to renumber fields, the edits of the set \(S\) are still not necessarily all on the one branch of the field code forest, because the edits of its superset \(\Lambda'(i_1, \ldots, i_{k-1})\) are not defined to be necessarily all on the one branch. Hence \(S\) still might not be able to be used in the FCFS Algorithm to generate the edit \(\gamma\).

**Attempt 3.** A successful proof of the error correction guarantee of FCFS\(_\omega\) depends on the construction of a new tree of edit sets to replace the tree of sets of the form \(\Lambda'(i_1, \ldots, i_k)\). In contrast to the edits of \(\Lambda'(i_1, \ldots, i_k)\), the edits of the new set should all be generated on the same branch of the field code forest. Thus we define the new edit set \(\Lambda''(i_1, \ldots, i_k)\) to be the same as \(\Lambda'(i_1, \ldots, i_k)\), except that all of its edits are on the same branch, and FH is replaced by FCFS\(_\omega\):  

\[
\Lambda''(i_1, \ldots, i_k) = \{ \alpha \in FCFS\_\omega(E) | \text{ fields } i_1, \ldots, i_k \text{ are uninvolved in } \alpha, \text{ and } \alpha \text{ is generated on the branch that includes } (i_1, \ldots, i_k) \}.
\]
Following the Fellegi-Holt proof method gives

\[ S \subseteq \Lambda''_{(i_1, \ldots, i_{k-1})}, \]

and \( \gamma = \text{FHG}(i_k, S) \).

However, the proof still fails because, although \( \gamma \) is in \( \text{FCFS}_\omega(E) \), it is not necessarily in \( \Lambda''_{(i_1, \ldots, i_k)} \) as needed in the proof. Figure 6.2 shows an example for \( \omega = (1, \ldots, 8) \) and \( (i_1, \ldots, i_k) = (1,3,5,7) \). In this example there are no edits generated at the node \((1,3,5)\), that is \( \text{GenS}((1,3,5), E) = \emptyset \). Therefore the edits of the set \( S \) are all generated at the nodes \( (), (1) \) and \( (1,3) \). As needed in the Fellegi-Holt method of proof, we have

\[ S \subseteq \Lambda''_{(1,3,5)}, \]

and \( \gamma = \text{FHG}(7, S) \).

Figure 6.2: Subtree of the field code forest \( F((1, \ldots, 8)) \). The annotations to the nodes refer to the example discussed as part of Attempt 3 on pages 181–182. Consider an edit set \( E \) for which, in the FCFS Algorithm, there are no edits generated at the node \((1,3,5)\), but there are edits generated at the nodes \( (), (1) \) and \( (1,3) \). Then the edits of \( S \) are generated at nodes \( (), (1) \) and \( (1,3) \). The edit \( \gamma = \text{FHG}(7, S) \) cannot be generated at node \((1,3,5,7)\), although it is generated at \((1,3,7)\). Therefore, although the edits of \( S \) are assumed to all be in \( \Lambda''_{(1,3,5)} \), the edit \( \gamma \) is not in \( \Lambda''_{(1,3,5,7)} \) as needed for the proof of the lifting property.
But $\gamma$ is in general not in $\Lambda''_{(1,3,5,7)}$ because, if it were in $\Lambda''_{(1,3,5,7)}$, then it would have to be in $\text{GenS}((1,3,5,7), E)$, which is empty because $\text{GenS}((1,3,5), E)$ is empty. Instead, the edit $\gamma$ is in $\text{GenS}((1,3,7), E)$, giving the inspiration for defining below a variation $\Lambda'''_{(i_1,\ldots,i_k)}$ to the set $\Lambda''_{(i_1,\ldots,i_k)}$.

**Attempt 4.** The two last attempts at proof failed for opposite reasons. The definition of $\Lambda''_{(i_1,\ldots,i_k)}$ was too restrictive, causing the edit $\gamma$ to not necessarily be in the set $\Lambda''_{(i_1,\ldots,i_k)}$. On the other hand, the definition of $\Lambda'_{(i_1,\ldots,i_k)}$ was not restrictive enough, causing the edits of the set $S$ to not necessarily be on the same branch.

In order to obtain a proof of the error correction guarantee of $\text{FCFS}_\omega$, we need conditions on the tree of edit sets that are somewhere between the two last attempts. We define below a new set $\Lambda'''_{(i_1,\ldots,i_k)}$ which is the same as $\Lambda''_{(i_1,\ldots,i_k)}$ except that its edits may also be generated at those nodes that are subsets of $(i_1, \ldots, i_k)$. Figure 6.3 shows the nodes of the field code forest $F((1,\ldots,8))$ at which the edits of $\Lambda'''_{(1,3,5,7)}$ may be generated.

$$\Lambda'''_{(i_1,\ldots,i_k)} = \{ \alpha \in \text{FCFS}_\omega(E) \mid \text{fields } i_1, \ldots, i_k \text{ are uninvolved in } \alpha, \text{ and there is a subset } \sigma \text{ of } (i_1, \ldots, i_k) \text{ such that } \gamma \in \text{GenS}(\sigma, E) \}.$$ 

This definition still looks like it is not restrictive enough because the edits of $S$ are defined to be not necessarily all on the same branch. However we will see in the next section (in the Single Branch Theorem, Theorem 6.6.1) that all the edits in $\Lambda'''_{(i_1,\ldots,i_k)}$ are also generated during the FCFS Algorithm on some single branch of the field code forest, not necessarily the branch of $(i_1, \ldots, i_k)$. On the other hand, the definition is not too restrictive, allowing $\gamma$ to be an element of $\Lambda'''_{(i_1,\ldots,i_k)}$.

![Figure 6.3](image-url)
Indeed, sequences of the form \((\Lambda'''(i_1, \ldots, i_k))^n_{k=0}\) do have the lifting property (to be proved in Theorem 6.7.1) and can be used to prove that the function \(\text{FCFS}_\omega\) has the error correction guarantee (to be proved in Corollary 6.7.3). When we give the formal definition below we will write \(\Gamma\) instead of the cumbersome \(\Lambda'''\). Before proceeding to the formal definition, we consider one other possible sequence of edits that looks like a likely candidate for the lifting property.

**Attempt 5.** The definition of \(\Lambda'''(i_1, \ldots, i_k)\) seems unduly complicated. Instead of the set \(\Lambda'''(i_1, \ldots, i_k)\), why not use the simpler set \(\text{GenS}((i_1, \ldots, i_k), E)\) of edits generated at the node \((i_1, \ldots, i_k)\)? We get

\[ S \subseteq \text{GenS}((i_1, \ldots, i_{k-1}), E), \]

and \(\gamma = \text{FHG}(i_k, S)\),

but the edit \(\gamma\) need not be in \(\text{GenS}((i_1, \ldots, i_k), E)\), because the edits in \(S\) need not have field \(i_k\) involved, and thereby fail property GS6 for \(\text{FCFS}_\omega\) (page 174). However this problem does not arise for \(\Lambda'''(i_1, \ldots, i_k)\), because of the restrictions on its definition.

We now formally define \(\Gamma_{(i_1, \ldots, i_k)}\).

**Definition 6.5.1.** Let \(\omega\) be a total ordering of the fields and let \(E\) be a set of edits. Let \(0 \leq k \leq N\) and let \((i_1, \ldots, i_k)\) be a node of the field code forest \(F(\omega)\). We define the set \(\Gamma((i_1, \ldots, i_k), E)\) below. For convenience, when the context is clear, we will write \(\Gamma_{(i_1, \ldots, i_k)}\) for \(\Gamma((i_1, \ldots, i_k), E)\).

\[ \Gamma_{(i_1, \ldots, i_k)} = \{ \gamma \in \text{FCFS}_\omega(E) \mid i_1, \ldots, i_k \text{ are uninvolved in } \gamma, \text{ and there is a } \sigma \subseteq (i_1, \ldots, i_k) \text{ such that } \gamma \in \text{GenS}(\sigma, E) \}. \]

Note: This definition implies that \(\Gamma() = \text{GenS}(), E) = E\).

With the above definition of \(\Gamma_{(i_1, \ldots, i_k)}\) for each node \((i_1, \ldots, i_k)\) of the field code forest, we will be able to use in Section 6.7 the three steps of the F & H proof method to show that \(\text{FCFS}_\omega\) has the error correction guarantee. We will first show that for any \(n\) with \(0 \leq n \leq N\), the sequence \((\Gamma_{(i_1, \ldots, i_k)})^n_{k=0}\) has the lifting property. Secondly, we will prove the repeated lifting property. Finally, we will choose a covering set \((i_1, \ldots, i_n)\) of the edits of \(\text{FCFS}_\omega(E)\) failed by the given record \(v\); prove that \(v\) satisfies \(\Gamma_{(i_1, \ldots, i_n)}\); and then conclude, by the repeated lifting property, that \(v\) can be changed on fields \(i_1, \ldots, i_n\) to satisfy \(\Gamma()\) which is the set \(E\).

The proof of the error correction guarantee of \(\text{FCFS}_\omega\) depends on the fact that all the edits of \(\Gamma_{(i_1, \ldots, i_k)}\) appear on some single branch during the FCFS Algorithm. That is, there is a node \((t_1, \ldots, t_s)\) such that \(\Gamma_{(i_1, \ldots, i_k)} \subseteq \text{BranchS}((t_1, \ldots, t_s), E)\). This property will be the subject of Theorem 6.6.1 (Single Branch Theorem) in the next section.
6.6 The single branch property of $\Gamma_{(i_1,\ldots,i_k)}$

The main property that makes $\Gamma_{(i_1,\ldots,i_k)}$ useful is in the Single Branch Theorem (Theorem 6.6.1), given below, which mainly states that all the edits in $\Gamma_{(i_1,\ldots,i_k)}$ are generated during the FCFS Algorithm on some single branch of the field code forest. Figure 6.4 includes a simple example, in which $\Gamma_{(1,3,5,7)}$ contains just two edits, $\gamma_1$ and $\gamma_2$, which are generated at nodes $\tau_{\gamma_1} = (1,5)$ and $\tau_{\gamma_2} = (5,7)$ respectively. However, the edit $\gamma_2$ is also generated at the node $\tau = (1,5,7)$, on the same branch as $\gamma_1$. Thus all of the set $\Gamma_{(1,3,5,7)}$ is generated on the one branch containing the node $(1,5,7)$. Note that, in general, the node $\tau$ need not be on the same branch as $\tau_{\gamma_1}$ or $\tau_{\gamma_2}$ even though both of $\gamma_1$ and $\gamma_2$ are also generated on the branch of $\tau$. The proof of Theorem 6.6.1 will refer to the figure and explain the various annotations.

**Theorem 6.6.1 (Single branch theorem).** Let $\omega$ be a total ordering of the fields and let $E$ be a set of edits. Let $0 \leq m \leq N$ and let $(i_1,\ldots,i_m)$ be a node of the field code forest $F(\omega)$. Then there exists a node $\tau = (t_1,\ldots,t_s)$ which is a subset of $(i_1,\ldots,i_m)$ such that

(a) $\Gamma_{(i_1,\ldots,i_m)} \subseteq \text{BranchS}(\tau,E)$; and

(b) if $E \neq \emptyset$ then $\text{GenS}(\tau,E) \neq \emptyset$.

(That is, there is a node $\tau$ so that all of $\Gamma_{(i_1,\ldots,i_m)}$ appears at or above $\tau$ in the FCFS Algorithm, and also $\tau$ uses only fields in $(i_1,\ldots,i_m)$, and also $\text{GenS}(\tau,E) \neq \emptyset$.)

**Outline of proof.** Before giving the full proof, we give an outline of the proof, demonstrated by the example in Figure 6.4, with precise definitions to follow. In the example, we suppose that

$$\omega = (1,\ldots,8)$$

and that

$$(i_1,\ldots,i_m) = (1,3,5,7).$$

We will first associate each edit $\gamma$ of $\Gamma_{(i_1,\ldots,i_m)}$ with a minimal node $\tau_\gamma$ at which $\gamma$ is generated. In the example we suppose that

$$\Gamma_{(1,3,5,7)} = \{\gamma_1, \gamma_2\}$$

where

$$\tau_{\gamma_1} = (1,5), \text{ and}$$

$$\tau_{\gamma_2} = (5,7).$$

We then let $\tau = (t_1,\ldots,t_s)$ be the union of all nodes of the form $\tau_\gamma$. In the example

$$\tau = (1,5,7), \text{ so that } t_1 = 1, t_2 = 5, t_3 = 7, \text{ and } s = 3.$$
The single branch property of $\Gamma_{(i_1, \ldots, i_k)}$

 '.. 185

Figure 6.4: Subtree of the field code forest $F((1, \ldots, 8))$ consisting of those nodes at which the edits of $\Gamma_{(1,3,5,7)}$ may be generated. The circles and annotations refer to the example discussed in the outline on pages 184–185 of the proof of Theorem 6.6.1. In the example, $\Gamma_{(1,3,5,7)}$ consists of just two edits, $\gamma_1$ and $\gamma_2$. Minimal nodes at which $\gamma_1$ and $\gamma_2$ are generated are $\tau_{\gamma_1} = (1,5)$ and $\tau_{\gamma_2} = (5,7)$ respectively. The edits $\gamma_1$ and $\gamma_2$ are also both generated on the branch containing $\tau = (1,5,7)$, which is the union of $\tau_{\gamma_1}$ and $\tau_{\gamma_2}$. Indeed, all of the ancestors of $\gamma_1$ on $\tau_{\gamma_1}$ and of $\gamma_2$ on $\tau_{\gamma_2}$ are also generated on the branch containing $\tau$. Such ancestors (other than those generated at the root node) comprise the union of the sets $\Delta_1$, $\Delta_2$ and $\Delta_3$, generated at the circled nodes and introduced on page 186 in the detailed proof. Statement 6.6.1 (page 186) means that the edits of $\Delta_1 \cup \Delta_2 \cup \Delta_3$ are also all generated on the branch containing $(1,5,7)$.

at nodes ending in $t_j$ (where $j = 1, \ldots, s$) are also generated at the node $(t_1, \ldots, t_j)$. In the example of Figure 6.4 this means that the ancestors of the edit $\gamma_2$ that are generated at node $\gamma_2$ are also generated at node $\tau_2$, and that the edit $\gamma_2$ itself is also generated at the node $(1,5,7)$. In general we will be able to conclude, since $\gamma$ is its own ancestor and $\tau_{\gamma} \subseteq (t_1, \ldots, t_s)$, that $\gamma$ is generated at the node $(t_1, \ldots, t_j)$ for some $j$, and result (a) of this theorem follows. Result (b) of this theorem can be proved using the fact that $\tau$ is defined to be the union of all the $\tau_{\gamma}$'s.

Proof. We now proceed to the proof itself. First consider the case when $\Gamma_{(i_1, \ldots, i_m)} = \emptyset$. Then let $\tau = ()$ and the result follows.

From now on assume that $\Gamma_{(i_1, \ldots, i_m)} \neq \emptyset$.

We first construct $\tau$.

For each $\gamma$ in $\Gamma_{(i_1, \ldots, i_m)}$ choose a node $\tau_{\gamma}$ such that $\tau_{\gamma}$ is a minimal string (under $\subseteq$) with the properties that $\tau_{\gamma} \subseteq (i_1, \ldots, i_m)$ and $\gamma \in \text{GenS}(\tau_{\gamma}, E)$. Such a $\tau_{\gamma}$ exists by
the definition of $\Gamma_{(i_1,\ldots,i_m)}$ in Definition 6.5.1. Let

$$\tau = \bigcup\{\tau_\gamma \mid \gamma \in \Gamma_{(i_1,\ldots,i_m)}\},$$

and write

$$\tau = (t_1,\ldots,t_s).$$

Note that $(t_1,\ldots,t_s) \subseteq (i_1,\ldots,i_m)$.

Consider the case $s = 0$.

Then $\tau = ()$. Then for each $\gamma$ in $\Gamma_{(i_1,\ldots,i_m)}$, we have that $\tau_\gamma = ()$ and that $\gamma \in \text{GenS}()$. Result (a) of this theorem follows. Result (b) of this theorem holds because $\text{GenS}() = \text{E}$.

Proof of result (a) for $s > 0$.

We will use the set $\Delta_k$, defined for each $k = 1,\ldots,s$ as the set of edits generated at those nodes that end with $t_k$ and that are ancestors of some element $\gamma$ of $\Gamma_{(i_1,\ldots,i_m)}$ on $\tau_\gamma$. That is,

$$\Delta_k = \{\delta \in \text{FCFS}_\omega(E) \mid \text{there is a } \gamma \in \Gamma_{(i_1,\ldots,i_m)} \text{ and there is a } \sigma \circ (t_k) \subseteq \tau_\gamma \text{ such that } \delta \text{ is an ancestor at } \sigma \circ (t_k) \text{ of } \gamma \text{ on } \tau_\gamma\}.$$

In the example of Figure 6.4, the edits of $\Delta_1 \cup \Delta_2 \cup \Delta_3$ are generated at the circed nodes:

- the edits of $\Delta_1$ are generated at the node (1),
- the edits of $\Delta_2$ are generated at the nodes (1, 5) and (5), and
- the edits of $\Delta_3$ are generated at the node (5, 7).

We will prove Statement 6.6.1 below about $\Delta_k$, from which we will eventually be able to deduce result (a). Statement 6.6.1 says that

for each $j = 1,\ldots,s$, we have that $\Delta_j \subseteq \text{GenS}((t_1,\ldots,t_j),E)$. \quad (6.6.1)

That is, although each $\delta$ in $\Delta_j$ appears in the FCFS Algorithm at some node $\sigma \circ (t_j)$, the edit $\delta$ also appears in the FCFS Algorithm at the specific node $(t_1,\ldots,t_j)$. Thus Statement 6.6.1 implies that all the edits in $\Delta_1 \cup \cdots \cup \Delta_s$ appear on the one branch $\text{BranchS}(\tau,E)$. In the example of Figure 6.4, the set $\Delta_1 \cup \Delta_2 \cup \Delta_3$ is a subset of $\text{BranchS}((1,5,7),E)$.

We can deduce result (a) from Statement 6.6.1 as follows. Suppose that $\gamma \in \Gamma_{(i_1,\ldots,i_m)}$. Firstly suppose that $\tau_\gamma = ()$. Then $\gamma \in \text{GenS}() \subseteq \text{BranchS}(\tau,E)$ as required. Now suppose that $\tau_\gamma \neq ()$. Since $\tau_\gamma \subseteq \tau$, write $\tau_\gamma$ as $\sigma \circ (t_l)$ where $1 \leq l \leq s$. Since $\gamma$ is an ancestor of itself, we can say that $\gamma$ is an ancestor at $\sigma \circ (t_l)$ of $\gamma$ on $\tau_\gamma$. Hence by Statement 6.6.1, we have that $\gamma \in \Delta_l \subseteq \text{GenS}((t_1,\ldots,t_l),E) \subseteq \text{BranchS}(\tau,E)$, as required.
We now prove Statement 6.6.1 by induction on \(j\). Firstly, suppose that \(j = 1\). Let \(\delta \in \Delta_1\). By the definition of \(\Delta_1\), there is a \(\sigma \circ (t_1) \subseteq \tau\) such that \(\delta \in \text{GenS}(\sigma \circ (t_1), E)\). But \(t_1\) is the first element of \(\tau\), hence \(\sigma = ()\). Hence \(\delta \in \text{GenS}((t_1), E)\), and hence \(\Delta_1 \subseteq \text{GenS}((t_1), E)\), as required.

Now suppose that \(\Delta_j \subseteq \text{GenS}((t_1, \ldots, t_j), E)\) for each \(j\) in \(\{1, \ldots, l\}\), where \(1 \leq l < s\). We will show that \(\Delta_{l+1} \subseteq \text{GenS}((t_1, \ldots, t_{l+1}), E)\). Suppose that \(\delta \in \Delta_{l+1}\). By the definition of \(\Delta_{l+1}\) there is an \(\alpha \in \Gamma_{(t_1, \ldots, t_m)}\) and a \(\sigma \circ (t_{l+1}) \subseteq \tau_\alpha\) such that \(\delta\) is an ancestor at \(\sigma \circ (t_{l+1})\) of \(\alpha\) on \(\tau_\alpha\). Hence

\[
\delta \in \text{GenS}(\sigma \circ (t_{l+1}), E).
\]

Write

\[
\sigma = (t_{u_1}, \ldots, t_{u_p}) \text{ where } p \geq 0.
\]

Then we have that

\[
0 \leq u_1 \leq \cdots \leq u_p \leq l.
\]

We will show that \(\delta \in \text{GenS}((t_1, \ldots, t_{l+1}), E)\) by checking the conditions GS1–GS7 for FCFS\(_{\omega}\) (page 174):

GS1. \(\text{GenS}((t_1, \ldots, t_l), E) \neq \emptyset\) by the following argument. Since \(\tau = \cup \{\tau_\gamma \mid \gamma \in \Gamma_{(t_1, \ldots, t_m)}\}\) and \(t_l \in \tau\), there is a \(\beta \in \Gamma_{(t_1, \ldots, t_m)}\) such that \(t_l \in \tau_\beta\). Write \(\tau_\beta = \sigma_1 \circ (t_l) \circ \sigma_2\). Then there is an ancestor \(\epsilon\) of \(\tau_\beta\), because otherwise \(\beta \in \text{GenS}(\rho, E)\) for some \(\rho\) that is a subset of \(\tau_\beta \setminus (t_l)\), so that \(\tau_\beta\) would not be minimal as assumed. This means that \(\epsilon \in \Delta_l\), so that \(\Delta_l \neq \emptyset\). By the induction assumption, \(\Delta_l \subseteq \text{GenS}((t_1, \ldots, t_l), E)\), so that \(\text{GenS}((t_1, \ldots, t_l), E) \neq \emptyset\), and Condition GS1 holds.

GS2, GS4, GS5, and GS6 (for \(\delta \in \text{GenS}((t_1, \ldots, t_{l+1}), E)\)) follow by applying properties GS2, GS4, GS5, and GS6 to the fact that \(\delta \in \text{GenS}(\sigma \circ (t_{l+1}), E)\):

GS2. \(\delta = \text{FHG}(t_{l+1}, X)\) where \(X \subseteq \text{BranchS}(\sigma, E)\);

GS4. \(\delta \neq \emptyset\);

GS5. the field \(t_{l+1}\) is not involved in \(\delta\);

GS6. the field \(t_{l+1}\) is involved in each element of \(X\).

GS3. To show that \(X \subseteq \text{BranchS}((t_1, \ldots, t_l), E)\), suppose that \(\xi \in X\).

Then, since \(X \subseteq \text{BranchS}(\sigma, E)\), there is an \(h\) with \(0 \leq h \leq p\) such that \(\xi \in \text{GenS}((t_{u_1}, \ldots, t_{u_h}), E)\), and hence \(\xi\) is an ancestor at \((t_{u_1}, \ldots, t_{u_h})\) of \(\delta\) on \(\sigma \circ (t_{l+1})\). We also have that \(\delta\) is an ancestor at \(\sigma \circ (t_{l+1})\) of \(\alpha\) on \(\tau_\alpha\). Hence by the transitivity of the ancestor relation (Lemma 6.4.6), \(\xi\) is an ancestor at \((t_{u_1}, \ldots, t_{u_h})\) of \(\alpha\) on \(\tau_\alpha\).

