Integrating Planning and Scheduling : A Constraint-based Approach

Debdeep Banerjee

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

28th May 2015

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

Debdeep Banerjee

28^{th} May 2015

To P@trik and my family.

Abstract

Automated decision making is one of the important problems of Artificial Intelligence (AI). Planning and scheduling are two sub-fields of AI that research automated decision making. The main focus of planning is on general representations of actions, causal reasoning among actions and domain-independent solving strategies. Scheduling generally optimizes problems with complex temporal and resource constraints that have simpler causal relations between actions. However, there are problems that have both planning characteristics (causal constraints) and scheduling characteristics (temporal and resource constraints), and have strong interactions between these constraints. An integrated approach is needed to solve this class of problems efficiently.

The main contribution of this thesis is an integrated constraint-based planning and scheduling approach that can model and solve problems that have both planning and scheduling characteristics. In our representation problems are described using a multi-valued state variable planning language with explicit representation of different types of resources, and a new action model where each action is represented by a set of transitions. This action-transition model makes the representation of actions with delayed effects, effects with different durations, and the representation of complex temporal and resource constraints like time-windows, deadline goals, sequence-dependent setup times, etc simpler.

Constraint-based techniques have been successfully applied to solve scheduling problems. Therefore, to solve a combined planning/scheduling problem we compile it into a CSP. This compilation is bounded by the number of action occurrences. The constraint model is based on the notion of “support” for each type of transition. The constraint model can be viewed as a system of CSPs, one for each state variable and resource, that are synchronized by a simple temporal network for action start times. Central to our constraint model is the explicit representation and maintenance of the precedence constraints between transitions on the same state variable or resource.

We propose a branching scheme for solving the CSP based on establishing supports for transitions, which imply precedence constraints. Furthermore, we propose new propagation and inference techniques that infer precedence relations from temporal and mutex constraints, and infer tighter temporal bounds from the precedence constraints. The distinguishing feature of these inference and propagation techniques is that they not only consider the transitions and actions that are included in the plan but can also consider actions and transitions that are not yet included in or excluded from the plan.

We conclude the thesis with a modeling case study of a complex satellite problem domain to demonstrate the effectiveness of our representation. This problem domain has action choices that are tightly coupled with temporal and resource constraints. We show that most of the complexities of this problem can be expressed in our representation in a simple and intuitive way.

Contents

Chapter 1

Introduction

Automated decision making is one of the important problems of Artificial Intelligence (AI). In AI, planning and scheduling are two sub-fields that research automated decision making. Although the goal of these two fields is the same, generating plans to achieve goals, they do focus on solving different parts of the problem. The focus of planning is more on generality and reasoning about causalities between activities in a domain-independent way, while scheduling traditionally focuses on resolving time and resource constraints for a given set of activities where most of the causal relations are known. However, there are practical problems that have both planning and scheduling characteristics, and the interactions between these characteristics are very tight. To solve this class of problems efficiently we need to integrate planning and scheduling techniques.

AI planning problems are usually formulated with: an initial state, a goal state, and a set of actions. The task of a planning algorithm is to select and order a subset of actions , which is called a *Plan*, such that if we execute the selected actions according to their ordering starting from the initial state, we will reach the goal state. For example, consider a problem where we have to plan a journey to the airport from home. We can either catch a bus or a taxi to go to the airport. This means that there are two possible plans to achieve our goal (to be at the airport).

1. *Plan-1:* walk-to-bus-stop → get-in-bus → get-out-at-airport

2. *Plan-2:* walk-to-taxi-stand → get-in-taxi → get-out-at-airport.