If \(u_h = 0\) then \(\xi \in \text{GenS}((), E) \subseteq \text{BranchS}((t_1, \ldots, t_l), E)\), as required.
Now suppose that \( u_h > 0 \). Since \( \xi \) is an ancestor at \((t_{u_1}, \ldots, t_{u_h})\) of \( \alpha \) on \( \tau_{\alpha} \), and \( \alpha \in \Gamma_{(i_1, \ldots, i_m)} \), and \( u_h \neq 0 \), we have that \( \xi \in \Delta_{u_h} \). Since \( u_h \leq l \), we can use the induction assumption to give us that \( \Delta_{u_h} \subseteq \text{GenS}((t_1, \ldots, t_{u_h}), E) \). Hence \( \xi \in \text{GenS}((t_1, \ldots, t_{u_h}), E) \subseteq \text{BranchS}((t_1, \ldots, t_l), E) \), since \( u_h \leq l \). So Condition GS3 holds.

GS7. We show that each of \( t_1, \ldots, t_l \) is uninvolved in each \( \xi \) of \( X \). We have that \( \delta \) is an ancestor at \( \sigma \circ (t_{l+1}) \) of \( \alpha \) on \( \tau_{\alpha} \). We also have that \( \alpha \) has \( i_1, \ldots, i_m \) uninvolved (since \( \alpha \in \Gamma_{(i_1, \ldots, i_m)} \)), and hence has \( (t_1, \ldots, t_l) \) uninvolved. Also, from the definition of \( \alpha \), we can write \( \tau_{\alpha} = \sigma \circ (t_{l+1}) \circ \sigma' \), where \( \sigma' \subseteq (t_{l+2}, \ldots, t_s) \).

By Corollary 6.4.8,

the set of uninvolved fields of \( \delta \)

\[
\Gamma_{(i_1, \ldots, i_m)}(t_1, \ldots, t_l) \setminus (\tau_{\alpha} \setminus (\sigma \circ (t_{l+1})))
\]

\[
= (t_1, \ldots, t_l) \setminus \sigma'
\]

\[
= (t_1, \ldots, t_l), \text{ since } \sigma' \subseteq (t_{l+2}, \ldots, t_s).
\]

We now use the fact that \( \delta = \text{FHG}(t_{l+1}, X) \) and Lemma 6.4.7 to consider the uninvolved fields of \( \xi \). Each field \( t_1, \ldots, t_l \) is distinct from \( t_{l+1} \) and is uninvolved in \( \delta \). Hence by Lemma 6.4.7 each field \( t_1, \ldots, t_l \) is uninvolved in \( \xi \).

Hence we have satisfied the properties GS1–GS7, so that \( \delta \in \text{GenS}((t_1, \ldots, t_{l+1}), E) \), and so \( \Delta_{l+1} \subseteq \text{GenS}((t_1, \ldots, t_{l+1}), E) \), thus completing the induction proof. Hence Statement 6.6.1 holds and we have result (a) of the theorem.

**Proof of result (b) for \( s > 0 \).**

Since \( s > 0 \), we have that \( \tau \neq () \). Since \( \tau = \bigcup \{ \tau_{\gamma} \mid \gamma \in \Gamma_{(i_1, \ldots, i_m)} \} \), there is a \( \gamma \) in \( \Gamma_{(i_1, \ldots, i_m)} \) such that \( t_s \in \tau_{\gamma} \). Then \( \tau_{\gamma} \) can be written \( \sigma \circ (t_s) \), since all elements of \( \tau_{\gamma} \) are of the form \( t_k \) where \( 1 \leq k \leq s \). Since \( \gamma \) is an ancestor of itself, we have that \( \gamma \) is an ancestor at \( \sigma \circ (t_s) \) of \( \gamma \) on \( \tau_{\gamma} \). That is, \( \gamma \in \Delta_s \) and hence \( \Delta_s \neq \emptyset \). But from Statement 6.6.1, we have that \( \Delta_s \subseteq \text{GenS}(\tau, E) \). Hence \( \text{GenS}(\tau, E) \neq \emptyset \).

The next section will use the above theorem to prove the error correction guarantee for the FCFS Algorithm. The proof will be able to use a subset of \( \Gamma_{(i_1, \ldots, i_{k-1})} \) to generate an edit in \( \Gamma_{(i_1, \ldots, i_k)} \) because all of \( \Gamma_{(i_1, \ldots, i_{k-1})} \) is generated during the FCFS Algorithm on the one branch.

### 6.7 Error correction guarantee for FCFS\(_\omega\)

This section completes the proof of the correctness of the FCFS Algorithm. Corollary 6.7.3 will state that FCFS\(_\omega\) has the error correction guarantee.

The proof of the error correction guarantee of FCFS\(_\omega\) parallels the three steps of Fellegi and Holt’s proof of the error correction guarantee of FH, given in this thesis.
as Theorem 2.4.6. The proofs below of Theorem 6.7.1, Corollary 6.7.2 and Corollary 6.7.3 follow the style of Fellegi and Holt’s Theorem 1, Corollary 1 and Corollary 2 respectively.

Theorem 6.7.1 has a similar structure to Fellegi and Holt’s Theorem 1. Similarly to F & H Theorem 1, Theorem 6.7.1 states that if the record \( v \) satisfies all the edits in the set \( \Gamma_{i_1, \ldots, i_k} \) (of Definition 6.5.1) then field \( i_k \) of \( v \) can be changed so that the new record satisfies all the edits in \( \Gamma_{i_1, \ldots, i_{k-1}} \). The proof works by contradiction. We assume that field \( i_k \) of record \( v \) cannot be changed to satisfy \( \Gamma_{i_1, \ldots, i_k} \). We use this assumption to create a subset of \( \Gamma_{i_1, \ldots, i_{k-1}} \) which then generates an edit of \( \Gamma_{i_1, \ldots, i_k} \) failed by \( v \). This contradicts our assumption that \( v \) satisfies all of \( \Gamma_{i_1, \ldots, i_k} \).

Corollary 6.7.2 is similar to Fellegi and Holt’s Corollary 1. Corollary 6.7.2 states that if \( v \) satisfies all edits in \( \Gamma_{i_1, \ldots, i_k} \) then fields \( i_1, \ldots, i_k \) can be changed so that the new record satisfies all edits in \( E \). Corollary 6.7.2, in effect, gives a weakened form of the error correction guarantee, where the covering set is constrained to be \( \{i_1, \ldots, i_k\} \), a particular subset of the set of fields.

Fellegi and Holt proved the error correction guarantee for any covering set in their Corollary 2, by renumbering the fields. An equivalent renumbering is not necessary for the FCFS Algorithm because each subset of the set of fields appears as a node somewhere in the field code forest. The proof of Corollary 6.7.3 uses this property of field code forests to give the error correction guarantee of FCFS\(_\omega\).

The first result, Theorem 6.7.1, is the core result of this section.

**Theorem 6.7.1 (Lifting property of \( (\Gamma_{i_1, \ldots, i_k})^n_{k=0} \)).** Let \( \omega \) be a total ordering of the fields, and let \( E \) be a set of edits. Let \( 1 \leq n \leq N \) and let \( (i_1, \ldots, i_n) \) be a node of the field code forest \( F(\omega) \). Then the sequence \( (\Gamma_{i_1, \ldots, i_k})^n_{k=0} \) has the lifting property.

(That is, if \( v \) is a record and \( 1 \leq k \leq n \) and \( v \) satisfies each edit in \( \Gamma_{i_1, \ldots, i_k} \), then \( \{i_k\} \in \mathcal{E}(\Gamma_{i_1, \ldots, i_{k-1}}), v \).)

**Proof.** If \( E = \emptyset \) then for all \( k \), the set \( \Gamma_{i_1, \ldots, i_k} = \emptyset \) and hence \( (\Gamma_{i_1, \ldots, i_k})^n_{k=0} \) has the lifting property. From now on assume that \( E \neq \emptyset \).

The proof works by contradiction. Assume that \( (\Gamma_{i_1, \ldots, i_k})^n_{k=0} \) does not have the lifting property. Then there is a \( k \) with \( 1 \leq k \leq n \) and a record \( v \) that satisfies all edits in \( \Gamma_{i_1, \ldots, i_k} \) but \( \{i_k\} \notin \mathcal{E}(\Gamma_{i_1, \ldots, i_{k-1}}, v) \). That is, \( \{i_k\} \) does not yield a correction to \( v \) with respect to \( \Gamma_{i_1, \ldots, i_{k-1}} \). We will use this assumption to construct a subset \( S \) of edits in \( \Gamma_{i_1, \ldots, i_{k-1}} \) such that the edit \( \text{FHG}(i_k, S) \) has the contradictory properties that it is both failed by \( v \) and satisfied by \( v \).

**Construction of the set \( S \) and the edit \( \beta = \text{FHG}(i_k, S) \).**

Since \( \{i_k\} \) does not yield a correction to \( v \) with respect to \( \Gamma_{i_1, \ldots, i_{k-1}} \), then no matter how the field \( i_k \) is changed, the new record will fail some edit in \( \Gamma_{i_1, \ldots, i_{k-1}} \). There is one such new record for each \( a \) in \( A_{i_k} \), and so we define the record \( v^a = (v_1^a, \ldots, v_N^a) \) as follows:

\[
v_j^a = \begin{cases} v_j & \text{if } j \neq i_k \\ a & \text{if } j = i_k \end{cases}
\]
Then for each $a$ in $A_{i_k}$ there is an edit $\alpha^a$ in $\Gamma_{(i_1, \ldots, i_{k-1})}$ such that $v^a$ fails $\alpha^a$. That is, for $j = 1, \ldots, N$, we have that $v_j^a \in A_j^a$. Let

$$S = \{\alpha^a \mid a \in A_{i_k}\},$$

and let

$$\beta = \text{FHG}(i_k, S).$$

**Normal form of \(\beta\).**

Before showing that $\beta$ is failed by $v$, we first find the normal form of $\beta$. By the definition of FHG (Definition 2.4.1), we can write

$$\beta = \prod_{j=1}^{i_k-1} \bigcap_{a^a \in S} A_j^a \times \bigcup_{a^a \in S} \prod_{j=i_k+1}^{N} \bigcap_{a^a \in S} A_j^a. \quad (6.7.1)$$

The above expression is in normal form provided that $\beta$ is non-empty. We can verify that $\beta$ is non-empty because each of its components is non-empty, as follows:

if $j \neq i_k$, then (j component of $\beta$) $=$ $\bigcap_{a^a \in S} A_j^a \supseteq \bigcap_{a \in A_{i_k}} \{v_j^a\} = \{v_j\}$, and \hspace{1cm} (6.7.2)

for field $i_k$, we have (i_k component of $\beta$) $=$ $\bigcup_{a^a \in S} A_{i_k}^a \supseteq \bigcup_{a \in A_{i_k}} \{v_{i_k}^a\} = \bigcup_{a \in A_{i_k}} \{a\} = A_{i_k}$. \hspace{1cm} (6.7.3)

Hence $\beta$ is non-empty and Equation 6.7.1 is in normal form.

The contradictory properties that $\beta$ is both failed by $v$ and satisfied by $v$.

We now show both that $v$ fails $\beta$ and that $v$ satisfies $\beta$. Firstly, by Expressions 6.7.2 and 6.7.3, we have that $v \in \beta$ and hence the record $v$ fails the edit $\beta$. However, the remainder of this proof will show that $v$ satisfies $\beta$.

We will show that $v$ satisfies $\beta$ by showing that $\beta \in \Gamma_{(i_1, \ldots, i_{k-1})}$, all of whose elements by assumption are satisfied by $v$. We will show that $\beta$ satisfies all of the conditions of Definition 6.5.1 to be in $\Gamma_{(i_1, \ldots, i_{k-1})}$.

We first find a substring of $(i_1, \ldots, i_k)$ at which $\beta$ is generated. By Theorem 6.6.1, there is a substring $\tau = (t_1, \ldots, t_s)$ of $(i_1, \ldots, i_{k-1})$ such that

(a) $\Gamma_{(i_1, \ldots, i_{k-1})} \subseteq \text{BranchS}(\tau, E)$; \hspace{1cm} (6.7.4)

and

(b) $\text{GenS}(\tau, E) \neq \emptyset$. \hspace{1cm} (6.7.5)

We will show that $\beta$ is generated at the node $\tau \circ (i_k)$, that is, that $\beta \in \text{GenS}(\tau \circ (i_k), E)$,

by verifying that $\beta$ satisfies conditions GS1 to GS7 for FCFS$_\omega$ (page 174). We will
then be able to deduce that $\beta \in \Gamma_{(i_1, \ldots, i_k)}$ by using the definition of $\Gamma_{(i_1, \ldots, i_k)}$.

We verify conditions GS1 to GS7 as follows:

GS1. $\text{GenS}(\tau, E) \neq \emptyset$ by Statement 6.7.5.

GS2. $\beta = \text{FHG}(i_k, S)$, by the definition of $\beta$.

GS3. $S \subseteq \Gamma_{(i_1, \ldots, i_{k-1})}$, by the definition of $\alpha^a$.

GS4. $\beta \neq \emptyset$ by Statements 6.7.2 and 6.7.3.

GS5. The field $i_k$ is not involved in $\beta$, by Statement 6.7.3.

GS6. The field $i_k$ is involved in each $\alpha^a$, because otherwise, by the argument below, we find that $v^a$ satisfies $\alpha^a$, contrary to assumption.

Suppose that $i_k$ is uninvolved in $\alpha^a$. Then, by the definition of $\Gamma_{(i_1, \ldots, i_k)}$, we have that $\alpha^a \in \Gamma_{(i_1, \ldots, i_k)}$, because:

(a) each of $i_1, \ldots, i_k$ is uninvolved in $\alpha^a$: $i_k$ is uninvolved in $\alpha^a$ by assumption; and each of $i_1, \ldots, i_{k-1}$ is uninvolved in $\alpha^a$ since $\alpha^a \in \Gamma_{(i_1, \ldots, i_{k-1})}$;

(b) there is an $\alpha \subseteq (i_1, \ldots, i_k)$ such that $\alpha^a \in \text{GenS}(\alpha, E)$: since $\alpha^a \in \Gamma_{(i_1, \ldots, i_{k-1})}$, we can choose $\alpha \subseteq (i_1, \ldots, i_{k-1})$ such that $\alpha^a \in \text{GenS}(\alpha, E)$.

Then, by the assumption that $v$ satisfies all edits in $\Gamma_{(i_1, \ldots, i_k)}$, we have that $v$ satisfies $\alpha^a$. Hence, there is an $l$ in $\{1, \ldots, N\}$ such that $v_l \notin A^a_{i_k}$.

We have that $l \neq i_k$ because:

$$v_{i_k} \in A_{i_k}, \quad \text{by the definition of a record}$$

$$= A^a_{i_k}, \quad \text{since } i_k \text{ is uninvolved in } \alpha^a.$$

But if $l \neq i_k$ then by the definition of $v^a$ we have that $v_l = v_l$. Hence $v_l^a \notin A^a_{i_k}$ and $v^a$ satisfies $\alpha^a$, contrary to the assumption that $v^a$ fails $\alpha^a$.

GS7. Each field of $\tau$ is uninvolved in each $\alpha^a$ because $\tau \subseteq (i_1, \ldots, i_{k-1})$ and $\alpha^a \in \Gamma_{(i_1, \ldots, i_{k-1})}$.

We have now shown that $\beta \in \text{GenS}(\tau \circ (i_k), E)$. The next step is to show that $\beta \in \Gamma_{(i_1, \ldots, i_k)}$, using Definition 6.5.1 of $\Gamma_{(i_1, \ldots, i_k)}$, as follows:

1. $\beta$ satisfies the first condition in Definition 6.5.1 of $\Gamma_{(i_1, \ldots, i_{k-1})}$: The field $i_k$ is uninvolved in $\beta$ because $\beta \in \text{GenS}(\tau \circ (i_k), E)$. Also, each of the fields $i_1, \ldots, i_{k-1}$ is uninvolved in $\beta$ because $\beta = \text{FHG}(i_k, S)$, and each edit $\alpha^a$ in $S$ has the fields $i_1, \ldots, i_{k-1}$ uninvolved.
2. \( \beta \) satisfies the second condition in Definition 6.5.1 of \( \Gamma_{(i_1, \ldots, i_{k-1})} \): From the above, \( \beta \in \text{GenS}(\tau \circ (i_k), E) \). Also, \( \tau \circ (i_k) \subseteq (i_1, \ldots, i_k) \) by the properties of \( \tau \).

We have now shown that \( \beta \in \Gamma_{(i_1, \ldots, i_k)} \), from which we can conclude that \( \psi \) satisfies \( \beta \). Thus we have derived a contradiction, that \( \psi \) both satisfies and does not satisfy \( \beta \), from the assumption that \( \{i_k\} \notin \mathcal{L}(\Gamma_{(i_1, \ldots, i_{k-1})}, \psi) \). Hence indeed \( \{i_k\} \in \mathcal{L}(\Gamma_{(i_1, \ldots, i_{k-1})}, \psi) \), and \( (\Gamma_{(i_1, \ldots, i_k)})_{k=0}^n \) does indeed have the lifting property, as required.

Thus Theorem 6.7.1 states a “weakened” version of the error correction guarantee, where just one particular field of the record is changed. Applying Theorem 6.7.1 repeatedly, to different fields, gives a less weak version of the error correction guarantee. This happens in Corollary 6.7.2, below, in which the set of fields that is changed is \( \{i_1, \ldots, i_k\} \), which is a subset of the set of fields \( \{1, \ldots, N\} \).

**Corollary 6.7.2.** Let \( \omega \) be a total ordering of the fields, and let \( E \) be a set of edits. Let \( 1 \leq n \leq N \) and let \( (i_1, \ldots, i_n) \) be a node of the field code forest \( F(\omega) \). Suppose that the record \( \psi = (v_1, \ldots, v_N) \) satisfies all the edits in \( \Gamma_{(i_1, \ldots, i_n)} \). Then \( \{i_1, \ldots, i_n\} \in \mathcal{L}(E, \psi) \).

**Proof.** We prove, by induction, for \( h = n, \ldots, 0 \), that
\[
\{i_{h+1}, \ldots, i_n\} \in \mathcal{L}(\Gamma_{(i_1, \ldots, i_h)}, \psi). \tag{6.7.6}
\]

Then, to obtain the required result, set \( h = 0 \) and note that \( \Gamma_{()} = E \).

Let \( h = n \). Then Statement 6.7.6 becomes \( \emptyset \in \mathcal{L}(\Gamma_{(i_1, \ldots, i_n)}, \psi) \) which holds because, by assumption, \( \psi \) satisfies \( \Gamma_{(i_1, \ldots, i_n)} \).

Now suppose that Statement 6.7.6 holds for \( h = k \) where \( n \geq k \geq 1 \). That is \( \{i_{k+1}, \ldots, i_n\} \in \mathcal{L}(\Gamma_{(i_1, \ldots, i_k)}, \psi) \). We will show that Statement 6.7.6 holds for \( h = k-1 \).

By the induction assumption, there is a record \( \psi^k = (v^k_1, \ldots, v^k_N) \) such that
\[
(a) \text{ if } j \notin \{i_{k+1}, \ldots, i_n\}, \text{ then } v^k_j = v_j; \quad \tag{6.7.7a}
(b) \psi^k \text{ satisfies all edits in } \Gamma_{(i_1, \ldots, i_k)}. \tag{6.7.7b}
\]

Using Statement 6.7.7b, by Theorem 6.7.1, we have that \( \{i_k\} \in \mathcal{L}(\Gamma_{(i_1, \ldots, i_{k-1})}, \psi) \).

Hence there is a record \( \psi^{k-1} = (v^{k-1}_1, \ldots, v^{k-1}_N) \) such that
\[
(a) \text{ if } j \neq i_k, \text{ then } v^{k-1}_j = v^k_j; \quad \tag{6.7.8a}
(b) \psi^{k-1} \text{ satisfies all edits in } \Gamma_{(i_1, \ldots, i_{k-1})}. \tag{6.7.8b}
\]

Using these results we now have that:
(a) If \( j \notin \{ i_k, \ldots, i_n \} \) then \( v_{k-1} = v_k \), by Statement 6.7.8a
\[
= v_j, \quad \text{by Statement 6.7.7a;}
\]
and
\[
(\text{b}) \ v_k \text{ satisfies all edits in } \Gamma_{(i_1, \ldots, i_{k-1})}, \quad \text{by Statement 6.7.8b.}
\]
Hence \( \{ i_k, \ldots, i_n \} \) yields a correction \( v_{k-1} \) to \( v \) with respect to \( \Gamma_{(i_1, \ldots, i_{k-1})} \), that is Statement 6.7.6 holds for \( h = k - 1 \).

Hence by induction, Statement 6.7.6 holds for \( h = n, \ldots, 0 \). Setting \( h = 0 \) gives the required result. \(\square\)

The next result, Corollary 6.7.3, uses the above corollary to give the error correction guarantee of FCFS. For any field ordering \( \omega \). Whereas the above corollary gives a “weak” form of the error correction guarantee, Corollary 6.7.3 gives the full error correction guarantee.

**Corollary 6.7.3.** If \( \omega \) is a total ordering of the fields, then FCFS has the error correction guarantee.

**Proof.** Suppose that \( v = (v_1, \ldots, v_N) \) is a record, and that \( E \) is an edit set. Let \( C \in \mathcal{C}(\text{FCFS}_\omega(E), v) \), ie \( C \) is a covering set of the edits of FCFS failed by \( v \). We will show that \( C \in \mathcal{E}(E, v) \).

Abusing notation slightly, let \( C \) also represent the node of \( F(\omega) \) with the same fields as the set \( C \). Such a node exists because every subset of the set of fields is represented by some node in the field code forest.

If \( C = () \), then there are no failed edits and \( v \) satisfies \( E \), so that \( C \in \mathcal{E}(E, v) \).

From now on, suppose that \( C \neq () \), and let \( C = (i_1, \ldots, i_n) \), with \( n \geq 1 \).

The record \( v \) satisfies all edits in \( \Gamma_{(i_1, \ldots, i_n)} \), by the following argument. Suppose, to the contrary, that there is an edit \( \alpha \) in \( \Gamma_{(i_1, \ldots, i_n)} \) that is failed by \( v \). Then \( \alpha \in \mathcal{X}(\text{FCFS}_\omega(E), v) \) and hence, by the definition of covering sets, \( \alpha \) has an involved field in the covering set \( \{ i_1, \ldots, i_n \} \). But by definition, each of the fields \( i_1, \ldots, i_n \) is uninvolved in each edit in \( \Gamma_{(i_1, \ldots, i_n)} \), so no such \( \alpha \) can exist and \( v \) cannot fail any edit of \( \Gamma_{(i_1, \ldots, i_n)} \).

Then by Corollary 6.7.2, \( C = \{ i_1, \ldots, i_n \} \in \mathcal{E}(E, v) \), as required. \(\square\)

This section has given the proof that the function FCFS has the error correction guarantee. The method of proof has the same spirit as the original proof by Fellegi and Holt for the function FH. But the sequence of edit sets of form \( \Omega_k \) used by Fellegi and Holt has been replaced by the more complicated tree of sets of form \( \Gamma_{(i_1, \ldots, i_k)} \).