Even though both plans are valid plans, they may fail during execution because these plans ignored the temporal and resource constraints. A plan where each action has a start time is called a *schedule* and the process of assigning start times to the actions is called *scheduling*. A schedule is executable if it satisfies all temporal and resource constraints. The following temporal information is given for our journey planning problem: we have to be in the airport within one hour, it takes 10 and 5 minutes to walk to the bus stop and taxi stand respectively from home. A bus takes 50 minutes to reach the airport and the next bus is due in 15 minutes time. A taxi will take 35 minutes to reach the airport. From this temporal information we can deduce that *Plan-1* is not valid any more, because we have to be in the airport with an hour but

Plan-1 will take at least 65 minutes (15 minutes before the next bus + 50 minutes traveling time). *Plan-2* remains valid, because we can reach to the airport before one hour using a taxi. Moreover, if we assume that the taxi would only accept cash as fare and we don't have any cash, then we need to the consider extra action of going to an ATM. In this case, validity of *Plan-2* would depend how far an ATM is from home and from the taxi stand. If this detour (home to ATM to taxi stand) takes more than 15 minutes, then *Plan-2* will become invalid.

Like in the example above, there are many real life problems (for example, consider the satellite problem described by Smith et al. in [47]) that have complex action choices (planning) and a set of time and resource constraints (scheduling), and the interaction between the consequences of action choices and the temporal and resource constraints is very tight. To make good executable plans (or schedules) for this class of problems we need to consider the causal constraints together with the temporal and resource constraints at the same time.

In this thesis we propose a integrated system for solving this class of planning problems that have tightly coupled planning and scheduling characteristics. In the remaining sections we first give a background on existing integrated planning scheduling systems. Then we will briefly outline the contributions of this thesis.

1.1 Integrating AI Planning and Scheduling

Problems that are in between planning and scheduling usually share some common complexities like time-windows, resource constraints, complex alternative choices of actions, sequence dependent setup times on resources between actions etc. Most of these complexities, except the action choices, can be seen as standard scheduling constraints. In general in these problems there is a subset of actions that only need to be performed at most once, while there are other actions that may have to be executed more than once. This characteristic of having a set of actions where each action occurs at most once has similarity with scheduling problems with alternative actions choices (multi-mode scheduling [7]). The action choices in multi-mode scheduling problems are generally due to availability of alternative resources and/or different action costs. There is a basic difference between these action choices in scheduling problems and the class of planning problems that we want to solve. In multi-mode scheduling the action choices are generally causally independent and limited to a small fixed set of actions, while problems in this "in-between" class often involve cascading sets of choices of actions that can interact with each other in a complex way and it is not computationally feasible to enumerate all valid choices beforehand [47]. This means that the interactions between action selection and ordering, and temporal and resource constraints are very strong in these problems.

There are mainly two types of modeling languages in the literature to model problems with planning and scheduling constraints: state/action-based representations, and timeline-based representations. The most commonly used state/action-based representation for model-

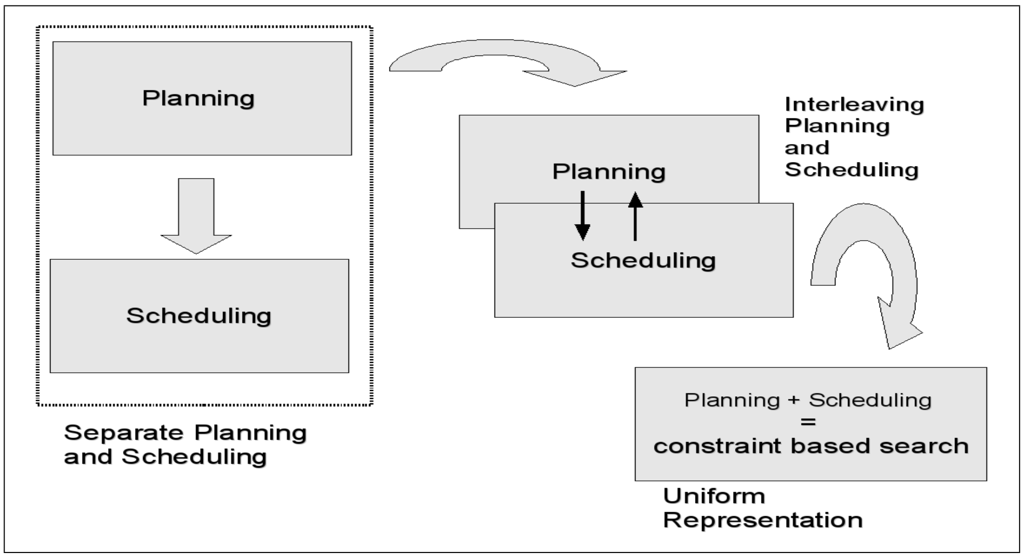

Planning
Scheduling
Separate Planning and Scheduling
Planning
Scheduling
Interleaving Planning and Scheduling
Planning + Scheduling = constraint based search
Uniform Representation

Figure 1.1: Planning scheduling integration

ing temporal and resource planning problems is PDDL2.1 [24]. PDDL2.1 can express most of the complexities of our class of problems. Problems in PDDL2.1 are described using state variables representing the state of the world, and actions with pre-conditions and effects whose execution changes one state to another. Timeline-based planning languages do not explicitly represent states or actions. Instead they keep track of evolution of each state variable and resource (i.e. timeline) by placing tokens on timelines. Each token represents a change within a time interval. Individual tokens on state variable and resources are synchronized via compatibility constraints (generally expressive temporal constraints). Another recently proposed planning language ANML [15] tries to combine the best of both timeline-based and state/action-based modeling languages. We will discuss the similarities and the differences between these languages and our representation in Chapter 2.

There are different ways to integrate planning and scheduling techniques in a system. Smith et al. [47] classify the most common integration approaches into three types, as shown in Figure 1.1.

At one end of the spectrum, we can treat planning and scheduling as two different processes. First the planning process finds a plan that will achieve the goal, and then the scheduler takes the initial plan and schedules the actions to satisfy the temporal and resource constraints. One major problem of this approach is that not all valid plans are schedulable given a set of temporal and resource constraints. So when the scheduler fails to find a schedule, a new plan has to be generated. This process will continue until a schedulable plan is generated. Although this approach is simple, it is not effective for solving problems that have tight interaction between planning and scheduling constraints.

The second possibility is to interleave planning and scheduling processes. This means that whenever the planner chooses an action to achieve a goal condition, the scheduler posts additional ordering constraints between the selected action and other actions already in the plan, to satisfy temporal and resource constraints. The majority of the integrated planning and scheduling systems are based on this idea. However, we can further distinguish these planners based on their degree of interleaving. One example of loosely interleaved planning and scheduling technique is the temporal planner CRIKEY [29]. It uses a classical planner (FF [31]) together with a simple temporal network (STN) [18] to generate a schedulable plan and then uses a scheduler that generates a better quality temporal plan. Planners like Zeno [41], IxTeT [28], COLIN [13], Filuta [20], etc are more tightly integrated (i.e these systems do not use a separate scheduling process to improve the quality of the plan) temporal planners based on the state-action representation. The emphasis of these planners is on generality of the representation and finding domain-independent heuristics to solve planning problems efficiently. Examples of planners that are based on timeline representations are HSTS [38], EUROPA [32], and OMPS [27]. Scalability is often a major problem for these systems. In most cases, it has been necessary to use domain-specific search control for them to achieve acceptable performance. However, recent work on domain-independent search control for timeline-based planners [6] may help overcome this problem in the future.

The third integration approach is to compile a problem with planning and scheduling characteristics into a constraint-based search problem. Examples of this approach include CPT [52], which takes a planning problem and converts it to a CSP [17], TM-LPSAT [45] which converts a problem to a combination of propositional logic and linear constraints, and CTN [43], which converts a planning problem described in a timeline-based representation to a CSP. The advantage of compiling a problem to constraint-based search is the availability of efficient solvers. In particular, converting to a CSP has the added advantage of being able to exploit the advancements in constraint-based scheduling [23] techniques. Note that planners that interleave planning and scheduling processes also use constraint-based scheduling techniques for temporal and resource reasoning. The difference between the compilation and interleaving approach is that in the latter approach planning and scheduling decisions are kept separate.