The function FCFS almost, but not completely, represents the Field Code Forest Algorithm: the function FCFS does not include the Domination Rules (Definition 2.6.5). In the next section we will consider the function FCF which is a modification of FCFS and does include the Domination Rules.
6.8 Error correction guarantee for $FCF_\omega$

This section gives the final steps of the proof of the correctness of the Field Code Forest Algorithm. Proposition 6.8.5 will state that, for any total ordering $\omega$ of the fields, the function $FCF_\omega$ has the error correction guarantee, thus sealing the correctness of the FCF Algorithm.

The proof that $FCF_\omega$ has the error correction guarantee follows from the error correction guarantee of $FCFS_\omega$ - which was proved in the previous section. The proof for $FCF_\omega$ uses the seemingly obvious fact, given in Corollary 6.8.4, that the function $FCF_\omega$ is superior to $FCFS_\omega$ - seemingly obvious because the only difference between $FCF_\omega$ and $FCFS_\omega$ is that $FCF_\omega$ uses the Domination Rules.

However the proof that $FCF_\omega$ is superior to $FCFS_\omega$ is more complex than one might expect. One might expect that $FCF_\omega$ is superior to $FCFS_\omega$ because each edit generated in the FCF Algorithm is dominated by some edit generated at the same node in the FCF Algorithm. In other words, one might expect that the two trees mirror each other, with dominating edits in the tree of the FCF Algorithm.

However the two trees need not mirror each other. Indeed, the edits at node $(i_1, \ldots, i_k)$ ($k \geq 1$) need not mirror each other even if the branch to $(i_1, \ldots, i_{k-1})$ is mirrored. The short reason for such a lack of mirroring is that property GS6 for $FCFS_\omega$ (page 174), requiring that the generating field be involved in each generating edit, need not carry over to dominating edits. In more detail, the reason is as follows. Suppose that the edit $\xi$ is used in the FCFS Algorithm to generate edit $\alpha$, with generating field $i_k$. Suppose that $\xi'$ mirrors $\xi$, that is $\xi'$ dominates $\xi$ and is generated in the FCF Algorithm at the same node as $\xi$. The edit $\xi$ must have field $i_k$ involved (by property GS6) but the edit $\xi'$ need not. Hence $\xi'$ need not satisfy property G6 for $FCF_\omega$ (page 43), which requires that the generating field be involved in each generating edit. Hence, in the FCF Algorithm, it might not be possible to use the edit $\xi'$ to generate an edit that dominates $\alpha$.

Even though there is a lack of mirroring of nodes between the two algorithms, the above discussion about $\xi'$ does give the insight to show that $FCF_\omega$ is superior to $FCFS_\omega$. For if $\xi'$ has $i_k$ uninvolved then $\xi'$ itself dominates the edit $\alpha$. So $\alpha$ is dominated by an edit generated in the FCF Algorithm which is not at the same node but is somewhere in the branch above the node. Theorem 6.8.3 generalises this insight to state that each edit generated at node $(i_1, \ldots, i_k)$ in the FCF Algorithm is dominated by an edit somewhere on the branch to $\rho_k$, which is a subset of $(i_1, \ldots, i_k)$. (The node $\rho_k$ is inductively defined in Definition 6.8.2.) It is then a simple step to Corollary 6.8.4 stating that $FCF_\omega$ is superior to $FCFS_\omega$.

The above discussion (but not the proof to follow) ignores the fact that, in the FCF Algorithm, the edits at any node change as the algorithm progresses because of the Domination Rules. However edits can only expand (as sets) as the algorithm progresses, ensuring that the domination relationships are retained. The next lemma gives the relevant superset relationships between the changing edits in the FCF Algorithm.

**Lemma 6.8.1.** Let $E$ be an edit set and let $\omega$ be a total ordering of the fields. Let $\sigma_1$, $\sigma_2$, $\sigma_3$ be nodes of $F(\omega)$ which appear in the order (not necessarily consecutively) $\sigma_1$, $\sigma_2$, $\sigma_3$.
\(\sigma_2, \sigma_3\) in the traversal of \(F(\omega)\) for the FCF Algorithm. Some or all of \(\sigma_1, \sigma_2, \sigma_3\) may be equal. Then the following statements hold.

(a) \(\text{BranchV}(\sigma_1, E, \sigma_2)\) is superior to \(\text{GenI}(\sigma_1, E)\), that is:
   
   if \(\alpha \in \text{GenI}(\sigma_1, E)\) then there is an edit \(\alpha^+\) in \(\text{BranchV}(\sigma_1, E, \sigma_2)\) such that \(\alpha^+ \supseteq \alpha\).

(b) \(\text{BranchV}(\sigma_1, E, \sigma_3)\) is superior to \(\text{BranchV}(\sigma_1, E, \sigma_2)\), that is:
   
   if \(\alpha \in \text{BranchV}(\sigma_1, E, \sigma_2)\) then there is an edit \(\alpha^+\) in \(\text{BranchV}(\sigma_1, E, \sigma_3)\) such that \(\alpha^+ \supseteq \alpha\).

(c) \(\text{FCF}_\omega(E)\) is superior to \(\text{BranchV}(\sigma_1, E, \sigma_2)\), that is:
   
   if \(\alpha \in \text{BranchV}(\sigma_1, E, \sigma_2)\) then there is an edit \(\alpha^+\) in \(\text{FCF}_\omega(E)\) such that \(\alpha^+ \supseteq \alpha\).

Proof. Parts (a) and (b) follow from the two Domination Rules for the FCF Algorithm, because in both rules edits are either replaced by larger edits or left unchanged. Part (c) follows from Part (b), using the facts that:

(i) \(\text{FCF}_\omega(E) = \bigcup \{\text{GenF}(\sigma, E) \mid \sigma \text{ is a node of } F(\omega)\}\), and

(ii) \(\text{GenF}(\sigma, E) = \text{GenV}(\sigma, E, \nu)\), where \(\nu\) is the last node traversed in the tree, and hence

(iii) \(\text{FCF}_\omega(E) = \bigcup \{\text{BranchV}(\sigma, E, \nu) \mid \sigma \text{ is a node of } F(\omega)\}\).

\(\Box\)

We now give the inductive definition of the sequence of nodes of form \((\rho_k)_{k=0}^m\). In Theorem 6.8.3, the nodes \(\rho_k\) are where the dominating edits are generated in the FCF Algorithm.

**Definition 6.8.2.** Suppose that \(E\) is an edit set, \(\omega\) is a total ordering of the fields, and \((i_1, \ldots, i_m)\) is a node of \(F(\omega)\). We inductively define the nodes \(\rho(E, (\))\), \(\rho(E, (i_1))\), \ldots, \(\rho(E, (i_1, \ldots, i_m))\). For convenience, when the context is clear, for \(j = 1, \ldots, m\), we write \(\rho_j\) instead of \(\rho(E, (i_1, \ldots, i_j))\).

Define \(\rho_0 = (\)\).

For \(k = 1, \ldots, m\), define \(\rho_k = \begin{cases} \rho_{k-1} \circ (i_k) & \text{if } \text{GenI}(\rho_{k-1} \circ (i_k), E) \neq \emptyset \\ \rho_{k-1} & \text{if } \text{GenI}(\rho_{k-1} \circ (i_k), E) = \emptyset. \end{cases} \)

We will write \(\rho_k = (r_1, \ldots, r_{g(k)})\), where \(g(0) = 0\), \(g\) is monotonic increasing, and \((r_1, \ldots, r_{g(k)}) \subseteq (i_1, \ldots, i_k)\). We can write \(\rho_k\) in this form because \(\rho_{k-1}\) is an initial string of \(\rho_k\), or empty.

Note that if \(E\) is non-empty, then \(\text{GenI}(\rho_0, E) = E\) is non-empty, and then for each \(k = 1, \ldots, m\) we also have that \(\text{GenI}(\rho_k, E)\) is non-empty.
The Field Code Forest Algorithm

We have now dealt with all the lemmas and definitions needed for the core of this section which is the next theorem. It tells us that if the edit $\alpha$ is generated at node $(i_1, \ldots, i_k)$ in the FCFS Algorithm, then a dominating edit $\alpha^+$ is generated at or above node $\rho_k$ in the FCF Algorithm.

**Theorem 6.8.3.** Let $E$ be an edit set, let $\omega$ be a total ordering of the fields, and let $(i_1, \ldots, i_k)$ be a node of the field code forest $F(\omega)$. Then $\text{BranchV}(\rho_k, E, \rho_k)$ is superior to $\text{GenS}((i_1, \ldots, i_k), E)$. That is, if $\alpha \in \text{GenS}((i_1, \ldots, i_k), E)$ then there exists an edit $\alpha^+$ such that $\alpha^+ \supseteq \alpha$ and $\alpha^+ \in \text{BranchV}(\rho_k, E, \rho_k)$.

**Proof.** Firstly note that if $E = \emptyset$ then $\text{GenS}((i_1, \ldots, i_k), E) = \emptyset$, and the result holds vacuously.

From now on, assume that $E \neq \emptyset$. Then by Definition 6.8.2 we have that $\text{GenI}(\rho_k, E) \neq \emptyset$, and hence $\text{BranchV}(\rho_k, E, \rho_k) \neq \emptyset$.

The proof for $E \neq \emptyset$ is by induction on $k$.

Firstly, let $k = 0$. Suppose that $\alpha \in \text{GenS}((i_1, \ldots, i_k), E) = E = \text{GenI}(()), E)$. By Lemma 6.8.1(a), there is an edit $\alpha^+$ in $\text{BranchV}(()), E, ())$ such that $\alpha^+ \supseteq \alpha$, as required.

Now, suppose that the theorem holds for $k = 0, \ldots, h$ where $h \geq 0$ and $(i_1, \ldots, i_{h+1})$ is a node of $F(\omega)$. Let $k = h + 1$. We are given that $\alpha \in \text{GenS}((i_1, \ldots, i_{h+1}), E)$, and we will find an $\alpha^+$ in $\text{BranchV}(\rho_k, E, \rho_k)$ such that $\alpha^+ \supseteq \alpha$.

Since $\alpha \in \text{GenS}((i_1, \ldots, i_{h+1}), E)$, Properties GS1 to GS7 for FCFS$_n$ (page 174) hold for $\alpha$ at node $(i_1, \ldots, i_{h+1})$. Let $X$ be an edit set with which the properties hold, so that $X \subseteq \text{BranchS}((i_1, \ldots, i_h), E)$ and $\alpha = \text{FHG}(i_{h+1}, X)$. The set $X$ and the edit $\alpha$ are represented on the right-hand side of Figure 6.5, which depicts some of the items to be presented in the proof.

For each edit $\xi$ in $X$, we will use the induction assumption to construct an edit $\xi'$ which dominates $\xi$ and which is generated during the FCF Algorithm at or above the node $\rho_h$. We will define the set $X'$ to be the set of all edits $\xi'$ where $\xi \in X$. The set $X'$ is superior to the set $X$ and is represented in the top left-hand corner of Figure 6.5.

Then we might hope that the edit $\alpha'$, generated from $X'$ using generating field $i_{h+1}$, dominates $\alpha$ and is generated in the FCF Algorithm at node $\rho_{h+1}$. The edit $\alpha' = \text{FHG}(i_{h+1}, X')$ and is depicted in the bottom left-hand corner of Figure 6.5. While it is true that $\alpha'$ dominates $\alpha$, it need not be true that $\alpha'$ is generated in the FCF Algorithm at node $\rho_{h+1}$, because the set $X'$ need not satisfy Property G6 for FCFS$_n$ (page 43). However when $X'$ fails Property G6, it turns out that some edit $\xi'_0$ in $X'$ dominates $\alpha$ anyway. Figure 6.6, which is identical to Figure 6.5 except in the bottom left-hand corner, represents the situation when $X'$ fails Property G6. Hence we will consider two cases: (1) $X'$ satisfies G6 (represented by Figure 6.5), and (2) $X'$ fails G6 (represented by Figure 6.6).

**Construction and properties of $X'$.** We first formally construct $X'$ and derive some of its properties. Since $X \subseteq \text{BranchS}((i_1, \ldots, i_h), E)$, then for each $\xi$ in $X$ there is a field
§6.8  Error correction guarantee for $\text{FCF}_\omega$

Figure 6.5: Main items in Case 1 of the proof of Theorem 6.8.3. At the top right-hand corner of the diagram is the set $X$, all of whose edits are generated in the FCFS Algorithm on the branch containing $(i_1, \ldots, i_h)$. The edit $\alpha$, shown at the bottom right-hand corner, is generated in the FCFS Algorithm from $X$ using the generating field $i_{h+1}$. By the induction assumption there is a set $X'$, shown at the top left-hand corner, that is superior to $X$ and whose edits are generated in the FCF Algorithm on the branch containing $\rho_h$. The edit $\alpha'$, shown at the bottom left-hand corner, parallels $\alpha$ in that it is generated from $X'$ using the generating field $i_{h+1}$, and therefore is a superset of $\alpha$. However $\alpha$ might not be generated during the FCF Algorithm, unless the assumption is made that $X'$ satisfies Property G6 (Case 1).

Figure 6.6: Main items in Case 2 of the proof of Theorem 6.8.3. This diagram is identical to the diagram of Figure 6.5 except for the item in the bottom left-hand corner. The diagram represents the case when $\alpha'$ of Figure 6.5 is not generated during the FCF Algorithm. Then $X'$ contains an edit $\xi'_0$, shown at the bottom left-hand corner, that is a superset of $\alpha$. 
\(i_{j_\xi}\) where \(0 \leq j_\xi \leq h\) such that

\[\xi \in \text{GenS}((i_1, \ldots, i_{j_\xi}), E).\]

By the induction assumption, for each \(\xi\) in \(X\) there is an edit \(\xi^*\) with \(\xi^* \supseteq \xi\) and

\[\xi^* \in \text{BranchV}(\rho_{j_\xi}, E, \rho_{j_\xi}).\]

Since \(g\) is monotonic increasing and \(j_\xi \leq h\), we have that \(g(j_\xi) \leq g(h)\). Then in the tree traversal, the node \((r_1, \ldots, r_{g(j_\xi)}) = \rho_{j_\xi}\) is before or the same as \((r_1, \ldots, r_{g(h)}) = \rho_h\). Hence we can use Lemma 6.8.1(b) to say that there is an edit \(\xi'\) with \(\xi' \supseteq \xi^*\) and

\[\xi' \in \text{BranchV}(\rho_{j_\xi}, E, \rho_h).\]

Also, since \(\rho_{j_\xi}\) is above \(\rho_h\) in the field code forest, and on the same branch,

\[\xi' \in \text{BranchV}(\rho_h, E, \rho_h).\] (6.8.1)

Note that since \(\xi' \supseteq \xi^*\) and \(\xi^* \supseteq \xi\), we have that

\[\xi' \supseteq \xi.\] (6.8.2)

Let

\[X' = \{\xi' | \xi \in X\}.\]

Then, from the above, for each \(\xi\) in \(X\), the edit \(\xi'\) dominates \(\xi\) and is generated during the FCF Algorithm at or above node \(\rho_h\).

We now consider the two foreshadowed cases. Either \(X'\) satisfies Property G6 or it does not.

**Case 1.** \(X'\) satisfies Property G6, that is, \(i_{h+1}\) is involved in every \(\xi'\) of \(X'\).

Let \(\alpha' = \text{FHG}(i_{h+1}, X')\). We will later use \(\alpha'\) to construct an edit \(\alpha^+\) which satisfies the theorem.

Note that \(\alpha' \supseteq \alpha\) by Lemma 2.4.10 which can be used because the function from \(X\) to \(X'\) that takes \(\xi\) to \(\xi'\) is surjective, and, for each \(\xi\) in \(X\), we have that \(\xi' \supseteq \xi\).

We now show that \(\alpha' \in \text{GenI}(\rho_h \circ (i_{h+1}), E)\) by checking the properties G1 to G7 for FCF\(\omega\) (page 43):

G1. Since \(E \neq \emptyset\), then by Definition 6.8.2, we have that \(\text{GenI}(\rho_h, E) \neq \emptyset\).

G2. \(\alpha' = \text{FHG}(i_{h+1}, X')\), by definition.

G3. \(X' \subseteq \text{BranchV}(\rho_h, E, \rho_h)\), by Statement 6.8.1.

G4. Since \(\alpha \neq \emptyset\) (by GS4 applied to \(\alpha\)) and \(\alpha' \supseteq \alpha\), we have that \(\alpha' \neq \emptyset\).

G5. The field \(i_{h+1}\) is uninvolved in \(\alpha\) (by GS5 applied to \(\alpha\)). Since \(\alpha' \supseteq \alpha\), the field \(i_{h+1}\) is also uninvolved in \(\alpha'\).
G6. The field $i_{h+1}$ is involved in each $\xi'$ of $X'$, by assumption.

G7. By GS7 applied to $\alpha$, each field $i_1, \ldots, i_h$ is uninvolved in each $\xi$ of $X$. Since each edit of $X'$ can be written as $\xi'$ for some $\xi$ in $X$ and $\xi' \supseteq \xi$, we have that each edit in $X'$ has the fields $i_1, \ldots, i_h$ uninvolved. But

\[ \rho_h = (r_1, \ldots, r_{g(h)}) \subseteq (i_1, \ldots, i_h), \] by Definition 6.8.2 for $\rho_h$.

Hence each field of $\rho_h$ (that is $r_1, \ldots, r_{g(h)}$) is uninvolved in each edit of $X'$.

Since Properties G1 to G7 hold, we have that $\alpha' \in \text{GenI}(\rho_h \circ (i_{h+1}), E)$.

In order to satisfy the theorem, we use Lemma 6.8.1(a) to obtain an edit $\alpha'$ in $\text{BranchV}(\rho_h \circ (i_{h+1}), E, \rho_h \circ (i_{h+1}))$ such that $\alpha' \supseteq \alpha$. Since $\alpha' \supseteq \alpha$, we have that $\alpha' \supseteq \alpha$.

We also have that $\rho_h \circ (i_{h+1}) = \rho_{h+1}$, by Definition 6.8.2 applied to $\rho_{h+1}$ and by noting that $\alpha' \in \text{GenI}(\rho_h \circ (i_{h+1}), E)$ which implies that $\text{GenI}(\rho_h \circ (i_{h+1}), E) \neq \emptyset$.

Hence $\alpha' \subseteq \text{BranchV}(\rho_{h+1}, E, \rho_{h+1})$ and $\alpha' \supseteq \alpha$ and so the theorem holds for $k = h + 1$ for Case 1.

Case 2. $X'$ fails Property G6, that is, $i_{h+1}$ is uninvolved in some $\xi' = \xi'_0$ in $X'$, where by the definition of $X'$, the edit $\xi_0 \in X$.

Then $\xi'_0 \supseteq \alpha$ because:

\[ A^{\xi'_0}_{i_{h+1}} = A_{i_{h+1}} \quad \text{since } i_{h+1} \text{ is uninvolved in } \xi'_0 \]
\[ \supseteq A^\alpha_{i_{h+1}}, \]

and if $j \neq i_{h+1}$ then

\[ A^{\xi'_0}_j \supseteq A^\xi_0 \quad \text{since } \xi'_0 \supseteq \xi_0 \text{ by Statement 6.8.2, and using Lemma 2.2.15} \]
\[ \supseteq \bigcap \{ A^\xi_j \mid \xi \in X \} \quad \text{since } \xi_0 \in X \]
\[ = A^\alpha_j \quad \text{since } \alpha = \text{FHG}(i_{h+1}, X) \text{ and } j \neq i_{h+1}, \text{ and } \alpha \neq \emptyset \text{ (by Property GS5 applied to } \alpha). \]

Hence $\xi'_0 \supseteq \alpha$.

Using Lemma 6.8.1(b) and the fact that $\xi'_0 \in \text{BranchV}(\rho_h, E, \rho_h)$, there is an edit $\alpha'$ such that $\alpha' \supseteq \xi'_0$ and

\[ \alpha' \in \text{BranchV}(\rho_h, E, \rho_{h+1}) \quad \text{since } \rho_{h+1} \text{ comes after or is the same as } \rho_h \text{ in the tree traversal} \]
\[ \subseteq \text{BranchV}(\rho_{h+1}, E, \rho_{h+1}) \quad \text{since } \rho_{h+1} \text{ is on the same branch as } \rho_h \text{ and is below or the same as } \rho_h. \]

Since $\xi'_0 \supseteq \alpha$ we have that $\alpha' \supseteq \alpha$ and the theorem holds for $k = h + 1$ for Case 2 as well as for Case 1. \[\therefore\]
The above theorem leads to the next corollary which gives us the domination relationship between the FCFS and FCF Algorithms. The final result, that FCF_ω has the error correction guarantee, follows quickly.

**Corollary 6.8.4.** If ω is a total ordering of the fields, then the function FCF_ω is superior to the function FCFS_ω.

*Proof.* Let E be an edit set and suppose that α ∈ FCFS_ω(E). Then there is a node (i_1, . . . , i_k) such that α ∈ GenS((i_1, . . . , i_k), E). By Theorem 6.8.3 there is an edit α⁺ such that α⁺ ⊇ α and α⁺ ∈ BranchV(ρ_k, E, ρ_k). Then by Lemma 6.8.1(c) there is an edit α' with α' ⊇ α⁺ such that α' ∈ FCF_ω(E). Since α⁺ ⊇ α and hence α' ⊇ α, the result follows.

The next, final, result of this section tells us that the function FCF_ω has the error correction guarantee. This property is the last result needed to give us that the FCF Algorithm is indeed correct.

**Proposition 6.8.5.** If ω is a total ordering of the fields, then FCF_ω has the error correction guarantee.

*Proof.* Since FCFS_ω has the error correction guarantee (Corollary 6.7.3) and FCF_ω is superior to FCFS_ω (Corollary 6.8.4), then, by Corollary 2.5.13, we have that FCF_ω has the error correction guarantee.

Having shown that FCF_ω has the error correction guarantee, we can, in the next section, wrap up the proof of the correctness of the Field Code Forest Algorithm.

### 6.9 The Field Code Forest Algorithm is correct

Early in this chapter we showed that FCF_ω has error correction totality (Lemma 6.3.1). Since the previous section also gives us the error correction guarantee for FCF_ω (Proposition 6.8.5), we now have that FCF_ω is covering set correctible. In other words the FCF Algorithm is correct:

**Proposition 6.9.1.** The Field Code Forest Algorithm is correct.

*Proof.* The statement that “the FCF Algorithm is correct” means that for each total ordering ω of the fields, the function FCF_ω is covering set correctible, which follows from Proposition 6.8.5 (error correction guarantee) and Lemma 6.3.1 (error correction totality).

### 6.10 Conclusion

This chapter has given a proof that the FCF Algorithm is correct, or equivalently that the function FCF_ω is covering set correctible. This means that the covering set method applied to the output of the FCF Algorithm gives exactly all error localisation solutions.
The main motivation for this chapter is that there have been questions about the correctness of the FCF Algorithm. The proof published by Garfinkel, Kunnathur, and Liepins (1986b) is about another edit generation function, here called GKL(ω, −), and not about FCFω. In addition Winkler (1997) has questioned the correctness of the FCF Algorithm.