Constraint-based scheduling is popular because of its power of inference for deducing tighter bounds on temporal variables. There are two main categories of inference techniques: one is based on absolute temporal information and the other is based on relative temporal information. Examples of the first kind of inference techniques include Time-Tabling [4], Not-First-Not-Last [50], Edge-Finding [40] etc. Examples of the second class of inference techniques include the Balance and the Energy Precedence constraints [36]. Inference techniques based on relative temporal information are very important for solving problems with tightly coupled planning and scheduling constraints. The final set of activities on a schedule are not known beforehand in most these cases, so it is better not to commit on the values of the temporal variables, but maintain the precedence order between activities.

1.2 Overview of Contributions

The main contribution of this thesis is an integrated constraint-based planning and scheduling approach that can model and solve problems that have both planning and scheduling characteristics. Our approach is to compile a planning problem described in our state/action-based representation to a CSP. To solve this CSP efficiently we have developed a new branching strategy and several new propagation and inference techniques.

In our representation, problems are described using a multi-valued state variable planning language (like in, for example, SAS+[2]) with explicit representation of different type of resources and a new action model. In our action model each action is represented by a set of transitions, where each transition represents either an effect on a state variable or a resource requirement of the action. This action-transition model makes the representation of delayed effects and effects with varying duration of actions simpler. Chapter 2 presents the basics of our representation in detail. Complex temporal and resource constraints like time-windows, deadline goals, sequence-dependent setup times etc, can be expressed in this representation in a simple and intuitive way. In Chapter 5 we describe how these features are added on top of the basic representation. We demonstrate modeling of a complex satellite domain in our representation in Chapter 6.

A problem described in our representation is solved by compiling it into a CSP. This compilation is bounded by the number of action occurrences (similar to the planner CPT). The constraint model is based on the notion of "support" for each type of transition, where the meaning of "support" is similar to that of a *causal link* in partial-order planning (POP) [55]. In this thesis we extend the concept of causal links to resource transitions which are called "support links". The constraint model can be viewed as a system of CSPs, one for each state variable and resource, that are synchronized by a simple temporal network (STN) for action start times. Central to our constraint model is the explicit representation and maintenace of the precedence constraints between transitions on same domain object. Chapter 3 describes the compilation process and the constraint model.

We propose a branching scheme based on establishing causal and support links, which imply precedence constraints. Furthermore, we propose several new propagation and inference techniques that infer new precedence relations from temporal and mutex constraints, and infer tighter temporal bounds from the precedence constraints. The main feature of these inference and propagation techniques is that they not only consider the transitions and actions that are included in the plan but can also deduce new constraints for actions and transitions that are not yet included in or excluded from the plan. The power of our propagation techniques is as good as the inference techniques like the Energey Precedence and envelope-based techniques, for bounding temporal variables, and in addition they infer additional precedence constraints. Chapter 4 describes our branching, propagation and inference techniques.

1.3 Publications

Part of this thesis is published in the following two conference papers, and a workshop paper:

1. Debdeep Banerjee. **Integrating Planning and Scheduling in a CP Framework: A Transition-Based Approach**. In Alfonso Gerevini, Adele E. Howe, Amedeo Cesta, and Ioannis Refanidis, Editors, Proceedings of the 19^{th} International Conference on Automated Planning and Scheduling. AAAI Press, 2009.

2. Debdeep Banerjee and Patrik Haslum. **Partial-Order Support-Link Scheduling**. In Fahiem Bacchus, Carmel Domshalk, Stefan Edelkamp, and Malte Helmert, Editors, Proceedings of the 21^{st} International Conference on Automated Planning and Scheduling. AAAI Press, 2011.

3. Debdeep Banerjee, Jason J. Li. **Resource-based Planning With Timelines**, In the ICAPS Workshop on Planning and Scheduling with Timelines, PSTL 12, 2012.

The first paper describes the transition-based representation and compilation techniques for temporal planning problems only involving state variables. The second paper describes the resource transitions and the CSP model for solving the single mode Resource Constrained Project Scheduling Problems with min/max time lags (RCPSP/max). In this paper we introduced the support link-based branching and inference techniques for resource transitions. We compared our method of creating partial order schedules for RCPSP/max problems with two other approaches: a two stages approach that first finds a fixed point schedule and then lifts it to get a partial order schedule, and a envelope-based precedence constraint posting method. The third paper describes a case study on how to model and solve a factory problem in our transition-based framework.

Chapter 2

Modeling Planning and Scheduling Problems

A systematic way of studying a problem is to model the problem of interest in some formal specification and simulate different solving strategies by exploiting the specification. AI planning and scheduling problems are usually modeled using a set of interconnected components in a formal specification that describes the problem in terms of its components and relationships between the components in a precise (mathematical) way such that an automated solving mechanism (usually search) can be defined. A planning model usually reflects the reality of the problem domain. But to able to solve a problem, often we need to trade off between expressiveness of the model and the solution techniques' efficiency. The more expressive a model is, the harder it is to solve. So in general, a model is a sufficient approximation of the real world, such that we can solve these problems efficiently. Although modeling is an important part of solving problems, in reality it's quite hard and time consuming and sometimes impossible to model a problem close to reality. It will always have to be an approximation. There is now research on how to solve problem with incomplete models [33].

The problems that we want to model and solve in this thesis are the problems that are in between planning and scheduling problems. In this type of problems, things that needed to be modeled are the domain objects such as states of the components and resources, and how these objects interact with each other via actions. For the planning part we need to model the causal interaction between actions, that describes how one action can enable another action, and for the scheduling part we need to model the validity of these interactions by making sure none of the domain objects are in an invalid state at any point in time. In this thesis we assume a deterministic model of actions under the closed-world assumption, meaning that we know exactly the effects of each action and only actions change the state of components of the planning problem. All the numeric quantities in our model, like durations, resource requirements, capacities of resources, time etc. are integer values.

2.1 Background and Related Work

There are many representation languages to represent planning problems. The most commonly used representation languages in the planning community are PDDL [1] and its variants. PDDL1.2, provides language constructs for classical planning problems based on STRIPS, where actions have instantaneous effects. Successors of PDDL1.2 such as PDDL2.1 [24] extends to temporal planning via introducing durative actions and continuous numeric variables, and define semantics for concurrency and temporal constraints. Generally, AI planning models are based on a description of the world in terms of propositional and numeric variables and functions defined over them, and actions that have pre- and post-conditions that changes the world. Although PDDL is used widely in the planning research community (due to the fact that there are lot of example problem domains available, as it is being used as the official language for International Planning Competitions), its not easy to model practical problems mainly due to its propositional nature and lack of support for modeling different kind of resources and different temporal constraints that occur in many real world problems. Scheduling [9] problem models are generally described by available resources and durative activities that have different requirements on these resources. Alternative options to achieve goals are generally represented by different modes of action execution. In general scheduling lacks a representation language that can be used to model problems in a high level description.

Other attempts have been made to represent problems that are in between planning and scheduling. A concrete approach was introduced in HSTS[38], called HSTS-DDL. The main idea was to represent planning and scheduling problem as a dynamic system model. This method was inspired by approaches in control to synthesize valid behaviors of dynamic systems. The main difference between this approach and PDDL-type planning representations is that in this approach a planning problem is described as set of timelines instead of propositional state variables, and there is no concept of a global state or actions. A timeline represents the temporal evolution of some feature of a component of the problem domain over a planning horizon. Each feature can have more than one state that it can assume, but can be only in one state at any time point. Each timeline is described by a set of temporal intervals, called tokens. A token on a timeline describes the state of the feature, that the timeline represents, over the interval. Since a feature can be only in one state at any time, this means tokens on a timeline are totally ordered and there is no time-gap between the end of a token and the start of the next token. Tokens are constrained with other tokens (on the same timeline or on different timeline) via compatibility constraints. To solve a problem, the task is to allocate and order tokens on each timeline, such that all compatibility constraints are satisfied, and at the end of the planning horizon states of the timelines satisfy the goal. The timeline-based planning problem representation has been further extended by languages like DDL [11] and NDDL [32], that are mainly used in space applications. Modeling in these languages requires a very good understanding of the domain and the working of constraint-based solving strategies.