It might seem that the correctness of the algorithm could be proved by simply adjusting GKL’s proof from being about GKL(ω, −) to being about FCFω. However, as discussed on page 168, my attempt at such an adjustment failed. In particular, although the function GKL(ω, −) is permutation invariant, the function FCFω is not permutation invariant. I therefore sought a different approach.

The proof of the correctness of the FCF Algorithm requires two steps. The first step, to prove that FCFω has error correction totality, follows directly from the fact that FCFω is a subfunction of FH. The second step, to prove that FCFω has the error correction guarantee, occupies the bulk of this chapter.

The proof presented here of the error correction guarantee of FCFω has a similar core idea to that in the Fellegi-Holt proof of the error correction guarantee of the function FH. The core idea of the Fellegi-Holt proof for FH is a sequence of special edit sets of the form Ωk, with the lifting property. Similarly, the core idea of the proof for FCFω is a tree of special edit sets of the form Γ(i1, ..., ik), where each branch of the tree has the lifting property.

However, the proof of the error correction guarantee of FCFω is much more complex than the proof of the error correction guarantee of FH. The complexity arises because, in the FCF Algorithm, any newly generated edit must be generated from edits above it in the tree according to restrictive rules.

The complexity of the proof of the error correction guarantee of FCFω is manifested in three ways. Firstly, the sets of the form Γ(i1, ..., ik) are arranged in a tree structure, unlike the simpler sets of form Ωk which are arranged as a sequence. Secondly, the definition of the set Γ(i1, ..., ik) depends on the field code forest F(ω), making the properties of Γ(i1, ..., ik) less obvious than those of Ωk. Thirdly, the Domination Rules cause additional complexity, because of the restrictions on edit generation in the FCF Algorithm.

Nevertheless, the proof of the error correction guarantee of FCFω is based on a lifting property in the same way as the proof of the error correction guarantee of FH. Both proofs work through a sequence of edit sets - in the case of FCFω the proof works up one branch of the tree. Both proofs successively “improve” the incorrect record, one field at a time, until finally the record is corrected. The proof that successive improvements exist is done by the same method of generating an edit in one edit set (Γ(i1, ..., ik) or Ωk) from edits in the neighbouring edit set (Γ(i1, ..., ik−1) or Ωk−1).

As de Waal (2003a) points out, the lifting property applies to more situations than the proof of the original function FH. This chapter has given yet another, albeit complicated, use of the lifting property to prove covering set correctibility.
6.11 Appendix 1: The function $\mathcal{FCF}_\omega$

Although this chapter is written in terms of edits, it could equally have been written in terms of logical clauses, using the function $\mathcal{FCF}_\omega$ instead of the function $\text{FCF}_\omega$. It was written in terms of edits in order to be consistent with other literature on the topic.

The function $\mathcal{FCF}_\omega$ is calculated via the $\mathcal{FCF}$ Algorithm, which is identical to the FCF Algorithm, except with edits replaced by normal clauses, dominated edits replaced by subsumed clauses, maximal edits replaced by minimal clauses, and the empty edit replaced by $\top\top$. Since the definitions of $\mathcal{FCF}_\omega$ and $\text{FCF}_\omega$ are essentially identical, we will use the same names for the intermediate functions. That is, we will use the notation GenI, GenV, GenF, and BranchV for both $\mathcal{FCF}_\omega$ and $\text{FCF}_\omega$. A full statement of the $\mathcal{FCF}$ Algorithm is given below.

**Definition 6.11.1 (Statement of the $\mathcal{FCF}$ Algorithm).**

**INPUT to the algorithm:** a set $\Sigma$ of normal clauses on the propositional atoms \(\{p_j^v \mid j = 1, \ldots, N, \quad v \in A_j\}\), and a total ordering $\omega$ of the set \(\{1, \ldots, N\}\).

**STEPS in the algorithm:** Traverse the field code forest $F(\omega)$ in depth-first order. At each node $\sigma$ of $F(\omega)$:

(i) Calculate the set $\text{GenI}(\sigma, \Sigma)$ according to Definition 6.11.3 below.

(ii) Calculate the set $\text{GenV}(\sigma, \Sigma, \sigma)$ using Subsumption Rule 1 given in Definition 6.11.4 below.

(iii) For each node $\tau$ visited before $\sigma$, calculate $\text{GenV}(\tau, \Sigma, \sigma)$ using Subsumption Rule 2 given in Definition 6.11.4 below.

Let $\text{GenF}(\sigma, \Sigma)$ be the set of clauses at node $\sigma$ after the last node $\nu$ of the field code forest has been traversed. That is, $\text{GenF}(\sigma, \Sigma) = \text{GenV}(\sigma, \Sigma, \nu)$.

**OUTPUT of the algorithm:** the set

$$\mathcal{FCF}_\omega(\Sigma) = \bigcup\{\text{GenF}(\sigma, \Sigma) \mid \sigma \text{ is a node of } F(\omega)\}.$$ 

In order to define $\text{GenI}(\sigma, \Sigma)$, we define the set $\text{BranchV}(\sigma, \Sigma, \tau)$ to be the set of clauses appearing in the tree at or above node $\sigma$ just after node $\tau$ has been traversed, as follows:

**Definition 6.11.2.** Suppose $\Sigma$ is a set of normal clauses and $\omega$ is a total ordering of the fields. Suppose also that $\sigma$ and $\tau$ are nodes of the field code forest $F(\omega)$, and that either $\sigma = \tau$ or $\tau$ is traversed after $\sigma$. Write $\sigma = (i_1, \ldots, i_k)$ where $k \geq 0$. Then

$$\text{BranchV}(\sigma, \Sigma, \tau) = \bigcup\{\text{GenV}((i_1, \ldots, i_j), \Sigma, \tau) \mid j = 0, \ldots, k\}.$$ 

We can now define the set $\text{GenI}(\sigma, \Sigma)$.

**Definition 6.11.3 (Definition of $\text{GenI}(\sigma, \Sigma)$ for step (i) of Definition 6.11.1).** Let $\sigma = (i_1, \ldots, i_m) \subseteq \omega$. Define the set $\text{GenI}((i_1, \ldots, i_m), \Sigma)$ inductively as follows:
6.11 Appendix 1: The function $\mathcal{F}_\omega$

a. $\text{GenI}((), \Sigma) = \Sigma$.

b. If $m \geq 1$ and $\text{GenV}((i_1, \ldots, i_{m-1}), \Sigma, (i_1, \ldots, i_{m-1})) = \emptyset$, then

$$\text{GenI}((i_1, \ldots, i_m), \Sigma) = \emptyset.$$  

c. If $m \geq 1$ and $\text{GenV}((i_1, \ldots, i_{m-1}), \Sigma, (i_1, \ldots, i_{m-1})) \neq \emptyset$ then

$$\text{GenI}((i_1, \ldots, i_m), \Sigma) = \left\{ \text{FHD}(i_m, X) \mid X \subseteq \text{BranchV}((i_1, \ldots, i_{m-1}), \Sigma, (i_1, \ldots, i_{m-1})), \text{ and} \right.$$  

$$\text{FHD}(i_m, X) \neq \mathbb{T}, \text{ and} \right.$$  

$$i_m \text{ is not involved in FHD}(i_m, X), \text{ and} \right.$$  

$$i_m \text{ is involved in each element of } X, \text{ and} \right.$$  

$$\text{each of } i_1, \ldots, i_{m-1} \text{ is uninvolved in each element of } X \right\}.$$

Definition 6.11.4 (Subsumption rules for Definition 6.11.1, steps (ii) and (iii)). Let $S_\sigma$ be the set of nodes visited prior to visiting $\sigma$. If $\sigma \neq ()$, then let $\sigma'$ be the node visited immediately before $\sigma$. The subsumption rules, applied after calculating $\text{GenI}(\sigma, \Sigma)$, are:

Subsumption Rule 1. Calculate $\text{GenV}(\sigma, \Sigma, \sigma)$:

(a) If $\sigma = ()$, then remove all subsumed clauses from $\text{GenI}(\sigma, \Sigma)$. That is,

$$\text{GenV}(() , \Sigma , ()) = \text{Min} \circ \text{GenI}((), \Sigma).$$

(b) If $\sigma \neq ()$, then replace each clause in $\text{GenI}(\sigma, \Sigma)$ by all minimal subsuming clauses already generated. That is,

$$\text{GenV}(\sigma, \Sigma, \sigma) = \text{Min} \left( \text{GenI}(\sigma, \Sigma) \cup \left\{ \beta \mid \beta \in \bigcup_{\tau \in S_\sigma} \text{GenV}(\tau, \Sigma, \sigma'), \text{ and there is an } \alpha \text{ in } \text{GenI}(\sigma, \Sigma) \text{ such that } \beta \subset \alpha \right\} \right).$$

Subsumption Rule 2. If $S_\sigma \neq \emptyset$, then for each $\tau$ in $S_\sigma$, calculate $\text{GenV}(\tau, \Sigma, \sigma)$: Replace any already generated clause by all minimal clauses that subsume it and that are in $\text{GenV}(\sigma, \Sigma, \sigma)$. That is,

$$\text{GenV}(\tau, \Sigma, \sigma) = \text{Min} \left( \text{GenV}(\tau, \Sigma, \sigma') \cup \left\{ \beta \mid \beta \in \text{GenV}(\sigma, \Sigma, \sigma), \text{ and there is an } \alpha \text{ in } \text{GenV}(\tau, \Sigma, \sigma') \text{ such that } \beta \subset \alpha \right\} \right).$$
6.12 Appendix 2: Winkler’s counter-example to the FCF Algorithm

This appendix examines the details of Winkler’s counter-example (Winkler 1997) to the permutation invariance of the FCF Algorithm. However, it seems that Winkler has used an interpretation of the FCF Algorithm different from that presented in this thesis in Section 2.6. In this appendix, we recalculate Winkler’s example according to the definition of the FCF Algorithm given in Section 2.6.

Winkler has chosen the example of Garfinkel, Kunnathur, and Liepins (1986b), except with the fields renumbered. Winkler’s example uses 6 fields where the field domains are:

\[
\begin{align*}
A_1 &= A_4 = \{1, 2, 3\} \\
A_2 &= A_6 = \{1, 2, 3, 4\} \\
A_3 &= A_5 = \{1, 2\}.
\end{align*}
\]

The edit set \( E = \{E^1, E^2, E^3, E^4, E^5\} \) where

\[
\begin{align*}
E^1 &= \{1, 2\} \times A_2 \times A_3 \times \{1, 2\} \times \{1\} \times A_6 \\
E^2 &= A_1 \times \{3, 4\} \times \{2\} \times A_4 \times \{2\} \times \{1, 2\} \\
E^3 &= A_1 \times A_2 \times \{1\} \times \{2, 3\} \times A_5 \times \{2, 3, 4\} \\
E^4 &= A_1 \times \{1, 2\} \times A_3 \times \{1, 3\} \times A_5 \times A_6 \\
E^5 &= \{2, 3\} \times A_2 \times \{2\} \times A_4 \times A_5 \times \{1\}.
\end{align*}
\]

Winkler considers two field orders: \( \omega = (1, 2, 3, 4, 5, 6) \) and \( \delta(\omega) = (3, 4, 5, 6, 1, 2) \). The latter order is equivalent to the original example of G, K & L.

Tables 6.1 and 6.2 give calculations of \( \text{FCF}_\omega(E) \) and \( \text{FCF}_{\delta(\omega)}(E) \) respectively and demonstrate that \( \text{FCF}_\omega(E) = \text{FCF}_{\delta(\omega)}(E) \). Note that in this example there are no dominated edits in the set of explicit and generated edits, and so the calculations are simplified.

Winkler’s calculations give the result that \( \text{FCF}_\omega(E) \neq \text{FCF}_{\delta(\omega)}(E) \). He omits the edit \( E^9 \) from the set \( \text{FCF}_\omega(E) \), but otherwise his calculations are the same as those in Tables 6.1 and 6.2.

Winkler seems to have omitted edit \( E^9 \) from \( \text{FCF}_\omega(E) \) because of his interpretation of the FCF Algorithm. He seems to require that any newly generated edit must use at least one edit from the predecessor node. Therefore, when using field order \( \omega \), there are no edits generated at node (1, 2), because edit \( E^{10} \) depends purely on edits from node (). Thus node (1, 2, 5) is never reached and edit \( E^9 \) is never generated. The edit \( E^{10} \), however, is also generated at node (2).

Winkler concludes that the FCF Algorithm is not permutation invariant. While it is true that the FCF Algorithm lacks permutation invariance (as shown in Example 6.2.7), Winkler’s example itself does not demonstrate this lack.

Since Garfinkel, Kunnathur and Liepins used permutation invariance in their proof, Winkler concludes that the FCF Algorithm is incorrect and proposes a variation. Luckily the variation is not needed, as this chapter has shown that the FCF Algorithm is correct in spite of its lack of permutation invariance.
§6.12 Appendix 2: Winkler’s counter-example to the FCF Algorithm 205

Table 6.1: Calculation of FCF\(\omega(E)\) where \(\omega = (1, 2, 3, 4, 5, 6)\)

<table>
<thead>
<tr>
<th>Node</th>
<th>Formula</th>
<th>Edit</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>Explicit edit</td>
<td>(E^1 = {1, 2} \times A_2 \times A_3 \times {1, 2} \times {1} \times A_6)</td>
</tr>
<tr>
<td>()</td>
<td>Explicit edit</td>
<td>(E^2 = A_1 \times {3, 4} \times {2} \times A_4 \times {2} \times {1, 2})</td>
</tr>
<tr>
<td>()</td>
<td>Explicit edit</td>
<td>(E^3 = A_1 \times A_2 \times {1} \times {2, 3} \times A_5 \times {2, 3, 4})</td>
</tr>
<tr>
<td>()</td>
<td>Explicit edit</td>
<td>(E^4 = A_1 \times {1, 2} \times A_3 \times {1, 3} \times A_5 \times A_6)</td>
</tr>
<tr>
<td>()</td>
<td>Explicit edit</td>
<td>(E^5 = {2, 3} \times A_2 \times {2} \times A_4 \times A_5 \times {1})</td>
</tr>
</tbody>
</table>

(1) \(\text{FHG}(1, \{E^1, E^5\}) = E^6 = A_1 \times A_2 \times \{2\} \times \{1, 2\} \times \{1\} \times \{1\}\)

(1, 2) \(\text{FHG}(2, \{E^2, E^4\}) = E^{10} = A_1 \times A_2 \times \{2\} \times \{1, 2\} \times \{1\} \times \{2\}\)

(1, 2, 3) \(\text{FHG}(3, \{E^3, E^{10}\}) = E^{11} = A_1 \times A_2 \times A_3 \times \{3\} \times \{2\} \times \{2\}\)

(1, 2, 5) \(\text{FHG}(5, \{E^6, E^{10}\}) = E^9 = A_1 \times A_2 \times \{2\} \times \{1\} \times A_5 \times \{1\}\)

(1, 3) \(\text{FHG}(3, \{E^2, E^3\}) = E^{13} = A_1 \times \{3, 4\} \times A_3 \times \{2, 3\} \times \{2\} \times \{2\}\)

(1, 4) \(\text{FHG}(4, \{E^3, E^4\}) = E^{17} = A_1 \times \{1, 2\} \times \{1\} \times A_4 \times A_5 \times \{2, 3, 4\}\)

(1, 4) \(\text{FHG}(4, \{E^4, E^6\}) = E^7 = A_1 \times \{1, 2\} \times \{2\} \times A_4 \times \{1\} \times \{1\}\)

(1, 5) \(\text{FHG}(5, \{E^2, E^6\}) = E^8 = A_1 \times \{3, 4\} \times \{2\} \times \{1, 2\} \times A_5 \times \{1\}\)

(2) \(\text{FHG}(2, \{E^2, E^4\}) = E^{10} = A_1 \times A_2 \times \{2\} \times \{1, 2\} \times \{1\} \times \{2\}\)

(2, 3) \(\text{FHG}(3, \{E^3, E^{10}\}) = E^{11} = A_1 \times A_2 \times A_3 \times \{3\} \times \{2\} \times \{2\}\)

(2, 4) \(\text{FHG}(4, \{E^1, E^3\}) = E^{15} = \{1, 2\} \times A_2 \times \{1\} \times A_4 \times \{1\} \times \{2, 3, 4\}\)

(2, 5) \(\text{FHG}(5, \{E^1, E^{10}\}) = E^{12} = \{1, 2\} \times A_2 \times \{2\} \times \{1\} \times A_5 \times \{1\} \times \{2\}\)

(3) \(\text{FHG}(3, \{E^2, E^3\}) = E^{13} = A_1 \times \{3, 4\} \times A_3 \times \{2, 3\} \times \{2\} \times \{2\}\)

(3, 4) \(\text{FHG}(4, \{E^1, E^4\}) = E^{16} = \{1, 2\} \times \{1, 2\} \times A_3 \times \{1\} \times A_6\)

(3, 5) \(\text{FHG}(5, \{E^1, E^{13}\}) = E^{14} = \{1, 2\} \times \{3, 4\} \times A_3 \times \{2\} \times A_5 \times \{2\}\)

(4) \(\text{FHG}(4, \{E^1, E^3\}) = E^{15} = \{1, 2\} \times A_2 \times \{1\} \times A_4 \times \{1\} \times \{2, 3, 4\}\)

(4) \(\text{FHG}(4, \{E^1, E^4\}) = E^{16} = \{1, 2\} \times \{1, 2\} \times A_3 \times \{1\} \times A_6\)

(4) \(\text{FHG}(4, \{E^3, E^4\}) = E^{17} = A_1 \times \{1, 2\} \times \{1\} \times A_4 \times A_5 \times \{2, 3, 4\}\)

(5) \(\text{FHG}(5, \{E^1, E^2\}) = E^{18} = \{1, 2\} \times \{3, 4\} \times \{2\} \times \{1, 2\} \times A_5 \times \{1, 2\}\)
Table 6.2: Calculation of $\text{FCF}_{\delta(\omega)}(E)$ where $\delta(\omega) = (3, 4, 5, 6, 1, 2)$

<table>
<thead>
<tr>
<th>Node</th>
<th>Formula</th>
<th>Edit</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>Explicit edit</td>
<td>$E^1 = {1, 2} \times A_2 \times A_3 \times {1, 2} \times {1} \times A_6$</td>
</tr>
<tr>
<td>(2)</td>
<td>Explicit edit</td>
<td>$E^2 = A_1 \times {3, 4} \times {2} \times A_4 \times {2} \times {1, 2}$</td>
</tr>
<tr>
<td>(3)</td>
<td>Explicit edit</td>
<td>$E^3 = A_1 \times A_2 \times {1} \times {2, 3} \times A_5 \times {2, 3, 4}$</td>
</tr>
<tr>
<td>(4)</td>
<td>Explicit edit</td>
<td>$E^4 = A_1 \times {1, 2} \times A_3 \times {1, 3} \times A_5 \times A_6$</td>
</tr>
<tr>
<td>(5)</td>
<td>Explicit edit</td>
<td>$E^5 = {2, 3} \times A_2 \times {2} \times A_4 \times A_5 \times {1}$</td>
</tr>
<tr>
<td>(3, 1)</td>
<td>$\text{FHG}(3, {E^2, E^3}) = E^{13} = A_1 \times {3, 4} \times A_3 \times {2, 3} \times {2} \times {2}$</td>
<td></td>
</tr>
<tr>
<td>(3, 4)</td>
<td>$\text{FHG}(4, {E^1, E^4}) = E^{16} = {1, 2} \times {1, 2} \times A_3 \times A_4 \times {1} \times A_6$</td>
<td></td>
</tr>
<tr>
<td>(3, 5)</td>
<td>$\text{FHG}(5, {E^1, E^{13}}) = E^{14} = {1, 2} \times {3, 4} \times A_3 \times {2} \times A_5 \times {2}$</td>
<td></td>
</tr>
<tr>
<td>(3, 2)</td>
<td>$\text{FHG}(2, {E^4, E^{13}}) = E^{11} = A_1 \times A_2 \times A_3 \times {3} \times {2} \times {2}$</td>
<td></td>
</tr>
<tr>
<td>(4)</td>
<td>$\text{FHG}(4, {E^1, E^{3}}) = E^{15} = {1, 2} \times A_2 \times {1} \times A_4 \times {1} \times {2, 3, 4}$</td>
<td></td>
</tr>
<tr>
<td>(4)</td>
<td>$\text{FHG}(4, {E^3, E^{4}}) = E^{16} = {1, 2} \times {1, 2} \times A_3 \times A_4 \times {1} \times A_6$</td>
<td></td>
</tr>
<tr>
<td>(4, 1)</td>
<td>$\text{FHG}(1, {E^5, E^{16}}) = E^{7} = A_1 \times {1, 2} \times {2} \times A_4 \times {1} \times {1}$</td>
<td></td>
</tr>
<tr>
<td>(5)</td>
<td>$\text{FHG}(5, {E^1, E^{2}}) = E^{18} = {1, 2} \times {3, 4} \times {2} \times {1, 2} \times A_5 \times {1} \times A_2$</td>
<td></td>
</tr>
<tr>
<td>(5, 1)</td>
<td>$\text{FHG}(1, {E^{5}, E^{18}}) = E^{8} = A_1 \times {3, 4} \times {2} \times {1, 2} \times A_5 \times {1}$</td>
<td></td>
</tr>
<tr>
<td>(5, 1, 2)</td>
<td>$\text{FHG}(2, {E^4, E^{5}}) = E^{9} = A_1 \times A_2 \times {2} \times {1} \times A_5 \times {1}$</td>
<td></td>
</tr>
<tr>
<td>(5, 2)</td>
<td>$\text{FHG}(2, {E^{4}, E^{18}}) = E^{12} = {1, 2} \times A_2 \times {2} \times {1} \times A_5 \times {1, 2}$</td>
<td></td>
</tr>
<tr>
<td>(1)</td>
<td>$\text{FHG}(1, {E^1, E^{5}}) = E^{6} = A_1 \times A_2 \times {2} \times {1, 2} \times {1} \times {1}$</td>
<td></td>
</tr>
<tr>
<td>(1, 2)</td>
<td>$\text{FHG}(2, {E^2, E^{1}}) = E^{10} = A_1 \times A_2 \times {2} \times {1, 3} \times {2} \times {1, 2}$</td>
<td></td>
</tr>
<tr>
<td>(2)</td>
<td>$\text{FHG}(2, {E^2, E^{1}}) = E^{10} = A_1 \times A_2 \times {2} \times {1, 3} \times {2} \times {1, 2}$</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 7

Arithmetic edits

7.1 Introduction

So far in this thesis we have concentrated on categorical edits, which deal with discrete data. Examples of discrete data fields include marital status, and whether or not someone has a driver’s licence. Often numerical data can be discretised while still retaining the edits, such as the field “profit” in the edit \( \{ \text{profit} \mid \text{profit} > 0 \} \times \{ \text{income} \mid \text{income} = 0 \} \). However in some cases it is impractical to discretise numerical data, for example in the edit requiring that profit be greater than or equal to income.