Recently, ANML [15] was introduced as a modeling language for planning and scheduling problems that tries to take best of both PDDL and timeline based representations. This language represents planning problem with multi-valued state variables (as in timeline-based languages) and has notions of actions and states (as in PDDL-based languages). It represents resources as bounded numeric fluents (like PDDL-based languages) and supports expressive temporal constraints (as in timeline-based languages). ANML supports both generative and HTN planning. Although it provides an expressive language to represent planning and scheduling problems, there is yet no planner that solves a planning problem described in ANML.

In this section we describe the representation that we will be using to model planning problems. This representation is based on the multi-valued state variable representation of planning problems, with the following extensions:

- In our representation we explicitly represent resources as domain objects. Usually in AI planning resources are represented as numeric fluents. In most real world problems, resources can be easily identified and they have structure. That means, resources in real world are more constrained w.r.t how they can be used than general numeric fluents. We would like to exploit this structure while reasoning about them by making different types of resources explicit in the representation.

- We represent actions as a set of synchronized durative effects, where these effects can have different durations. We call these durative effects of actions *Transitions*. This means that in our representation, instead of talking about the durations of actions, we talk about the durations of transitions of actions. In planning and scheduling representations, generally durations are associated with actions, which forces each effects of an action (or resource requirement of an activity in the scheduling case) to have same duration. The motivation behind our action representation is to allow modeling of a actions where effects can have different durations. For example, in a factory planning setting, lets say action *MAKE-P_1-M_1* represents making of *Product_1* in *Machine_1* and it takes 15 units of time to make the product. This means that *MAKE-P_1-M_1* needs *Machine_1*, which is a resource, for 15 units of time. It also needs a worker to configure *Machine_1*, but only for the first 5 units of time. To model this action either in PDDL or in standard scheduling model we need two actions (or activities), where the first one action requires a worker and *Machine_1* for 5 units of time, and produces the precondition of the second one, and the second action needs *Machine_1* and makes the product *Product_1* in 10 units of time. In our presentation this can be achieved using only one action with two different effects that start at the same time, but the first one requires the machine for 15 units of time, and the second one requires a worker for 5 units of time.

- For each transition we define a temporal constraint, which is a part of the transition description, that expresses the time delay between start of the transition and start of its action. This allows us to model delayed effect of actions.

- Each action effect (transition) is typed depending on if the transition is on a multi-valued state variable or on a resource.

The main aim here is to make the modeling task as simple as possible for the domain modeler and provide a flexible way to describe complex actions. Our representation is closely related to the ANML language. Like ANML we also describe a planning problem with multi-valued state variables, and have notion of action. The differences are:

- Our representation only supports generative planning, where ANML can also support HTN planning.
- Our representation explicitly represents resources as domain objects, where ANML describes resources as bounded numeric fluents.
- ANML provides temporal qualifiers that helps to represent expressive actions. In our representation we introduce a different action model where each action is described by a set of transitions. It our action model, actions can have only certain types of transitions, where each type has a clear semantics how they affect state variable and resources. Although this restricts our representation to express arbitrary effects of actions, it helps us to develop a solution technique that is described in the next chapter.

The planning language used by the partial-order planner IxTeT [28] is also closely related to our representation. In this representation, changes caused by actions are instantaneous. The problem with instantaneous change is that the domain modeler becomes responsible for ensuring the safe execution of actions on state variables and resources. We represent changes as transitions which can encapsulate complex durative processes of change. Especially on a resource, it is complicated to represent such complex durative processes using instantaneous resource events in a way that guarantees safe execution. This is also why ANML represents consumptions and productions of resources as transitions [15]. We believe that transitions make the domain modeler job easier to ensure the safe execution of actions (see Section 2.2.3.4).

2.1.1 Base representation and extensions

In the following sections we describe our planning problem representation in detail. First we describe the components of our base representation, which includes resources, state variables and actions. Then we describe the planning problem in terms of these components, initial states and goal description, and then discuss what we mean by a solution to a planning problem in the base representation. In Chapter 3 we will describe automatic compilation techniques that transforms the planning problem described in the base representation to a constraint satisfaction problem. To keep the base representation, and its compilation to a constraint model, simple, it does not include explicit representation of common scheduling constraints such as setup time, deadlines, time-windows etc. Instead, in Chapter 5 we will show how these constraints can be modelled as add-on constraints on top of the constraint model generated from the base representation. In Chapter 6 we will demonstrate the modeling of a complex satellite problem using the extended representation.

2.2 Components of a Planning Model

To model a planning problem, we need to model the planning world and actions that manipulate the world. By planning world we mean the collection of domain objects relevant to problem, that can be in different states. For example, for a factory planning problem, the planing world can consist of machines, workers, products, orders etc. Actions are the components that manipulate the planning world by changing one or more domain objects states. For example, making a product is an example of an action for factory planning that changes the state of the product from not-yet-made to made, and changes the state of a machine from available to not available during execution.

The planning model in our representation consists of three components: Resources, State Variables[1] and Actions. Resources and state variables are the domain objects that describes the state of the world. Resources and state variables evolve over time, meaning that the availability of resources and the state of state variables changes over time.

We view state variables and resources as *timelines* as described in control-based modeling of planning and scheduling problems. That means by timelines of state variable and resource we will mean their evolution over time in terms of states and resource availability.

2.2.1 Resources

Resources play an important role in most realistic planning and scheduling problems. Any object that enables an action to execute can be thought of as a resource. For example, in a factory planing problem machines, workers, raw materials to produce goods, and money are examples of resources. In general resources can be categorized in different types [5, 49] depending on if a resource can be consumed, produced, usage is either discrete or continuous, if the capacity of a resource changes over time etc. For example, raw material in a factory scheduling problem can be considered as a consumable resource as actions need to consume raw materials to produce goods, and in the process generate waste materials that can be considered as producible resource, which enables a cleaning action to execute. On the other hand, money can be seen

[1]In the rest of the thesis, a state variable will always mean a multi-valued state variable.

as a both producible and consumable resource as paying workers wages consumes money and selling products to customers produces money for the factory.

Resources that appear in real world problems can be categorized in the following two broad categories, based on what the way they can be accessed:

- **Reservoir Resource**: This is a resource that can be either consumed by actions at the beginning of execution or produced by actions at the end of execution. For example, consider an inventory that has finite capacity to store finished products in a factory. Each produced good needs to be stored before it can go out to a customer, where it consumes some space in the inventory, and when it is delivered to the customer it produces the same amount of space in the inventory.
- **Reusable Resource**: This is a resource that can be borrowed by actions at the beginning of their execution, and returned with the same amount at the end of the execution. An example of this kind of resource is a machine in a factory. When an product is being made in a machine, it borrows the machine until it finishes. No other products have access to the machine during this time.

In this thesis we assume all resources have constant integer capacity. This means that the capacity of resources does not change with time and can only be required by discrete integer amounts.

2.2.1.1 Resource Model

For each resource r, $\mathsf{capacity}(r)$ represents the integer capacity of resource r, which represents maximum amount of resource available at any point in time. Note that resources have constant capacity over time. If $\mathsf{capacity}(r) = 1$, we will call the resource *Unit-Capacity Resource*, otherwise we will call r a *Multi-Capacity Resource*. Let $\mathsf{type}(r)$ denote the type of the resource r which can be either *Reusable* or *Reservoir*. To reason about action execution, it is important to track the amount of resource available at any given time point. For any given time point t, let $\mathsf{level}(r,t)$ represent the amount of resource available to use at that time point. Note that $\mathsf{level}(r,t)$ is an non-negative integer and bounded within the range $[0, \mathsf{capacity}(r)]$. When $\mathsf{level}(r,t) = 0$ it means there is no resource available, and when $\mathsf{level}(r,t) = \mathsf{capacity}(r)$ it means resource is full. $\mathsf{level}(r,t)$ can be seen as the resource *profile* function of r, as defined by Cesta et al [12]. Similarly, let $\mathsf{free\text{-}space}(r,t)$ denote how much free space is there in the resource at a given time t. The variable $\mathsf{free\text{-}space}(r,t)$ is the complement[2] of the variable $\mathsf{level}(r,t)$. Note that, at any time point t the invariant $\mathsf{free\text{-}space}(r,t) + \mathsf{level}(r,t) = \mathsf{capacity}(r)$ holds. Since $\mathsf{free\text{-}space}(r,t)$ represents the complement of the $\mathsf{level}(r,t)$, $0 \leq \mathsf{free\text{-}space}(r,t) \leq \mathsf{capacity}(r)$ also holds. Figure 2.1