This chapter is about edits on numerical data that are not discretised. In particular we will deal with non-strict linear inequalities which we will call arithmetic edits (defined precisely in Section 7.2). A typical example is the above edit that profit be greater than or equal to income, which we will write as \( x_1 - x_2 \geq 0 \) where the variables \( x_1 \) and \( x_2 \) represent the two fields. We will not deal with the more restricted form of arithmetic edits where the field values are restricted to integers.

The main purpose of this chapter is to give a sketch of a logical formalisation for arithmetic edits and for “covering set correctibility” for arithmetic edits. This paper extends the work of Fellegi and Holt (1976) who defined an edit generation function (to be called FM) for arithmetic edits that parallels the function FH, and also gave most of the proof (fleshed out by de Waal (2003a)) that FM has a property parallel to the error correction guarantee. A previous paper (Boskovitz, Goré, and Wong 2005) presents a summary of the main results presented here.

My work in this chapter answers two questions: (1) do the results presented in Chapters 2 and 4 for categorical edits transfer completely to arithmetic edits? and (2) in a logical formalisation, do the results presented in Chapter 5 about covering set correctibility for categorical edits transfer to arithmetic edits? The answers to both questions are yes, although the explanations depend on definitions and proofs that are not always direct translations from the categorical case.

In order to do the formalisation, we follow the program of Chapters 2, 4 and 5, and obtain results parallel to those obtained for categorical edits. We first find, in Section 7.2, that covering set correctibility can be defined for arithmetic edits much like it was defined for categorical edits in Chapter 2. Then, in Section 7.3, we follow the program of Chapter 4 and formalise arithmetic edits in terms of propositional logic. The formalisation looks very different from that for categorical edits but we can still
formalise covering set correctibility for arithmetic edits in much the same way as it was for categorical edits in Chapter 5, Section 5.2. In addition, in Section 7.4, we obtain a statement of covering set correctibility for arithmetic edits in terms of some natural constructs of logic, in results that are analogous to the corresponding results for categorical edits (Chapter 5, Section 5.3). Finally, in Section 7.5, we find that covering set correctibility for arithmetic edits is an extension of refutation completeness and soundness, analogously to the material about categorical edits in Section 5.4 of Chapter 5.

There are some differences between the formalisations for arithmetic edits and for categorical edits. Firstly, the arithmetic edits themselves are formalised differently from categorical edits. Secondly, the definition of “reduction” (Definition 7.4.1) for arithmetic edits does not at first sight appear analogous to the definition (Definition 5.3.1) for categorical edits. Thirdly, although many of the proofs are similar to the corresponding proofs for categorical edits, there are some that appear quite different, for example, the proof that strong completeness implies the error correction guarantee (Proposition 7.5.5 for arithmetic edits and Proposition 5.4.2 for categorical edits).

The work presented in this chapter is rigorous but does have some gaps. However I felt it worthwhile to include it in the thesis, in order to save others some work. Some of the gaps and obvious extensions are discussed briefly towards the end of the chapter, in Section 7.6.

One gap in this chapter is that not every type of linear arithmetic edit is considered. In particular, we will consider only those arithmetic edits that are expressible as single linear non-strict inequalities, using the sign $\geq$. We will not consider disjunctions of linear inequalities such as “$1 - 2x_1 + x_2 \geq 0$ OR $x_1 + 4x_3 \geq 0$”. Nor will we consider combinations of arithmetic and categorical edits, such as “$1 - 2x_1 + x_2 \geq 0$ AND has driver’s licence = yes”. We do however briefly discuss the implications of considering such more complex edits, in Section 7.6.4.

The chapter makes some use of vector notation. As in previous chapters, all vectors are represented in bold form, such as $\mathbf{v}$. Such notation is taken to represent a row vector. Column vectors are represented with the transpose notation, for example $\mathbf{v}^T$.

### 7.2 Arithmetic edits using sets

This section introduces the main definitions used for arithmetic edits in terms of sets. It follows the definitions given in Chapter 2 for categorical edits. The differences are in the definition of edits themselves and in the definition of the word “involved”. The error localisation problem is defined in the same way as for categorical edits. One solution to the error localisation problem is in terms of covering sets. The terms “covering set correctible”, “error correction guarantee”, and “error correction totality” can be used in the same way as for categorical edits to mean that the covering set method produces various types of useful results.

Records, fields, field domains, and the data domain are defined as for categorical edits (Definition 2.2.1), that is
1. A record $v$ is an $N$-tuple $(v_1, \ldots, v_N)$ where $N$ is a positive integer which is the same for all records;

2. the values $j = 1, \ldots, N$ are called fields;

3. for each $j = 1, \ldots, N$, the value $v_j \in A_j$, where each $A_j$ is a set called the $j$th field domain;

4. the set of all possible records is the data domain $D = A_1 \times \cdots \times A_N$ (also called the domain).

However, unlike for categorical edits, the field domains for arithmetic edits are restricted to subsets of the set of rational numbers $\mathbb{Q}$, that is, for each $j = 1, \ldots, N$, the set $A_j \subseteq \mathbb{Q}$.

Each arithmetic edit, similarly to each categorical edit, specifies a set of acceptable and unacceptable records. But instead of listing the set of unacceptable records, each arithmetic edit gives the formula which acceptable records must satisfy.

We will consider only those arithmetic edits that can be specified as linear inequalities using the sign $\geq$, of the form $e_0 + \sum_{j=1}^N e_j x_j \geq 0$, where $e_0, \ldots, e_N$ are rationals. A record $v = (v_1, \ldots, v_N)$ satisfies this edit if and only if $e_0 + \sum_{j=1}^N e_j v_j \geq 0$. Note that we have recycled some notation: previously, for the categorical case, we used notation such as $e_j$ for the edits themselves, but here such notation represents rational numbers.

The variables $x_j$ in the above arithmetic edit are merely placeholders. Therefore, instead of writing each arithmetic edit as an inequality, we will write it as the vector of its coefficients, as in Definition 7.2.1 below.

**Definition 7.2.1.** An arithmetic edit is an $(N+1)$-tuple $e = (e_0, \ldots, e_N)$, where $e_0, \ldots, e_N$ are rational numbers. The arithmetic edit $e$ represents the linear inequality

$$e_0 + \sum_{j=1}^N e_j x_j \geq 0,$$

or equivalently, with $x = (x_1, \ldots, x_N)$:

$$e \begin{pmatrix} 1 \\ x^T \end{pmatrix} \geq 0.$$

A record $v = (v_1, \ldots, v_N)$ is correct with respect to the arithmetic edit $e$ if $e_0 + \sum_{j=1}^N e_j v_j \geq 0$. We also say that $v$ satisfies $e$. The record $v$ is incorrect with respect to the arithmetic edit $e$, or fails $e$, if $e_0 + \sum_{j=1}^N e_j v_j < 0$.

The record $v$ satisfies a set $E$ of edits if $v$ satisfies all edits in $E$. The record $v$ fails a set $E$ of arithmetic edits if $v$ fails some edit in $E$. 
We represent a set $E$ of $m$ arithmetic edits (ambiguously but comprehensibly) in two equivalent ways:

1. using set notation $\{e^1, \ldots, e^m\}$, where for each $j = 1, \ldots, m$ the edit $e^j = (e^j_0, \ldots, e^j_N)$; or

2. as an $m \times (N+1)$ matrix

$$
\begin{pmatrix}
  e^1_0 \cdots e^1_N \\
  \vdots \\
  e^m_0 \cdots e^m_N 
\end{pmatrix},
$$

each row of which is an edit.

Note 1: The set of all arithmetic edits is $\mathbb{Q}^{N+1}$, representing the set of all possible linear inequalities using $\geq$ on $N$ variables.

Note 2: If the edit $e = (e_0, 0, \ldots, 0)$, that is $e_j = 0$ for all $j = 1, \ldots, N$, then the edit is either satisfied by all records (when $e_0 \geq 0$) or failed by all records (when $e_0 < 0$).

Note 3: In a previous paper (Boskovitz, Góre, and Wong 2005), arithmetic edits are written as inequalities rather than as $(N+1)$-tuples.

Example 7.2.2. A table of numerical data has three fields, and $x_1$, $x_2$ and $x_3$ are variables representing the field values. The domain of each field is the set of rational numbers. The edit set $EE$ contains two edits:

- $ea = (1, -2, 1, 0)$ representing $1 - 2x_1 + x_2 \geq 0$, and
- $eb = (0, 1, 0, 4)$ representing $x_1 + 4x_3 \geq 0$.

Suppose that $vv = (2, 1, -1)$. Since $1 - 2 \times 2 + -1 < 0$, the record $vv$ fails $ea$. Similarly $vv$ fails $eb$, while the record $ww = (2, 1, 3)$ fails the edit $ea$ and satisfies $eb$. The edit set $EE$ can be written as $\{ea, eb\}$ or as

$$
EE = \begin{pmatrix}
  1 & -2 & 1 & 0 \\
  0 & 1 & 0 & 4 
\end{pmatrix}.
$$

For arithmetic edits, the definition of the error localisation problem, the definition of the term “yields a correction”, and the notation $EL$ are the same as for categorical edits, as given in Definition 2.2.8. That is:

If $v = (v_1, \ldots, v_N)$ is a record and $E$ is a set of arithmetic edits, then the **error localisation problem** is the problem of deciding which sets of fields can be changed to correct $v$ with respect to $E$. That is, it is the following problem: given a set $C$ of fields and a record $v$, decide whether there is a record $w = (w_1, \ldots, w_N)$ such that

- a. $C \supseteq \{j \mid w_j \neq v_j\}$, and
- b. $w$ satisfies $E$.

If there exists a record $w$ such that a and b are satisfied, then we say that **the set $C$ yields a correction $w$ to $v$ with respect to $E$** and we shall...
also say that $C$ is a solution to the error localisation problem for $v$ and $E$.

The set of solutions to the error localisation problem for the record $v$ and the set $E$ of arithmetic edits will be written $\mathcal{EL}(E,v)$. This means that $C$ yields a correction to $v$ with respect to $E$ if and only if $C \in \mathcal{EL}(E,v)$.

We can try to solve the error localisation problem for arithmetic edits by way of covering sets, just as we did for categorical edits. In order to use the same solution technique, we will make definitions parallel to those used for categorical edits: we define covering sets in terms of involved fields, and the notations $X$ and $CX$ to represent failed arithmetic edits and their covering sets respectively.

Covering sets, and the notations $X$ and $CX$, are defined for arithmetic edits in terms of involved fields in the same way as for categorical edits, and are given below. However, the term “involved” for arithmetic edits is defined differently from the definition for categorical edits, and is given next. After the four definitions, there follows an example.

**Definition 7.2.3.** The field $k \ (k \in \{1, \ldots, N\})$ is **involved** in the arithmetic edit $e = (e_0, \ldots, e_N)$ if $e_k \neq 0$. The field $k$ is **uninvolved** in the arithmetic edit $e$ if $e_k = 0$.

We can now define covering sets in terms of involved fields in the same way as Definition 2.3.3 for categorical edits.

**Definition 7.2.4.** A **covering set** $C$ of the set $E$ of arithmetic edits is a set of fields such that for each edit $e$ in $E$, there exists a field in $C$ that is involved in $e$. We write $C(E)$ for the set of all covering sets of $E$:

$$C(E) = \left\{ C \subseteq \{1, \ldots, N\} \ \middle| \ C \text{ is a covering set of } E \right\}.$$ 

As we were for categorical edits, we are especially interested in finding a covering set of the edits of $E$ failed by $v$. As in Definitions 2.3.5 and 2.3.6 for categorical edits, we use the notation $X$ and $CX$ to represent failed edits and their covering sets respectively:

**Definition 7.2.5.** Define the set $X(E,v)$ to be the set of edits in the set $E$ that are failed by $v$. That is, $X(E,v) = \left\{ e \in E \ \middle| \ e \left( \frac{1}{v^T} \right) < 0 \right\}$.

**Definition 7.2.6.** Define $CX(E,v)$ to be the set of covering sets of the edits in the set $E$ that are failed by the record $v$. That is $CX(E,v) = C(X(E,v))$.

**Example 7.2.7.** As in Example 7.2.2, suppose that $EE = \{ea, eb\}$, where

$$ea = (1, -2, 1, 0) \quad \text{and} \quad eb = (0, 1, 0, 4),$$

and that the record $vv = (2, 1, -1)$ and that the record $ww = (2, 1, 3)$. 

The arithmetic edit $ea$ has fields 1 and 2 involved and field 3 uninvolved. The arithmetic edit $eb$ has fields 1 and 3 involved and field 2 uninvolved.

The field sets $\{1\}$ and $\{2,3\}$ are covering sets of $EE$. That is $\{1\} \in C(EE)$ and $\{2,3\} \in C(EE)$.

Since $vv$ fails both edits in $EE$, we have that $\chi(EE,vv) = EE$. Hence $C\chi(EE,vv) = C(EE)$.

Since $ww$ fails $ea$ and satisfies $eb$, we have that $\chi(EE,ww) = \{ea\}$. Elements of $C\chi(EE,ww)$ include $\{1\}$ and $\{2\}$.

Just as we sought to solve the error localisation problem for categorical edits using edit generation functions (Definition 2.3.13), so we seek to solve the error localisation problem for arithmetic edits using arithmetic edit generation functions, defined below.

**Definition 7.2.8.** An arithmetic edit generation function is a function from arithmetic edit sets to arithmetic edit sets, that is, a function $G$ where

$$G : \mathcal{P}(\mathbb{Q}^{N+1}) \rightarrow \mathcal{P}(\mathbb{Q}^{N+1}).$$

We define below for arithmetic edits the terms error correction totality, error correction guarantee, and covering set correctibility. The definitions follow the definitions for categorical edits (Definition 2.3.14 and 2.3.16).

**Definition 7.2.9.** The arithmetic edit generation function $G$ has error correction totality if for any set $E$ of arithmetic edits and any record $v$,

$$E\mathcal{L}(E,v) \subseteq C\chi(G(E),v).$$

The arithmetic edit generation function $G$ has the error correction guarantee if for any set $E$ of arithmetic edits and any record $v$,

$$C\chi(G(E),v) \subseteq E\mathcal{L}(E,v).$$

The arithmetic edit generation function $G$ is covering set correctible if for any set $E$ of arithmetic edits and any record $v$,

$$C\chi(G(E),v) = E\mathcal{L}(E,v).$$

We consider two examples, one of an edit generation function that does not have the error correction guarantee, and one of an edit generation function that is covering set correctible. The first example is of the simplest edit generation function, namely the identity function, which does not have the error correction guarantee.

**Example 7.2.10.** The identity edit generation function $I$ has $I(E) = E$ for each edit set $E$. The function $I$ does not have the error correction guarantee. For example, as in the previous examples, suppose that $EE = \{ea, eb\}$ where

$$ea = (1, -2, 1, 0) \quad \text{and} \quad eb = (0, 1, 0, 4),$$
and the record \( \nu \nu = (2, 1, -1) \). Then \( \mathcal{X}(I(EE), \nu \nu) = EE \) and \( \{1\} \in \mathcal{C} \mathcal{X}(I(EE), \nu \nu) \). However it is not possible to satisfy both edits by changing only the value of field 1, because in that case field 1 must satisfy \( 1 - 2x_1 + 1 \geq 0 \) and \( x_1 - 4 \geq 0 \) which are incompatible. Hence \( \{1\} \notin \mathcal{E} \mathcal{L}(EE, \nu \nu) \).

The second example is based on the Fourier-Motzkin elimination method (Fourier 1823, 1824, 1826; Motzkin et al. 1953). The Fourier-Motzkin elimination method takes as input a set of linear inequalities and an ordering of the variables, then eliminates the variables in order by taking positive linear combinations of pairs of inequalities, thereby returning a derived set of linear inequalities. The function \( \text{FM} \), defined formally below, differs in two details from the Fourier-Motzkin elimination method: firstly, the variables are not ordered in a sequence; and secondly, linear combinations are allowed of sets of inequalities rather than just pairs of inequalities.

**Definition 7.2.11.** Let \( E \) be an arithmetic edit set, where

\[
E = \begin{pmatrix}
e_0^1 & \cdots & e_N^1 \\
\vdots & \ddots & \vdots \\
e_0^m & \cdots & e_N^m
\end{pmatrix}
\]

For each \( j \in \{0, \ldots, N\} \) we write \( E_j \) as the \( j \) column of \( E \), that is

\[
E_j = \begin{pmatrix}
e_0^j \\
\vdots \\
e_N^j
\end{pmatrix}
\]

Then define \( \text{FM}(E) \) to be the set of those non-negative linear combinations of the edits of \( E \) where at least one variable is eliminated, excluding the edit 0. That is,

\[
\text{FM}(E) = \left\{ \lambda E \mid \lambda = (\lambda_1, \ldots, \lambda_m) \geq 0, \lambda \neq 0, \text{ and there is an } i \in \{1, \ldots, N\}, \right. \text{ and a } k \in \{1, \ldots, m\} \text{ such that } \lambda_k \neq 0 \text{ and } e_i^k \neq 0 \text{ and } \lambda E_i = 0 \}
\]

The function \( \text{FM} \) is covering set correctible. The proof has two steps: the proof of error correction totality and the proof of the error correction guarantee. The proof of the error correction guarantee is the more complex and depends on a lifting property (Definition 2.4.5), much like that used for the proof of the error correction guarantee of the function \( \text{FH} \).

**Theorem 7.2.12.** The function \( \text{FM} \) is covering set correctible.

**Proof.**

**Error correction totality.** I give a direct proof here, although the error correction totality of \( \text{FM} \) follows from the strong soundness of its logical formalisation, \( \mathcal{F} \mathcal{M} \), to be introduced in Definition 7.3.15, and from the fact that strong soundness is equivalent to error correction totality, to be given in Proposition 7.5.1. Let \( E \) be an arithmetic edit set, and let \( \nu = (v_1, \ldots, v_N) \) be a record and suppose that \( C \in \mathcal{E} \mathcal{L}(E, \nu) \). We will
show that \( C \in C(X(FM(E), v)) \), by showing that each edit \( e \) in \( X(FM(E), v) \) has an involved field in \( C \). Firstly, since \( C \in E(L(E, v)) \), there is a record \( w = (w_1, \ldots, w_N) \) such that (a) if \( j \notin C \) then \( w_j = v_j \), and (b) \( w \) satisfies \( E \). Now suppose that \( e = (e_1, \ldots, e_N) \in X(FM(E), v) \). Then \( e(w^T) < 0 \). On the other hand, since \( w \) satisfies \( E \), we have that \( e(\frac{1}{w^T}) \geq 0 \). Combining the two inequalities gives us that \( e(w^T - v^T) > 0 \). Since \( w_j = v_j \) when \( j \notin C \), the last inequality implies that \( \sum_{j \in C} e_j (w_j - v_j) \neq 0 \). Hence there is a \( j \) in \( C \) such that \( e_j \neq 0 \), so that \( e \) has an involved field in \( C \), as required.

Error correction guarantee. De Waal (2003a, page 47) gives an explanation of why the function \( FM \) has the error correction guarantee. Similarly to the proof of the error correction guarantee of the function \( FH \), the proof of the error correction guarantee of the function \( FM \) uses three steps: a lifting property, a repeated lifting property, and finally the error correction guarantee itself.

The first step is the lifting property of the sequence \((E^j)_{j=0}^N\) of edit sets, defined formally below in terms of a given arithmetic edit set \( E \). Note that \( E^j \) is different from \( E_j \) used in Definition 7.2.11. Informally, we start with the set \( E^0 = E \) and then successively eliminate the variables \( x_1, \ldots, x_N \) from all possible pairs of inequalities represented by the edits. That is, \( E^j \) is obtained by eliminating the variable \( x_j \) from all possible pairs of inequalities represented by \( E^{j-1} \); the set \( E^j \) also contains any elements of \( E^{j-1} \) that do not involve the field \( j \). The formal definition is given inductively for a given arithmetic edit set \( E \):

\[
E^0 = E;
\]

if \( j \in \{1, \ldots, N\} \) then \( E^j = \{ e \mid \text{there are edits } e' = (e'_1, \ldots, e'_N) \text{ and } e'' = (e''_1, \ldots, e''_N) \text{ in } E^{j-1} \text{ such that } e'_j e''_j < 0 \text{ and } e = |e''_j| e' + |e'_j| e'' \} \cup \{ e \in E^{j-1} \mid \text{field } j \text{ is uninvolved in } e \} \).

Note that for each \( j \in \{0, \ldots, N\} \), the set \( E^j \subseteq FM(E) \). Also note that each edit in \( E^j \) has the fields \( 1, \ldots, j \) uninvolved.

The lifting property states that if \( 1 \leq k \leq N \) and the record \( v \) satisfies all edits in \( E^k \), then \( \{k\} \in E(L(E^{k-1}, v)) \). The proof has been published many times, including by Fourier (1824) who refers on page 327 of *Œuvres II* to a proof in an earlier “Mémoire”. Later proofs were provided by Dines (1919) who incorporated it on pages 193–196 into the proof of his Theorem II; by Kuhn (1956) who incorporated it on pages 227–228 into the proof of his Theorem III; by Duffin (1974, Lemma 1, page 81); and by Fellegi and Holt (1976, Theorem 3, pages 29–30, 34–35).

The second step is the repeated lifting property: if \( 0 \leq k \leq N \) and the record \( v \) satisfies all edits in \( E^k \) then \( \{1, \ldots, k\} \in E(L(E, v)) \). The repeated lifting property follows from repeated application of the lifting property.
The final step is the error correction guarantee of FM. To prove it, suppose that $C \in C\mathcal{X}(FM(E), v)$. If $C = \emptyset$, then $v$ satisfies $FM(E)$ and hence also satisfies $E$, so that $C \in E\mathcal{L}(E,v)$, as required. From now on, suppose that $C \neq \emptyset$, and renumber the fields so that $C = \{1, \ldots, k\}$, where $k \geq 1$. Then $v$ satisfies all edits of $E^k$ by the following argument.

Suppose, on the contrary, that there is an edit $e$ in $E^k$ that is failed by $v$. Then, since $E^k \subseteq FM(E)$, we have that $e \in \mathcal{X}(FM(E), v)$. Hence, by the definition of covering sets, $e$ has an involved field in the covering set $\{1, \ldots, k\}$. But, by the definition of $E^k$, each of the fields $1, \ldots, k$ is uninvolved in each edit of $E^k$, so no such $e$ can exist and $v$ cannot fail any edit of $E^k$.

Since $v$ satisfies all edits in $E^k$, the repeated lifting property tells us that $C = \{1, \ldots, k\} \in E\mathcal{L}(E,v)$, giving us the error correction guarantee of FM. $\dashv$

The following is an example demonstrating the error correction guarantee of FM.

**Example 7.2.13.** Suppose, as in the previous examples, that $EE = \{ea, eb\}$ where

$ea = (1, -2, 1, 0)$ representing $1 - 2x_1 + x_2 \geq 0$, and

$eb = (0, 1, 0, 4)$ representing $x_1 + 4x_3 \geq 0$.

We pre-multiply the matrix $EE$ by $(1, 2)$ to obtain the edit $ec$ of $FM(EE)$:

$ec = (1, 2)EE = (1, 0, 1, 8)$ representing $1 + x_2 + 8x_3 \geq 0$.

Then

$FM(EE) = \{\lambda ea, \lambda eb, \lambda ec | \lambda \in \mathbb{Q}, \lambda > 0\}$.

The record $vv = (2, 1, -1)$ fails all edits of $FM(EE)$ and hence we have that

$\mathcal{X}(FM(EE), vv) = FM(EE)$,

one of whose covering sets is $\{2, 3\}$, which yields a correction to $vv$, for example by changing the value of field 2 to 3 and the value of field 3 to 0, giving a possible corrected record as $(1, 3, 0)$.

Just as the function $FH$ could be modified to create another covering set correctible function $FCF_\omega$, so the function $FM$ can also be modified using a field code forest. Further discussion is provided later, in Section 7.6.3.

This section has given the basic definitions relevant to arithmetic edits. The definitions are the same as for categorical edits except that:

1. edits themselves are represented as $(N + 1)$-tuples which in turn represent linear inequalities in $N$ variables;
2. an involved field of an edit is one whose coefficient is non-zero in that edit; and
3. edit generation functions are defined on the set $P(\mathbb{Q}^{N+1})$. 

\section*{§7.2 Arithmetic edits using sets}
7.3 Logical formalisation of arithmetic edits

We now formalise arithmetic edits and related concepts in propositional logic. Our formalisation follows the same steps as for categorical edits. We formalise edits, records, correctness of records, error localisation, yielding a correction, involved fields, covering sets, and edit generation.

At first sight it might seem that arithmetic edits should be formalised using first-order logic, because of the use of variables in the inequalities represented by arithmetic edits. The strength of first-order logic is that it provides a method of representing relationships between different records. However, in the examples of data editing considered here, each record is treated individually and the relationships between records are not considered. We therefore use propositional logic to formalise arithmetic edits.

We will formalise each arithmetic edit $e$ as a propositional atom, as done in many approaches to solving systems of linear and Boolean constraints, for example by Aude-mard et al. (2002) and by Bozzano et al. (2005). We will write $q_e$ as the propositional atom formalising $e$. The representation of propositional atoms in the form $q_e$ is different from the usual practice of writing out the corresponding full linear inequality when referring to $q_e$.

**Definition 7.3.1.** The set of arithmetic propositional atoms, or more briefly arithmetic atoms, is

$$QAtm = \{q_e \mid e \in Q^{N+1}\}.$$  

**Example 7.3.2.** The arithmetic edit $ea = (1, -2, 1, 0)$ is formalised by the arithmetic atom $q_ea = q(1,-2,1,0)$.

For each record $v$ we define a truth function $g_v$, which we call an arithmetic truth function, defined below. We say that a set of formulae that is satisfied by an arithmetic truth function is arithmetically satisfiable, and we define arithmetic implication:

**Definition 7.3.3.** If $v$ is a record then the arithmetic truth function $g_v$ is defined for each arithmetic atom $q_e$ by:

$$g_v(q_e) = \begin{cases} 
\text{true} & \text{if } e \left(\frac{1}{v^r}\right) \geq 0 \\
\text{false} & \text{if } e \left(\frac{1}{v^r}\right) < 0.
\end{cases}$$

The arithmetic truth functions are defined to be exactly those truth functions of the form $g_v$:

the set of arithmetic truth functions = $\{g_v \mid v \in D\}$.

We say that the set $\Sigma$ of formulae is *arithmetically satisfiable* if there is some arithmetic truth function that satisfies $\Sigma$. The set $\Sigma$ of formulae *arithmetically implies*
the formula $\sigma$ if the following holds:

$$\text{if } g_v \text{ is an arithmetic truth function that satisfies } \Sigma, \text{ then } g_v(\sigma) = \text{true}. $$

If $\Sigma$ arithmetically implies $\sigma$ then we write $\Sigma \models_a \sigma$, where “a” stands for “arithmetic”.

**Example 7.3.4.** As in the previous examples, suppose that the edit $ea = (1, -2, 1, 0)$ and that the record $vv = (2, 1, -1)$. Then, since $vv$ fails the edit $ea$, we have that $g_{vv}(qe) = \text{false}$.

The arithmetic truth functions are distinct in the sense that if the records $v$ and $w$ are distinct then $g_v \neq g_w$, as seen in the next proposition.

**Proposition 7.3.5.** The mapping $v \mapsto g_v$ is a bijection from records to arithmetic truth functions.

**Proof.** The mapping $v \mapsto g_v$ is surjective since all arithmetic truth functions have the form $g_v$ for some record $v$.

To show that the mapping is injective, we suppose that $v \neq w$ and show that $g_v \neq g_w$. Since $v \neq w$, then with a suitable renumbering of the fields and with a possible swapping of $v$ and $w$, we have that $v_1 < w_1$. Let $e = (v_1, -1, 0, \ldots, 0)$. Then

$$e \left( \frac{1}{v^T} \right) = v_1 - v_1 \geq 0, \text{ and}$$

$$e \left( \frac{1}{w^T} \right) = v_1 - w_1 < 0.$$

Hence $g_v(q_e) = \text{true}$ and $g_w(q_e) = \text{false}$. Therefore $g_v \neq g_w$, as required. \(\square\)

We can now see that the function $g_v$ formalises the record $v$ in two senses:

1. it gives a bijection between records and arithmetic truth functions; and

2. it preserves the “correctness” of records in the sense that the record $v$ satisfies the arithmetic edit $e$ if and only if $g_v(q_e) = \text{true}$ (by the definition of $g_v$).

The arithmetic truth functions are the only truth functions we will consider. We should preferably define a theory Tha (where “a” stands for “arithmetic”) to be satisfied by all and only the arithmetic truth functions. Further comments about the theory Tha are given later, in Section 7.6.1.

For each record $v$, we have that $g_v(q_{(e_0, 0, \ldots, 0)}) = \text{true}$ if $e_0 \geq 0$, and that $g_v(q_{(e_0, 0, \ldots, 0)}) = \text{false}$ if $e_0 < 0$. We will write $\Box$ for the equivalence class of arithmetic atoms $q_{(e_0, 0, \ldots, 0)}$ where $e_0 < 0$. We should preferably define $\Box$ as one particular formula, but the definition as an equivalence class is adequate for our purposes here. Further discussion about $\Box$ is given later, in Section 7.6.2.
Definition 7.3.6. □ is the equivalence class of arithmetic atoms of the form q(e_0,0,...,0) where e_0 < 0. Ambiguously but comprehensibly, we will also use □ to represent any one member of the equivalence class.

We now formalise covering set correctibility, starting first with the concepts used in its definition. We will first formalise the concepts of error localisation, covering sets and involved fields. We will then formalise covering set correctibility, and its two directions, namely error correction guarantee and error correction totality.

Just as we did for categorical edits in Chapter 5, we will give the same name to each of the formalised concepts as to the original concept. That is, within logic the formalised concepts will also be called error localisation, covering sets, involved fields, covering set correctibility, error correction guarantee, and error correction totality. We will also use the same symbols. As for categorical edits, the ambiguity should not cause any confusion because the context will be clear.

The statement of the error localisation problem for arithmetic atoms is a direct translation of the statement of the error localisation problem, given on page 210, for arithmetic edits:

**Definition 7.3.7.** Let v = (v_1, ..., v_N) be a record and Σ be a set of logical formulae. (In general, we will require Σ to be a set of arithmetic atoms.) The error localisation problem is the problem of deciding which sets of fields can be changed to correct g_v with respect to Σ. That is, it is the problem of deciding, for a set C of fields, whether there is a record w = (w_1, ..., w_N) such that

a. C ⊇ {j | w_j ≠ v_j}, and
b. g_w satisfies Σ.

If there exists a record w such that a and b are satisfied, then we say that the set C yields a correction w to v with respect to Σ or that the set C yields a correction g_w to g_v with respect to Σ and we shall also say that C is a solution to the error localisation problem for v (or g_v) and Σ.

The set of solutions to the error localisation problem for the record v and the set Σ of arithmetic atoms will be written EL(Σ, v). This means that C yields a correction to v with respect to Σ if and only if C ∈ EL(Σ, v).

We use the words “involved” and “covering set”, and the notations X and C(X) in the same way for arithmetic atoms as defined for arithmetic edits (in Definitions 7.2.3, 7.2.4, 7.2.5 and 7.2.6 respectively):

**Definition 7.3.8.** The field k (k ∈ {1, ..., N}) is involved in the arithmetic atom q_e (e = (e_0, ..., e_N)) if e_k ≠ 0. The field k is uninvolved in the arithmetic atom q_e if e_k = 0.

**Definition 7.3.9.** A covering set C of the set Σ of arithmetic atoms is a set of fields such that for each atom q_e in Σ, there exists a field in C that is involved in q_e. The set of all covering sets of Σ is written C(Σ):

\[ C(Σ) = \{ C ⊆ \{1, ..., N\} \mid C \text{ is a covering set of Σ} \}. \]
Definition 7.3.10. Define the set \( \mathcal{X}(\Sigma, v) \) to be the set of arithmetic atoms in the set \( \Sigma \) that are failed by \( v \). That is, \( \mathcal{X}(\Sigma, v) = \{ q_e \in \Sigma \mid e(\frac{1}{v_T}) < 0 \} \).

Definition 7.3.11. Define \( \mathcal{CX}(\Sigma, v) \) to be the set of covering sets of the arithmetic atoms in the set \( \Sigma \) of arithmetic atoms that are failed by the record \( v \). That is \( \mathcal{CX}(\Sigma, v) = C(\mathcal{X}(\Sigma, v)) \).

Since we are only interested in arithmetic atoms, we will only consider those deduction functions whose domains are arithmetic atoms, and will use the term “arithmetic deduction function”:

Definition 7.3.12. An arithmetic deduction function is a function from sets of arithmetic atoms to sets of arithmetic atoms, that is a function \( \mathcal{G} \) where

\[
\mathcal{G} : \mathcal{P}(\text{QAtm}) \rightarrow \mathcal{P}(\text{QAtm}).
\]

Instead of the terms “soundness” and “completeness” we will use the terms “arithmetically sound” and “arithmetically complete”, because we are restricting the truth functions to arithmetic truth functions:

Definition 7.3.13. The arithmetic deduction function \( \mathcal{G} \) is arithmetically strongly sound if, for any set \( \Sigma \) of arithmetic atoms and for any arithmetic atom \( q_e \), the following statement holds:

\[
\text{if } q_e \in \mathcal{G}(\Sigma) \text{ then } \Sigma \models_a q_e.
\]

The arithmetic deduction function \( \mathcal{G} \) is arithmetically refutation sound if, for any set \( \Sigma \) of arithmetic atoms, the following statement holds:

\[
\text{if } \Box \in \mathcal{G}(\Sigma) \text{ then } \Sigma \models_a \Box.
\]

The arithmetic deduction function \( \mathcal{G} \) is arithmetically strongly complete if, for any set \( \Sigma \) of arithmetic atoms and for any arithmetic atom \( q_e \), the following statement holds:

\[
\text{if } \Sigma \models_a q_e \text{ then } q_e \in \mathcal{G}(\Sigma).
\]

The arithmetic deduction function \( \mathcal{G} \) is arithmetically refutation complete if, for any set \( \Sigma \) of arithmetic atoms, the following statement holds:

\[
\text{if } \Sigma \models_a \Box \text{ then } \Box \in \mathcal{G}(\Sigma).
\]

We now define covering set correctibility, and its two directions, error correction totality and error correction guarantee, following the definitions used for arithmetic edits (Definition 7.2.9):

Definition 7.3.14. The arithmetic deduction function \( \mathcal{G} \) has error correction totality if for any set \( \Sigma \) of arithmetic atoms and any record \( v \),

\[
\mathcal{EL}(\Sigma, v) \subseteq \mathcal{CX}(\mathcal{G}(\Sigma), v).
\]
The arithmetic deduction function $G$ has the **error correction guarantee** if for any set $\Sigma$ of arithmetic atoms and any record $v$,

$$\mathcal{C}(G(\Sigma), v) \subseteq \mathcal{E}(\Sigma, v).$$

The arithmetic deduction function $G$ is **covering set correctible** if for any set $\Sigma$ of arithmetic atoms and any record $v$,

$$\mathcal{C}(G(\Sigma), v) = \mathcal{E}(\Sigma, v).$$

An example of an arithmetic deduction function with covering set correctibility is the function $FM$, defined below, which is the logical formalisation of the function FM introduced in the previous section.

**Definition 7.3.15.** Let $\Sigma = \{q_e | e \in E\}$ be a set of arithmetic atoms, where $E$ is a set of arithmetic edits and

$$E = \begin{pmatrix} e_{01}^{1} & \cdots & e_{0N}^{1} \\ \vdots & \ddots & \vdots \\ e_{m1}^{N} & \cdots & e_{mN}^{N} \end{pmatrix}.$$  

For each $j \in \{0, \ldots, N\}$ we write $E_j$ as the $j$ column of $E$, that is

$$E_j = \begin{pmatrix} e_{j1}^{1} \\ \vdots \\ e_{jm}^{N} \end{pmatrix}.$$  

Then define $FM(\Sigma)$ to be the arithmetic atoms representing those non-negative linear combinations of the edits of $E$ where at least one variable is eliminated, excluding the edit $0$. That is,

$$FM(\Sigma) = \left\{ q_{\lambda E} | \lambda = (\lambda_1, \ldots, \lambda_m) \geq 0, \lambda \neq 0, \text{and there is an } i \in \{1, \ldots, N\} \text{ and a } k \in \{1, \ldots, m\} \text{ such that } \lambda_k \neq 0 \text{ and } e_{k}^{i} \neq 0 \text{ and } \lambda E_i = 0 \right\}.$$

**Example 7.3.16.** As in the previous examples, suppose that the edit $ea = (1, -2, 1, 0)$ and that the edit $eb = (0, 1, 0, 4)$. Suppose that $\Sigma \Sigma = \{q_{ea}, q_{eb}\}$. Then, as in Example 7.2.13, $FM(\Sigma) = \{q_{\lambda ea}, q_{\lambda eb}, q_{\lambda ec} | \lambda \in \mathbb{Q}, \lambda > 0\}$, where $ec = (1, 0, 1, 8)$.

**Proposition 7.3.17.** $FM$ is covering set correctible.

**Proof.** Translation of the corresponding proof for FM (proof of Theorem 7.2.12).

In this section we have seen a logical formalisation of arithmetic edits and related concepts. Similarly to the formalisation for categorical edits, arithmetic edits are formalised in propositional logic as logical formulae. However, differently from categorical edits, arithmetic edits are formalised as arithmetic propositional atoms. Similarly to categorical edits, records (when applied to arithmetic edits) are formalised as arithmetic truth functions. The main concepts related to arithmetic edits are formalised
via a direct translation from their definitions in relation to arithmetic edits. It would be useful, just as we did for categorical edits, to formalise covering set correctibility in terms of some natural constructs of logic, which we do in the next section.

7.4 Covering set correctibility for arithmetic atoms in terms of some natural constructs of logic

In this section, we will find a logical expression for covering set correctibility for arithmetic atoms in terms of some constructs that arise naturally in logic. The logical expression that we will find is an exact parallel to the result found in Chapter 5 for covering set correctibility for normal clauses. That is, the property of covering set correctibility for arithmetic atoms is an extension of the properties of completeness and soundness.

Our method follows the method used in Chapter 5 for the logical formalisation of categorical edits, but with an important difference. As in Chapter 5, we consider separately the two properties linked together by covering set correctibility, namely that of yielding a correction and that of being a covering set, and end up with a parallel set of results. Also, as in Chapter 5, the results depend on the definition of “reduced arithmetic atoms”, in parallel with Definition 5.3.1 of reduced normal clauses. But the definition of reduced normal clauses cannot be directly converted to a definition of reduced arithmetic atoms simply by replacing normal clauses by arithmetic atoms.

We can convert the definition of reduced normal clauses to a definition of reduced arithmetic atoms if we first remember the observation, of Note 4 on page 144, that the reduction of clauses is based on the same idea as the DPLL splitting rule (Davis et al. 1962), which itself is described in Chapter 3, Section 3.2.6. As in the splitting rule, we obtain the reduced normal clause \( \sigma[v, Z] \) from the normal clause \( \sigma \), the record \( v \) and the field set \( Z \), by first assigning the truth value determined by \( f_{v} \) to each propositional atom \( p_{j} \) whose field is in \( Z \), and then recalculating \( \sigma \). For example, suppose that \( N = 3 \); that \( A_{1} = A_{2} = A_{3} = \{1, 2, 3\} \); that \( \sigma = p_{1}^{2} \lor p_{3}^{2} \); that \( Z = \{1\} \); and that \( v = (3, 3, 3) \). By Definition 5.3.1, we have that \( \sigma[v, Z] = p_{3}^{2} \). The reduced clause \( \sigma[v, Z] \) could equally have been calculated by assigning the truth value \( f_{v}(p_{1}^{2}) \), which equals false, to the propositional atom \( p_{1}^{2} \) in \( \sigma \).

We now apply the above observation, not directly to the arithmetic atom \( q_{e} \), but to the arithmetic edit \( e \) and its corresponding linear inequality

\[
e_{0} + \sum_{j=1}^{N} e_{j} x_{j} \geq 0.
\]

Just as we calculated reduced normal clauses, so we can calculate the reduced arithmetic edit \( e[v, Z] \) by assigning the value \( v_{j} \) to each variable \( x_{j} \) whose field is in \( Z \). Thus we obtain the inequality

\[
e_{0} + \sum_{j \in Z} e_{j} v_{j} + \sum_{j \in \{1, \ldots, N\} \setminus Z} e_{j} x_{j} \geq 0,
\]
from which we can obtain each component of \( e[v, Z] \). In particular, its first component, 
\[
e[v, Z]_0, \quad \text{equals } e_0 + \sum_{j \in Z} e_j v_j. \]
The formal definition of the reduction of the arithmetic atom and its corresponding arithmetic edit is given below in Definition 7.4.1.

With the definition below of reduced arithmetic atoms, the results of this chapter parallel those of Chapter 5. Definition 7.4.1 of reduced arithmetic atoms is itself in parallel to Definition 5.3.1 of reduced clauses. In Theorems 7.4.5 and 7.4.7 we express the two properties linked together by covering set correctibility, in parallel to Theorems 5.3.5 and 5.3.6 respectively. The proof of Theorem 7.4.5 depends on Lemma 7.4.3 and Corollary 7.4.4, which parallel Lemma 5.3.3 and Corollary 5.3.4 respectively. Finally, in Proposition 7.4.9, which parallels Proposition 5.3.7 of Chapter 5, we will link our expressions for the two properties to obtain an expression for covering set correctibility.

**Definition 7.4.1.** If \( e = (e_0, \ldots, e_N) \) is an arithmetic edit, \( v = (v_1, \ldots, v_N) \) is a record, and \( Z \) is a field set, then the **reduction of \( e \) by \( v \) and \( Z \)** is written \( e[v, Z] \) whose components are defined as follows. First write \( Z = \{1, \ldots, N\} \setminus Z \). Then

\[
\text{the } j \text{ component of } e[v, Z] = e[v, Z]_j = \begin{cases} 
  e_0 + \sum_{k \in Z} e_k v_k & j = 0 \\
  0 & j \in Z \\
  e_j & j \in Z.
\end{cases}
\]

If \( E \) is a set of arithmetic edits, then the **reduction of \( E \) by \( v \) and \( Z \)** is

\[
E[v, Z] = \left\{ e[v, Z] \mid e \in E \right\}.
\]

Equivalently, if \( E = \{e_1, \ldots, e_m\} \) then, as given in Definition 7.2.1, we can write

\[
E[v, Z] = \begin{pmatrix}
  e_1[v, Z]|_0 & \cdots & e_1[v, Z]|_N \\
  \vdots & \ddots & \vdots \\
  e_m[v, Z]|_0 & \cdots & e_m[v, Z]|_N
\end{pmatrix}.
\]

If \( q_e \) is an arithmetic atom, then the **reduction of \( q_e \) by \( v \) and \( Z \)** is

\[
q_e[v, Z] = q_e[v, Z].
\]

If \( \Sigma \) is a set of arithmetic atoms, then the **reduction of \( \Sigma \) by \( v \) and \( Z \)** is

\[
\Sigma[v, Z] = \left\{ q_e[v, Z] \mid q_e \in \Sigma \right\}.
\]

Equivalently, if \( \Sigma = \{q_e \mid e \in E\} \), where \( E \) is a set of arithmetic edits, then

\[
\Sigma[v, Z] = \left\{ q_d \mid d \in E[v, Z] \right\}.
\]

**Example 7.4.2.** As in the previous examples, suppose that \( vv = (2, 1, -1) \) and that
Chapter 7.4  Covering set correctibility in terms of some natural constructs of logic

\[ EE = \{ea, eb\}, \]

where

\[ ea = (1, -2, 1, 0), \quad \text{and} \]
\[ eb = (0, 1, 0, 4), \]

and that \( \Sigma\Sigma = \{q_{ea}, q_{eb}\} \). Suppose also that the field set \( ZZ = \{2, 3\} \). We calculate the reduction of \( \Sigma\Sigma \) by \( vv \) and \( ZZ \).

We first calculate the reduction of each arithmetic edit in \( EE \). Since the edit \( ea \) represents the linear inequality \( 1 - 2x_1 + x_2 \geq 0 \), the reduced edit \( ea[vv, ZZ] \) represents the linear inequality \( 1 - 2x_1 + 1 \geq 0 \). Hence

\[ ea[vv, ZZ] = (2, -2, 0, 0), \quad \text{and} \]
\[ q_{ea}[vv, ZZ] = q_{(2, -2, 0, 0)}. \]

We can similarly calculate the reduction of the edit \( eb \):

\[ eb[vv, ZZ] = (-4, 1, 0, 0), \]
\[ q_{eb}[vv, ZZ] = q_{(-4, 1, 0, 0)}. \]

Hence

\[ EE[vv, ZZ] = \{(2, -2, 0, 0), (-4, 1, 0, 0)\}, \quad \text{and} \]
\[ \Sigma\Sigma[vv, ZZ] = \{q_{(2, -2, 0, 0)}, q_{(-4, 1, 0, 0)}\}. \]

Note that there is no record that satisfies \( EE[vv, ZZ] \), and hence \( \Sigma\Sigma[vv, ZZ] \) is not arithmetically satisfiable.

By Example 7.3.16, we have that \( FM(\Sigma\Sigma) = \{q_{\lambda ea}, q_{\lambda eb}, q_{\lambda ec} \mid \lambda \in \mathbb{Q}, \lambda > 0\} \), where \( ec = (1, 0, 1, 8) \).

We calculate the reduction of the edit \( ec \):

\[ ec[vv, ZZ] = (-6, 0, 0, 0), \quad \text{and} \]
\[ q_{ec}[vv, ZZ] = q_{(-6, 0, 0, 0)}. \]

Note that \( q_{ec}[vv, ZZ] = \Box. \) Then

\[ (FM(\Sigma\Sigma))[vv, ZZ] = \{q_{\lambda(-2, -2, 0, 0)}, q_{\lambda(-4, 1, 0, 0)}, \Box \mid \lambda \in \mathbb{Q}, \lambda > 0\}. \]

Having defined reduced arithmetic atoms, we are ready to express the first of the two properties linked together by covering set correctibility. We will express the property of yielding a correction in terms of the arithmetic satisfiability of reduced arithmetic atoms. We will show below, in Theorem 7.4.5, that \( C \) yields a correction to the record \( v \) with respect to \( \Sigma \) if and only if \( \Sigma[v, Z] \) is arithmetically satisfiable.

The proof of Theorem 7.4.5 depends on the following lemma and corollary compar-
ing arithmetic truth functions that satisfy $\Sigma$ to arithmetic truth functions that satisfy the reduced set $\Sigma[v, Z]$, for $Z$ a set of fields. Rather than being in terms of the set $\Sigma$ of arithmetic atoms, the lemma and corollary are in terms of an individual arithmetic atom $q_e$.

**Lemma 7.4.3.** Let $Z$ be a set of fields; and let $q_e$ be an arithmetic atom. Let $v = (v_1, \ldots, v_N)$, $y = (y_1, \ldots, y_N)$ and $w = (w_1, \ldots, w_N)$ be records such that $w$ is the same as $v$ on the fields in $Z$, and $w$ is the same as $y$ on the fields in $\overline{Z}$. That is, for each $j$ in $\{1, \ldots, N\}$,

$$j \in Z \Rightarrow w_j = v_j,$$

$$j \in \overline{Z} \Rightarrow w_j = y_j.$$

Then $gw$ satisfies $q_e$ if and only if $gy$ satisfies $q_{e[v, Z]}$.

**Proof.** We first note, by Definition 7.3.3, that

$$gy(q_{e[v, Z]}) = \text{true} \text{ if and only if } e[v, Z]\left(\frac{1}{y^T}\right) \geq 0 \quad (7.4.1)$$

and that

$$gw(q_e) = \text{true} \text{ if and only if } e\left(\frac{1}{w^T}\right) \geq 0. \quad (7.4.2)$$

We will show that the inequality in Statement 7.4.1 holds if and only if the inequality in Statement 7.4.2 holds. We first calculate each component of the left-hand side of the inequality in Statement 7.4.1:

$$e[v, Z]_0 = e_0 + \sum_{j \in Z} e_j v_j, \quad \text{by Definition 7.4.1 of } e[v, Z]$$

$$= e_0 + \sum_{j \in Z} e_j w_j, \quad \text{by the definition of } w;$$

if $j \in Z$, then $e[v, Z]_j y_j = 0$, \quad \text{by Definition 7.4.1 of } e[v, Z];

if $j \in \overline{Z}$, then $e[v, Z]_j y_j = e_j y_j$, \quad \text{by Definition 7.4.1 of } e[v, Z]

$$= e_j w_j, \quad \text{by the definition of } w.$$

Hence

$$e[v, Z]\left(\frac{1}{y^T}\right) = e_0 + \sum_{j \in Z} e_j w_j + \sum_{j \in Z} e_j w_j = e\left(\frac{1}{w^T}\right),$$

giving us that the inequality of Statement 7.4.1 holds if and only if the inequality of Statement 7.4.2 holds. Hence, by Statements 7.4.1 and 7.4.2, the result follows. \qed

**Corollary 7.4.4.** Let $Z$ be a set of fields; and let $q_e$ be an arithmetic atom. Let $v = (v_1, \ldots, v_N)$ and $w = (w_1, \ldots, w_N)$ be records such that if $j \in Z$ then $w_j = v_j$. 

Then \( g_w \) satisfies \( q_e \) if and only if \( g_w \) satisfies \( q_e[v, Z] \).

**Proof.** In Lemma 7.4.3, let \( y = w \).

We now use the above lemma and corollary in the proof of the next theorem, giving a logical expression for the property of yielding a correction.

**Theorem 7.4.5.** Let \( C \) be a set of fields, let \( v \) be a record, and let \( \Sigma \) be a set of arithmetic atoms. Then the field set \( C \) yields a correction to \( v \) with respect to the set \( \Sigma \) if and only if \( \Sigma[v, C] \) is arithmetically satisfiable.

**Proof.**

**Forward direction:** Suppose that \( C \) yields the correction \( w \) to \( v \) with respect to \( \Sigma \). We write \( v = (v_1, \ldots, v_N) \) and \( w = (w_1, \ldots, w_N) \). Then, by Definition 7.3.7,

a. if \( j \in C \), then \( w_j = v_j \); and

b. \( g_w \) satisfies \( \Sigma \).

Then, by Corollary 7.4.4 (with \( Z = C \)), the truth function \( g_w \) satisfies each element of \( \Sigma[v, C] \), which is thereby arithmetically satisfiable, as required.

**Backward direction.** Suppose that \( \Sigma[v, C] \) is satisfied by the arithmetic truth function \( g_y \), where \( v = (v_1, \ldots, v_N) \) and \( y = (y_1, \ldots, y_N) \). Define \( w = (w_1, \ldots, w_N) \) so that it differs from \( v \) only on the fields of \( C \), where it is defined to be the same as \( y \):

\[
    w_j = \begin{cases} 
        v_j & j \in C \\
        y_j & j \in C.
    \end{cases}
\]

Then, by Lemma 7.4.3 (with \( Z = C \)), we have that \( g_w \) satisfies \( \Sigma \). Hence \( C \) yields a correction \( g_w \) to \( g_y \) with respect to \( \Sigma \), as required. \( \square \)

**Example 7.4.6.** As in the previous examples, suppose that \( vv = (2, 1, -1) \) and that \( EE = \{ ea, eb \} \), where

\[
    ea = (1, -2, 1, 0), \quad \text{and} \\
    eb = (0, 1, 0, 4),
\]

and that \( \Sigma \Sigma = \{ q_{ea}, q_{eb} \} \). Then as seen in Example 7.2.10, the set \( CC = \{ 1 \} \) does not yield a correction to \( EE \) with respect to the record \( vv \). Similarly \( CC \) does not yield a correction to \( \Sigma \Sigma \) with respect to the record \( vv \). Also, as predicted by the above theorem, and as noted in Example 7.4.2, the set \( \Sigma \Sigma[vv, CC] = \{ q_{1,-2,0,0}, q_{(-4,1,0,0)} \} \) is not arithmetically satisfiable.

Having found a logical expression for the property of yielding a correction, we will now find a logical expression for the other component of covering set correctibility, namely (in generalised form) the property of being a covering set of a set of failed arithmetic atoms.
Theorem 7.4.7. If $C$ is a set of fields and $\Sigma$ is a set of arithmetic atoms, then $C$ is a covering set of $X(\Sigma, v)$ if and only if $\Box \not\in \Sigma[v, \overline{C}]$.

Proof.
Forward direction. Suppose that $\Box \in \Sigma[v, \overline{C}]$, and suppose that $\Sigma = \{q_e \mid e \in E\}$. Then there is an arithmetic edit $e[v, \overline{C}]$ in $E[v, \overline{C}]$ such that $q_{e[v, \overline{C}]} = \Box$, that is
\[
e[v, \overline{C}]_0 < 0, \quad \text{and} \quad e[v, \overline{C}]_1 = \cdots = e[v, \overline{C}]_N = 0. \tag{7.4.3}
\]
We have, by Definition 7.4.1, that
\[
e[v, \overline{C}]_0 = e_0 + \sum_{k \in \overline{C}} e_k v_k, \quad \text{and} \quad (7.4.5)
\]
if $j \in C$, then $e[v, \overline{C}]_j = e_j$, and hence, by Equation 7.4.4, $e_j = 0$. \tag{7.4.6}
Hence
\[
e \left( \frac{1}{v} \right) = e_0 + \sum_{k \in \overline{C}} e_k v_k, \quad \text{since by Statement 7.4.6, if } j \in C, \text{ then } e_j = 0
\]
\[
= e[v, \overline{C}]_0, \quad \text{by Equation 7.4.5}
\]
\[
< 0, \quad \text{by Statement 7.4.3.}
\]
Hence, by Definition 7.3.3 for $g_v$, we have that
\[
g_v(q_e) = \text{false}.
\]
Hence
\[
qu \in X(\Sigma, v).
\]
But no field of $C$ is involved in $q_e$ because, by Statement 7.4.6 above, if $j \in C$ then $e_j = 0$. So $C$ does not cover $X(\Sigma, v)$.

Backward direction. The proof is the reverse of the proof for the forward direction. Suppose that the field set $C$ does not cover $X(\Sigma, v)$. Then there is an arithmetic atom $q_e$ in $X(\Sigma, v)$ such that no element of $C$ is involved in $q_e$, that is, writing $e = (e_0, \ldots, e_N)$,
\[
\text{for each } j \in C, \quad e_j = 0. \tag{7.4.7}
\]
We calculate $e[v, \overline{C}]$, by calculating its components for $j = 0, \ldots, N$:
\[
e[v, \overline{C}]_0 = e_0 + \sum_{j \in \overline{C}} e_j v_j, \quad \text{by Definition 7.4.1}
\]
\[
= e_0 + \sum_{j \in \overline{C}} e_j v_j + \sum_{j \in C} e_j v_j, \quad \text{by Statement 7.4.7}
\]
= e\left( \frac{1}{v^T} \right)

< 0, \quad \text{since } g_v(q_e) = \text{false and using Definition 7.3.3 for } g_v; 

\text{if } j \in C, \quad \text{then } e[v, C]_j = e_j, \quad \text{by Definition 7.4.1} 

= 0, \quad \text{by Statement 7.4.7}; 

\text{if } j \in \overline{C}, \quad \text{then } e[v, C]_j = 0, \quad \text{by Definition 7.4.1}.

Hence, by Definition 7.3.6, we have that 

\quad q_e[v, C] = \square, \quad \text{and since } q_e \in \Sigma, \quad \text{we have that} 

\quad \square \in \Sigma[v, C], \quad \text{as required.}

\quad \square \in \Sigma[v, C], \quad \text{as required.}

Example 7.4.8. As in the previous examples, suppose that 

\quad vv = (2, 1, -1) \quad \text{and that} 

\quad EE = \{ea, eb\}, \quad \text{where} 

\quad ea = (1, -2, 1, 0), \quad \text{and} 

\quad eb = (0, 1, 0, 4), 

\quad \text{and that } \Sigma = \{e_{ea}, e_{eb}\}. \quad \text{Then, as seen in Example 7.2.7, the set } \mathcal{X}(EE, vv) = EE, \quad \text{one of whose covering sets is the field set } CC = \{1\}. \quad \text{Similarly, the set } \mathcal{X}(\Sigma, vv) = \Sigma, \quad \text{one of whose covering sets is the field set } CC. \quad \text{Also, from Example 7.4.2, we have that} 

\quad \Sigma[v, \overline{C}] = \{q_{(1, -2, 0, 0)}, q_{(-4, 1, 0, 0)}\}, \quad \text{which, as predicted by the above theorem, does not contain } \square.

With the above theorem, we now have a logical expression for each of the two properties linked together by covering set correctibility. We bring the two theorems together in the next proposition to give a logical expression for covering set correctibility, as well as of its two directions, error correction totality and error correction guarantee.

Proposition 7.4.9. Let \( \mathcal{G} \) be an arithmetic deduction function.

1. The arithmetic deduction function \( \mathcal{G} \) has error correction totality if and only if the following statement holds: For each set \( \Sigma \) of arithmetic atoms, each field set \( Z \), and each record \( v \)

\quad \Sigma[v, Z] \text{ is arithmetically satisfiable } \Rightarrow \square \notin (\mathcal{G}(\Sigma))[v, Z].

2. The arithmetic deduction function \( \mathcal{G} \) has the error correction guarantee if and only if the following statement holds: For each set \( \Sigma \) of arithmetic atoms, each field set \( Z \), and each record \( v \)

\quad \square \notin (\mathcal{G}(\Sigma))[v, Z] \Rightarrow \Sigma[v, Z] \text{ is arithmetically satisfiable.}

3. The arithmetic deduction function \( \mathcal{G} \) is covering set correctible if and only if the following statement holds: For each set \( \Sigma \) of arithmetic atoms, each field set \( Z \), and each record \( v \)

\quad \square \notin (\mathcal{G}(\Sigma))[v, Z] \iff \Sigma[v, Z] \text{ is arithmetically satisfiable.}
Proof. For the proof of part 1, we use the definition of error correction totality given in Definition 7.3.14. We start with the formula used in that definition and derive the following sequence of equivalent statements:

\[ E \mathcal{L}(\Sigma, v) \subseteq C \mathcal{X}(G(\Sigma), v) \]

\( \Leftrightarrow \) for each field set \( C \), if \( C \) yields a correction to \( v \) with respect to \( \Sigma \), then \( C \) is a covering set of \( \mathcal{X}(G(\Sigma), v) \)

(by the definitions of \( C \) and \( E \mathcal{L} \))

\( \Leftrightarrow \) for each field set \( C \), if \( \Sigma[v, C] \) is arithmetically satisfiable, then \( \square \not\in (G(\Sigma))[v, C] \)

(by Theorems 7.4.5 and 7.4.7)

\( \Leftrightarrow \) for each field set \( Z \), if \( \Sigma[v, Z] \) is arithmetically satisfiable, then \( \square \not\in (G(\Sigma))[v, Z] \)

(since there is a one-to-one correspondence between field sets and their complements).

Since the above sequence of statements holds for any \( \Sigma \) and \( v \), then part 1 follows.

The proof of part 2 is identical in structure to the proof of part 1. In the first line of the sequence of equivalent statements, the subset relation is reversed to obtain the definition of the error correction guarantee. In the remaining lines of the sequence of equivalent statements, the implication direction is reversed.

Part 3 follows directly from parts 1 and 2. \( \dashv \)

Example 7.4.10. The deduction function \( F.M \) is covering set correctible. The previous examples support the above proposition, as follows. We supposed that \( vv = (2, 1, -1) \) and that \( \Sigma \Sigma = \{q_{ea}, q_{eb}\} \), where \( ca = (1, -2, 1, 0) \), and \( eb = (0, 1, 0, 4) \). In Example 7.3.16, we saw that \( F.M(\Sigma \Sigma) = \{q_{\lambda ea}, q_{\lambda eb}, q_{\lambda ec} \mid \lambda \in \mathbb{Q}, \lambda > 0\}, \) where \( ec = (1, 0, 1, 8) \). In Example 7.4.2, we saw that the condition of the above proposition is satisfied as follows:

1. \( \Sigma \Sigma[v, ZZ] \) is arithmetically unsatisfiable because it is \( \{q_{(1, -2, 0, 0)} : q_{(-4, 1, 0, 0)}\} \);

and

2. \( \square \in (F.M(\Sigma \Sigma))[v, ZZ] \) which is \( \{q_{\lambda(1, -2, 0, 0)} : q_{\lambda(-4, 1, 0, 0)} \mid \square \lambda \in \mathbb{Q}, \lambda > 0\} \).

The above proposition is parallel to Proposition 5.3.7 in Chapter 5, and gives strengthenings of arithmetic refutation soundness and arithmetic refutation completeness. Indeed, if \( Z \) is empty, then the above proposition simplifies in the same way as does Proposition 5.3.7. For if \( Z \) is empty, the above proposition becomes a statement of the arithmetic refutation soundness of those arithmetic deduction functions that have the error correction totality, and a statement of the arithmetic refutation completeness of those arithmetic deduction functions that have the error correction guarantee. In the next section we consider in more detail the relationships between soundness, completeness and covering set correctibility.
7.5 Arithmetic soundness, completeness and covering set correctibility

In this section we look at the relationships between soundness, completeness and covering set correctibility. We will obtain similar results to those of Section 5.4 in Chapter 5. We will find that error correction totality is equivalent to arithmetic strong soundness, while the error correction guarantee is between arithmetic strong and arithmetic refutation completeness.

We start with error correction totality, firstly with the equivalence of arithmetic strong soundness and error correction totality.

**Proposition 7.5.1.** The arithmetic deduction function $G$ has error correction totality if and only if it is arithmetically strongly sound.

*Proof.*

**Forward direction.** Suppose that $G$ has error correction totality. Let $\Sigma$ be a set of arithmetic atoms and let $v$ be a record such that $gv$ satisfies $\Sigma$, so that $\emptyset \in CL(\Sigma, v)$. Then, since $G$ has error correction totality, we have that $\emptyset \in CX(G(\Sigma), v)$, which means that $X(G(\Sigma), v) = \emptyset$. Hence $gv$ satisfies $G(\Sigma)$. This gives us that $G$ is arithmetically strongly sound.

**Backward direction.** Suppose that $G$ is arithmetically strongly sound. We will use the characterisation of error correction totality given by Proposition 7.4.9, part 1. Let $\Sigma$ be a set of arithmetic atoms; let $v = (v_1, \ldots, v_N)$ be a record; and let $Z$ be a set of fields. We suppose that $\Sigma[v, Z]$ is arithmetically satisfiable, and we will show that $\Box \not\in (G(\Sigma))[v, Z]$.

Suppose that an arithmetic truth function that satisfies $\Sigma[v, Z]$ is $g_y$, with $y = (y_1, \ldots, y_N)$. Let $w = (w_1, \ldots, w_N)$ be defined for each $j = 1, \ldots, N$ by

$$w_j = \begin{cases} v_j, & j \in Z \\ y_j, & j \in \mathbb{Z}. \end{cases}$$

Then by Lemma 7.4.3, we have that $gw$ satisfies $\Sigma$. Then, by the arithmetic strong soundness of $G$, we also have that $gw$ satisfies $G(\Sigma)$. Then using Corollary 7.4.4, we have that $gw$ satisfies $(G(\Sigma))[v, Z]$, so that $\Box \not\in (G(\Sigma))[v, Z]$, as required. \(\dashv\)

Having dealt with error correction totality, we now consider the error correction guarantee. Firstly, we confirm that every arithmetic deduction function with the error correction guarantee is also arithmetically refutation complete (Definition 7.3.13).

**Lemma 7.5.2.** If the arithmetic deduction function $G$ has the error correction guarantee, then it is arithmetically refutation complete.

*Proof.* The result follows from Proposition 7.4.9 in which we replace the set $Z$ by the empty set, and note that for any set $\Gamma$ of arithmetic atoms (including $\Gamma = \Sigma$ and $\Gamma = G(\Sigma)$), we have that $\Gamma[v, \emptyset] = \Gamma$. \(\dashv\)
In order to show the relationship between the error correction guarantee and arithmetic strong completeness, we will use a corollary of Farkas’ Lemma, a variant of which follows.

**Theorem 7.5.3 (variant of Farkas’ Lemma (Farkas 1894, 1895)).** Let \( A \) be an \( m \times n \) matrix and let \( b \) be an \( m \)-dimensional vector. Then the inequality \( Ax^T \leq b^T \) has no solution if and only if there is an \( m \)-dimensional vector \( y \) such that \( y \geq 0 \) and \( yA = 0 \) and \( yb^T < 0 \).

**Proof.** See for example the proof presented by Korte and Vygen (2002, Theorem 3.19, pages 58–59).

**Corollary 7.5.4.** Let \( \Sigma = \{ q_s \mid s \in S \} \) be a set of arithmetic atoms, where \( S \) is a set of arithmetic edits and
\[
S = \begin{pmatrix}
s_0^1 & \cdots & s_N^1 \\
s_0^2 & \cdots & s_N^2 \\
\vdots & \ddots & \vdots \\
s_0^m & \cdots & s_N^m
\end{pmatrix}.
\]
Then \( \Sigma \) is arithmetically unsatisfiable if and only if there is a vector \( \lambda \) such that \( \lambda \geq 0 \) and \( \lambda S = (-1, 0, \ldots, 0) \).

**Proof.** Apply Theorem 7.5.3, with \( b^T \) equal to the left-hand column of \( S \), that is,
\[
b^T = \begin{pmatrix}
s_0^1 \\
\vdots \\
s_0^m
\end{pmatrix};
\]
with \( A \) equal to the negation of the \( N \) right-hand columns of \( S \), that is,
\[
A = -\begin{pmatrix}
s_1^1 & \cdots & s_N^1 \\
\vdots & \ddots & \vdots \\
s_1^m & \cdots & s_N^m
\end{pmatrix};
\]
and, for the forward direction, with
\[
\lambda = -\frac{y}{yb^T};
\]
and, for the backward direction, with
\[
\lambda = y.
\]

Note: The above corollary can be viewed as a statement of the arithmetic strong soundness and arithmetic refutation completeness of \( \mathcal{FM} \). Bockmayr and Weispfenning (2001, p. 762) also note that Farkas’ Lemma is a soundness and completeness
result for linear inequalities.

Using the above consequence of Farkas’ Lemma, we can now prove the following proposition that every arithmetic deduction function that is arithmetically strongly complete also has the error correction guarantee.

**Proposition 7.5.5.** If \( G \) is an arithmetic deduction function that is arithmetically strongly complete, then it has the error correction guarantee.

*Proof.* We will use the characterisation of the error correction guarantee given by Proposition 7.4.9, part 2. Let \( \Sigma \) be a set of arithmetic atoms; let \( v = (v_1, \ldots, v_N) \) be a record; and let \( Z \) be a set of fields. Suppose that \( \Sigma = \{ q_s \mid s \in S \} \), where \( S \) is a set of arithmetic edits and \( S = \{ s_1, \ldots, s_m \} \), or in matrix notation

\[
S = \begin{pmatrix}
s^1_0 & \cdots & s^1_N \\
\vdots & \ddots & \vdots \\
s^m_0 & \cdots & s^m_N
\end{pmatrix}.
\]

We suppose that \( \Sigma[v, Z] \) is not arithmetically satisfiable, and we will show that \( \square \in (G(\Sigma))[v, Z] \).

We first note that \( \Sigma[v, Z] = \{ q_d \mid d \in S[v, Z] \} \), and \( S[v, Z] = \{ s^1[v, Z], \ldots, s^m[v, Z] \} \), or in matrix notation

\[
S[v, Z] = \begin{pmatrix}
s^1[v, Z]_0 & \cdots & s^1[v, Z]_N \\
\vdots & \ddots & \vdots \\
s^m[v, Z]_0 & \cdots & s^m[v, Z]_N
\end{pmatrix}.
\]

Since \( \Sigma[v, Z] \) is not arithmetically satisfiable, then by Corollary 7.5.4 there is a vector \( \lambda = (\lambda_1, \ldots, \lambda_n) \) such that \( \lambda \geq 0 \) and

\[
\lambda S[v, Z] = (-1, 0, \ldots, 0).
\]

We use \( \lambda \) to define the arithmetic edit \( e \):

\[
e = \lambda S = \sum_{k=1}^{m} \lambda_k s^k.
\]

We will prove two things. Firstly, we will show that \( q_e[v, Z] \in (G(\Sigma))[v, Z] \). Secondly, we will show that \( q_e[v, Z] = \square \). We will then have the required result that \( \square \in (G(\Sigma))[v, Z] \).

We first show that \( q_e[v, Z] \in (G(\Sigma))[v, Z] \), or equivalently that \( q_e \in G(\Sigma) \), which we obtain by showing that \( \Sigma \models_a q_e \) and then using arithmetic strong completeness. In order to show that \( \Sigma \models_a q_e \), let \( g_y \) be an arithmetic truth function that satisfies \( \Sigma \),
where $y = (y_1, \ldots, y_N)$. Then for each $k = 1, \ldots, m$, we have that

$$s^k \left( \frac{1}{y^r} \right) \geq 0.$$  

Since $\lambda \geq 0$, we can conclude that

$$\left( \sum_{k=1}^{m} \lambda_k s^k \right) \left( \frac{1}{y^r} \right) \geq 0.$$  

Hence, by the definition of $e$,

$$e \left( \frac{1}{y^r} \right) \geq 0,$$

so that

$$g_{y}(qe) = \text{true}.$$  

But $y$ is any record, and hence $\Sigma \models_a qe$. Then by arithmetic strong completeness, $qe \in G(\Sigma)$, and hence $qe[v, Z] \in (G(\Sigma))[v, Z]$.

We now show that $qe[v, Z] = \Box$, by showing that $e[v, Z] = (-1, 0, \ldots, 0)$. We calculate each component of $e[v, Z]$:

- For $j \in Z$,
  
  $$e[v, Z]_j = e_j = s^k_j[v, Z],$$
  
  by Definition 7.4.1 of $qe[v, Z]$, by the definition of $e$.

- For $j \not\in Z$, then
  
  $$e[v, Z]_j = e_j = s^k_j[v, Z],$$
  
  by Definition 7.4.1 of $qe[v, Z]$.

Hence $e[v, Z] = (-1, 0, \ldots, 0)$, and hence $qe[v, Z] = \Box$.

We have shown that $qe[v, Z] \in (G(\Sigma))[v, Z]$ and that $qe[v, Z] = \Box$. Hence we have that $\Box \in (G(\Sigma))[v, Z]$, as required.
Example 7.5.6. The function $\mathcal{F}M$ is covering set correctible, and hence is arithmetically strongly sound and arithmetically refutation complete. In effect, the covering set correctibility of $\mathcal{F}M$ is a strengthening of Farkas' Lemma.

In this section we have examined the relationship between error correction totality and arithmetic soundness, and the relationship between the error correction guarantee and arithmetic completeness. Figure 7.1 summarises the results of this section: the upper box represents the fact that error correction totality is equivalent to arithmetic strong soundness, while the lower box represents the fact that the error correction guarantee lies between arithmetic strong completeness and arithmetic refutation completeness.

Figure 7.1 is entirely in parallel with Figure 5.1 of Chapter 5. Thus the relationships between soundness, completeness and covering set correctibility for arithmetic edits and their logical formalisation are entirely in parallel with the corresponding results for categorical edits and their logical formalisation.

![Diagram 7.1a](image1)

Figure 7.1a: Error correction totality related to arithmetic soundness.

![Diagram 7.1b](image2)

Figure 7.1b: Error correction guarantee related to arithmetic completeness.

**Figure 7.1:** Diagrammatic representation of the relationship between error correction totality and arithmetic soundness, and the relationship between the error correction guarantee and arithmetic completeness. The upper figure, Figure 7.1a, represents the fact that error correction totality is equivalent to arithmetic strong soundness. The lower figure, Figure 7.1b, represents the fact that the error correction guarantee lies between arithmetic strong completeness and arithmetic refutation completeness.
Our next step is to summarise the results of this chapter. But first, we point out some possible further work.

7.6 Further work

This section contains brief comments on four ways of extending the work of this chapter. Firstly, as noted in Section 7.3, on page 217, arithmetic truth functions would be more satisfactorily defined in terms of a theory Tha: such a theory is the topic of the next subsection. Secondly, as noted in Section 7.3, on page 217, the notation $\Box$ is used both as an equivalence class and as individual formulae within that equivalence class. As discussed below in Subsection 7.6.2, the proofs and other discussion would still follow through if $\Box$ were defined as a specific formula, but would become more complex. Thirdly, as noted in Section 7.2 on page 215, and discussed further below in Subsection 7.6.3, the function FM, and its logical formalisation $FM$, can be modified by choosing fields to eliminate according to a field code forest in a similar way to the Field Code Forest Algorithm. Fourthly, as noted in the introduction to this chapter, there are many more types of edits than categorical and arithmetic edits. Some extensions to more general edits are discussed in Subsection 7.6.4.

7.6.1 The theory Tha

In Section 7.3, we defined the semantics of the logic by requiring that the only truth functions we consider be the arithmetic truth functions. A more satisfactory definition of the semantics would be in terms of a theory, which could be called Tha. We would then prove that

1. each Tha-truth function is a unique arithmetic truth function, and that
2. each arithmetic truth function is a Tha-truth function, that is each arithmetic truth function satisfies Tha.

The theory for the semantics would include the sets Tha1, Tha2 and Tha3, defined below, but would need to include other formulae as well.

Definition 7.6.1.

$$\text{Tha1} = \{ \neg q_e \lor \neg q_{e'} \lor q_{\lambda e + \mu e'} \mid e, e' \in \mathbb{Q}_N^N, \lambda, \mu \text{ are positive rationals} \},$$

and

$$\text{Tha2} = \{ q_e \lor q_{-e} \mid e \in \mathbb{Q}_N^N \},$$

and

$$\text{Tha3} = \{ \neg q_{(-\lambda,0,...,0)} \mid \lambda \text{ is a positive rational} \}.$$
With a suitable theory Tha, we would prove items 1 and 2 above. We show below that item 2 holds for the sets Tha1, Tha2 and Tha3, that is, that each arithmetic truth function satisfies the sets Tha1, Tha2 and Tha3.

**Proposition 7.6.2.** If $v$ is a record then the arithmetic truth function $g_v$ satisfies Tha1, Tha2 and Tha3.

**Proof.** The result follows from the properties of inequalities given below for the $(N+1)$-tuples $e$ and $e'$ and the positive rationals $\lambda$ and $\mu$.

(i) If $e\left(\frac{1}{v^T}\right) \geq 0$ and $e'\left(\frac{1}{v^T}\right) \geq 0$, then $(\lambda e + \mu e')\left(\frac{1}{v^T}\right) \geq 0$.

(ii) If $e\left(\frac{1}{v^T}\right) < 0$ then $-e\left(\frac{1}{v^T}\right) > 0$.

(iii) If $\lambda > 0$ then $(-\lambda, 0, \ldots, 0)\left(\frac{1}{v^T}\right) < 0$.

There remains additional work:

a. Prove item 1 above, that is, if $f$ is a Tha-truth function, then there is a unique record $v$ such that $f = g_v$.

b. Define a theory Tha such that it is possible to prove item 1 above and which is satisfied by all arithmetic truth functions. It seems likely that the theory Tha will have to include a property of limits or continuity (see for comparison Kuhn (1956, page 231)); and upper and lower bounds on the fields (or a specific field value $\infty$) - see for comparison Riera-Ledesma and Salazar-González (2007a).

### 7.6.2 The notation □

In Section 7.3, Definition 7.3.6, we defined the formula □ to be either a certain equivalence class of formulae or, ambiguously but comprehensibly, as any one member of the equivalence class. The definition is less than rigorous, but all the results follow through smoothly.

In order to be more rigorous it would be better to define □ as one particular formula chosen from its equivalence class, for example as the formula $q(−1,0,\ldots,0)$. In that case other definitions would also have to be amended. For example, the logical formalisation of any arithmetic edit of the form $(e_0,0,\ldots,0)$ where $e_0 < 0$ would have to be the arithmetic atom $q(−1,0,\ldots,0)$ rather than, as currently required by Definition 7.3.1, the arithmetic atom $q(e_0,0,\ldots,0)$. Some of the proofs would also have to be modified, as would the definition of the function $\mathcal{F}_\mathcal{M}$. 
7.6.3 A field code forest variation of the function $\mathcal{FM}$

There exists an ordered version of the arithmetic deduction function $\mathcal{FM}$: instead of taking all positive linear combinations of the input set of edits, we start with an ordering of the fields and eliminate one field at a time according to the ordering. In fact the original work of Fourier (1823, 1824, 1826) presents such an ordered function. The ordered version of $\mathcal{FM}$ is analogous to ordered resolution (Definition 3.2.19).

As shown by Fourier (1824), Dines (1919), Kuhn (1956), Duffin (1974), and Fellegi and Holt (1976), the ordered version of $\mathcal{FM}$ is arithmetically refutation complete and arithmetically strongly sound. However, similarly to ordered resolution, the ordered version of $\mathcal{FM}$ is not covering set correctible.

However, there is another version of the arithmetic deduction function $\mathcal{FM}$ which is analogous to the field code forest edit generation function $\text{FCF}_\omega$. In the field code forest version of the function $\mathcal{FM}$, we start with an ordering of the fields and work through a field code forest eliminating fields according to the field code forest. It seems likely that a field code forest version of $\mathcal{FM}$ is covering set correctible.

7.6.4 Extension of the formalisation to more general edits

There are two ways in which it would be useful to extend the formalisation. Firstly, it would be useful to consider disjunctions of linear inequalities such as

$$1 - 2x_1 + x_2 \geq 0 \text{ OR } x_1 + 4x_3 \geq 0,$$

which is formalised by arithmetic atoms as

$$q(1,-2,1,0) \text{ OR } q(0,1,0,4).$$

Secondly, it would be useful to formalise combinations of categorical and arithmetic edits, such as

$$1 - 2x_1 + x_2 \geq 0 \text{ AND } \text{has driver’s licence} = \text{yes}.$$

In order to deal with the first extension, that is, with disjunctions of linear inequalities, we would have to define a useful deduction function on clauses of arithmetic atoms. A deduction function that might be considered could incorporate resolution. However, if resolution is to be used, then negated arithmetic atoms need to be allowed. A consequence is that we would expand the repertoire of inequalities to include strict inequalities, using $>$, making things more complicated but perhaps more realistic.

In order to deal with the second extension, that is, the combination of categorical and arithmetic edits, we would have to create a logic which incorporates both types of propositional atoms introduced in this thesis, which are

1. the propositional atoms of the form $p^v_j$ introduced in Chapter 4; and
2. the arithmetic propositional atoms of the form $q_e$ introduced in this chapter.
While the logical formulae are easily constructed from the above propositional atoms, the deduction functions are less easily defined. Some work that may be relevant is the work done on combining logic systems and fibering logics, for example the work of Sernadas and Sernadas (2003) and Caleiro et al. (2005).

Several other authors have considered such generalised edits, but not from the point of view of a logical formalisation. They include Bruni (2004a), de Waal and Quere (2003), and Schaffer (1987).

7.7 Conclusion

This chapter has presented the beginnings of a theoretical logical framework for analysing the error localisation problem for arithmetic edits. The results parallel those of the earlier chapters, in particular Chapter 5, for categorical edits.

In addition, this chapter has examined the function $FM$, and its logical formalisation $FM$, which are based on the Fourier-Motzkin elimination method. Since $FM$ is covering set correctible, it is arithmetically strongly sound and arithmetically refutation complete, which are the two properties given by Farkas’ Lemma. In other words, the covering set correctibility of $FM$ is a strengthening of Farkas’ Lemma. Thus the connection between logic and arithmetic edits is an extension of the connection between soundness and completeness on the one hand and Farkas’ Lemma on the other hand.

The proof of the covering set correctibility of $FM$ depends on a lifting property in much the same way as the proof of the error correction guarantee of the function $FH$. Once again, the lifting property is fundamental.

The main aspects of this chapter are listed below.

1. The generation of new arithmetic edits can be seen as propositional logical deduction, where information implicit in a set of edits is extracted as conclusions.

2. The covering set method for error localisation is successful exactly when the arithmetic deduction function has covering set correctibility.

3. The property of covering set correctibility is a strengthening of arithmetic refutation completeness and arithmetic strong soundness.

4. The covering set correctibility of Fourier-Motzkin elimination is a strengthening of Farkas’ Lemma, and its proof depends on a lifting property.
Arithmetic edits
The covering set method is a method of trying to solve the error localisation problem, which itself is a step in the editing of erroneous data. While data editing is the process of correcting erroneous data, the error localisation problem is the problem of deciding which fields to correct in an erroneous record.

The covering set method consists of two steps. The first step is the generation of new edits from the explicit edits. The second step is the finding of covering sets.

The covering set method is not necessarily successful, in the sense that the covering sets that are output are not necessarily all and only error localisation solutions. The success of the covering set method depends on the edit generation function chosen for the first step of the covering set method.

The covering set method is guaranteed to be successful if the edit generation function used in the first step has the property of covering set correctibility, which means having both error correction totality and the error correction guarantee, defined as follows. The edit generation function has error correction totality when the corresponding covering set method outputs all error localisation solutions. The edit generation function has the error correction guarantee when the corresponding covering set method outputs only the error localisation solutions.

In this thesis we have formalised the two main aspects of the covering set method in terms of classical propositional logic, for both categorical edits and arithmetic edits. The two aspects are edit generation functions and covering set correctibility.

Firstly, edit generation functions can be formalised as logical deduction functions for propositional logic. In particular, the edit generation function FH turns out to be essentially the same as propositional resolution deduction.

The second aspect of the covering set method, namely the property of covering set correctibility, turns out to be a strengthening of soundness and refutation completeness. In particular, error correction totality is a strengthening of soundness, and the error correction guarantee is a strengthening of refutation completeness.

Since the error correction guarantee and refutation completeness are related, and FH and propositional resolution are related, it is not surprising that the proofs of the error correction guarantee of FH and of the refutation completeness of propositional resolution are also related. The two proofs depend on related properties: the lifting property for the error correction guarantee of FH and the reduction property for counter-examples for propositional resolution.
The lifting property comes into play in the proof of the error correction guarantee of another function, written as FCF\(_\omega\) and derived from the Field Code Forest Algorithm. Although the error correction guarantee of FCF\(_\omega\) has been questioned, this thesis gives a full proof of both the error correction guarantee and error correction totality of FCF\(_\omega\).

The parallel between the error correction guarantee and refutation completeness extends beyond the parallels between proofs. It extends to parallels between the problems these properties are connected to, namely the error localisation problem and the propositional satisfiability problem (SAT). In particular it extends to the solution of the two problems by a pure deduction method. In both cases, the pure deduction method succeeds when the deduction function used has special properties: in the case of error localisation the special property is covering set correctibility, while in the case of SAT the pure deduction method succeeds when the special property is refutation completeness and soundness. What is more, there are parallels between the deduction functions used to solve the two problems: two of the successful functions for error localisation, namely FH and FCF\(_\omega\), are respectively essentially the same as resolution deduction and ordered resolution which are two of the successful functions for SAT.

The existence of strong parallels between the error localisation problem and SAT means that the methods of solving SAT might be able to be extended to the methods of solving the error localisation problem, or vice versa. Since there is a vast range of techniques for solving SAT, there is hope that some of those techniques could be modified for the error localisation problem.

The results described above, formalising edit generation functions and covering set correctibility in terms of propositional logic, apply equally to categorical edits and arithmetic edits. The formalisation for arithmetic edits is different from that for categorical edits, but the same results nonetheless apply. The edit generation functions can be formalised as propositional deduction functions, and covering set correctibility is a strengthening of soundness and refutation completeness. In particular, the covering set correctibility of the edit generation function FM is a strengthening of Farkas’ Lemma.

There are many possible directions in which this work could be further developed:

1. The theoretical work is now almost ready to develop an implementation using propositional consequence finders.

2. The theoretical work could be extended to methods other than the pure deduction method for solving the error localisation problem.

3. The various details about arithmetic edits need to be tidied up and the work on arithmetic edits needs to be integrated with categorical edits.

4. Integer edits, which have characteristics both of arithmetic and categorical edits, could also be formalised.

5. Although this work has formalised data arranged in tables in terms of propositional logic, it could be that data arranged in relational databases could be formalised in terms of modal logic.
6. When the details of possible implementations are decided, then it will be necessary to assess their computational complexities.

This thesis has shown that there are many benefits of investigating the covering set method from the point of view of logic. Logic gives an alternative way of analysing the problem and thus potentially gives new insights. Logic also has a collection of sophisticated automated tools that could potentially be modified to use covering set correctibility for solving error localisation problems. Finally the strong parallels between propositional logic and data editing are of aesthetic appeal.
Conclusion


Index

2, 60
2, 73

$\Gamma_{(i_1, \ldots, i_k)}$, 171, 178, 183
$\Pi$ for Boolean algebras, 74
$\rho_k$, 195
$\sum$ for Boolean algebras, 74
$\chi$, 80
$\Omega_k$, 31
$[\alpha]_{Thy}$, 61
$\emptyset$, 14
$\equiv_{Thy}$, 61
$\boxslash$, 59
$\vdash$, 61
$\vdash_a$, 217
$\vdash_{Thy}$, 61
$\vdash$, 64

$A_k$, 72
$A_j$, 18
$A^j$, 21
ancestor, 174, 175
arithmetic atoms, 216
arithmetic deduction function, 219
arithmetic edit, 209
arithmetic edit generation function, 212
arithmetic propositional atoms, 216
arithmetic truth function, 216
arithmetically implies, 216
arithmetically refutation complete, 219
arithmetically refutation sound, 219
arithmetically satisfiable, 216
arithmetically strongly complete, 219
arithmetically strongly sound, 219

ba, 95
BH, 76
BoolAtm, 77
Boolean algebra, 72

Boolean algebra of edits, 93
Boolean atom, 77
Boolean homomorphism, 75
Boolean isomorphism, 75
BoolHom, 75
BranchS, 174
BranchV, 42

$C$, 23, 211, 218
categorical edit, 19
characteristic homomorphism, 80
clause, 63
clause, positive, 63
Clauses, 63
correct record, for arithmetic edits, 209
correct record, for categorical edits, 19
correction: error correction problem, 19
covering set, 23, 139, 211, 218
covering set correctible, 26, 140, 212, 220
covering set method, 25
covering set of failed edits, 23
$C\chi$, 23, 140, 211, 219

$D$, 18
data domain, 18
data domain, for arithmetic edits, 209
data editing, 2
deduction function, 58, 64
directional resolution, 65
domain, 18
domain, for arithmetic edits, 209
dominate, 19
domination rules for FCF$\omega$, 43

$E$, 93
ec, 96
edit, 4
edit generation function, arithmetic, 212
edit generation function, categorical, 26
editing, 2
$\mathcal{E}L$, 20, 139, 211, 218
empty clause, 63
end result of an FH-deduction, 111
end result of an $\mathcal{R}_{\text{Thy}}$-deduction, 65
ENFH, 34
entering, 23
error correction guarantee, 27, 140, 212, 220
error correction problem, 19
error correction totality, 27, 140, 212, 219
error localisation problem, 6, 20, 139, 210, 218
essential, 23
essentially new edit, 34
essentially new FH edit generation function, 34
essentially new FH-generated edit, 33
explicit edits, 25
F, 40
$\mathcal{F}$, 107
$f_v$, 97
failed edit, 23
failed record, for arithmetic edits, 209
failed record, for categorical edits, 19
fails, of a truth function, 60
Farkas’ Lemma, 230
FCF$_\omega$, 40, 42
$\mathcal{F}C\mathcal{F}_\omega$, 107, 202
FCF Algorithm, 40, 42
$\mathcal{F}C\mathcal{F}$ Algorithm, 202
FCFS$_\omega$, 162, 171, 173
FCFS Algorithm, 171, 173
FH, 30
FH edit generation function, 30
FH-deduction, 111
FH-generated edit, 29
FHD, 108
FHD1, 109
FHG, 29
field, 18
field, for arithmetic edits, 209
field code forest, 40
Field Code Forest Algorithm, 40, 42
field domain, 18
field domain, for arithmetic edits, 209
final clause, 65, 111
FM, 213
$\mathcal{F}M$, 220
formulae, propositional, 58
Fourier-Motzkin elimination, 213
$g_v$, 216
G1 to G7, 43
GenF, 41, 42
GenI, 41–43
GenI, properties, 43
GenS, 173, 174
GenS, properties, 174
GenV, 41, 42
GKL, 167
GS1 to GS7, 174
imputation, 6
incorrect record, for arithmetic edits, 209
incorrect record, for categorical edits, 19
inferior, 15
input set of clauses, 65, 111
involved, for arithmetic atoms, 218
involved, for arithmetic edits, 211
involved, for categorical edits, 23, 139
$\ell$, 63
LE$_{\text{Thy}}$, 62
$\mathcal{L}E_{\text{Thy}}$, 73
lifting property, 31
Lindenbaum algebra, 73
literal, 63
logically equivalent, 61
logically implies, 61
Max, 15
maximal essentially new FH edit generation function, 35
maximal FH edit generation function, 35
MENFH, 35
M, 107
MH, 35
MGKL, 167
Min, 15
model, 60
MRTh, 115
N, 18
NC, 103
nc, 99, 103
non-entering, 23
norm, 103
normal edit, 21
normal edit generation function, 26
normal form of an edit, 21
NRTh, 115
ordered resolution, 65
ordered Thy-resolution, 65
output clause, 65, 111
P, 14
pj, 89
ΨX, 73
Ψ′X, 73
partial order, 14
PAtm, 89
PC, 103
positive clause, 63
predecessor, 175
Prop, 58
PropAtom, 58
PropForm, 58
propositional atom, 58
propositional formulae, 58
propositional satisfiability problem, 70
propositional theory, 59
pure deduction method to solve SAT, 71
qe, 216
QAtm, 216
record, 18
record, for arithmetic edits, 209
reduced arithmetic edit, 221
reduced clause, 143, 144
reduction of a normal clause by a record
and a set of fields, 144
reduction of a set of arithmetic atoms
by a record and a set of fields, 222
reduction of a set of arithmetic edits by
a record and a set of fields, 222
reduction of a set of normal clauses by
a record and a set of fields, 144
reduction of an arithmetic atom by a
record and a set of fields, 222
reduction of an arithmetic edit by a record
and a set of fields, 222
refutation complete, 68
refutation sound, 67
refutation Thy-complete, 68
refutation Thy-sound, 67
resolution deduction function, 64
resolvent, 64
resultant clause, 65, 111
RThy, 64
RThy-deduction, 65
SAT, 70
satisfiable, 60
satisfies, of a record, for arithmetic ed-
its, 209
satisfies, of a record, for categorical ed-
its, 19
satisfies, of a truth function, 60
SC, 27
SCX, 27
SEL, 20
semantics, 57
sequentially generated, 16
smallest weighted covering set, 27
smallest weighted covering set correctible,
smallest weighted error localisation problem, 20
sound, refutation, 67
sound, strongly, 67
starting set of clauses, 65, 111
strongly complete, 67
strongly sound, 67
strongly Thy-complete, 67
strongly Thy-sound, 67
subfunction, 15
subsume, 63
sufficient, 162
superfunction, 15
superior, 15
syntax, 57

\top, 59
TF, 76
Th, 90
Th1, 90
Th2, 90
Tha, 217, 234
theory, propositional, 59
Thy, 60
Thy-equivalent, 61
Thy-logically equivalent, 61
Thy-logically implies, 61
Thy-resolution, 64, 65
Thy-satisfiable, 61
Thy-truth function, 60
Thy-valid, 61
ThyTF, 76
total order, 14
truth function, 60
\top \top, 102

uninvolved, for arithmetic edits, 211
uninvolved, for categorical edits, 23

valid, 61

\mathcal{X}, 23, 140, 211, 219

yields a correction, 20, 139, 210, 218