[2] Similar idea to the dual view of resources of Cushing and Smith[56].

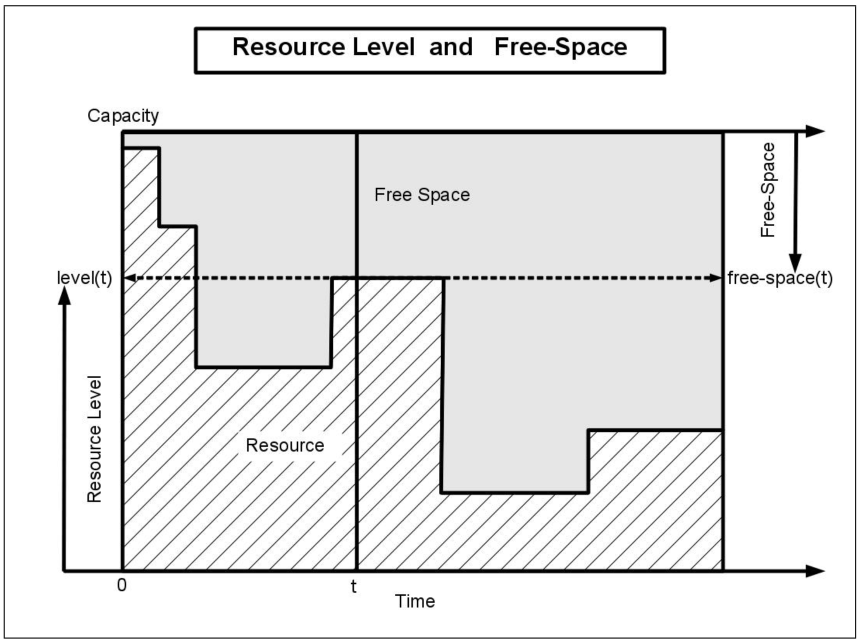


Figure 2.1: Resource Model

describes usage of a capacitate resource over time.

2.2.2 State Variables

State variables are the domain objects in the planning world that can be in one of many (finite) possible states at any given time. An action can either change states of a state variable from one to another, or require a particular state to hold during its execution. A simple example would be a light bulb, that can be in two possible states: On and Off. Figure 2.2 describes a transition graph of a state variable representing a light bulb. Nodes in the graph represent the possible states. Edges are labeled with actions that either change states or require a particular state. Action SWTICH-ON changes the state OFF to ON and similarly action SWITCH-OFF changes the state ON to OFF. Action DO-STUFF requires the state ON during its execution.

2.2.2.1 State Variable Model

For each state variable sv, $\mathtt{dom}(sv)$ represents the possible set of values (or states) that sv can assume. At any given time point t, let $\mathtt{state}(sv, t)$, denote the state of sv at t. $\mathtt{state}(sv, t)$ can be seen as the *timeline* function [43] of the state variable sv. At any time point t, sv can be either in one of its possible states or transiting from one state to another. The state of a state variable when it is changing states is *undefined* and denoted by $\emptyset$. For each state variable sv at any time point t, $\mathtt{state}(sv, t)$ has exactly one possible value, i.e. $\mathtt{state}(sv, t) = v$, where

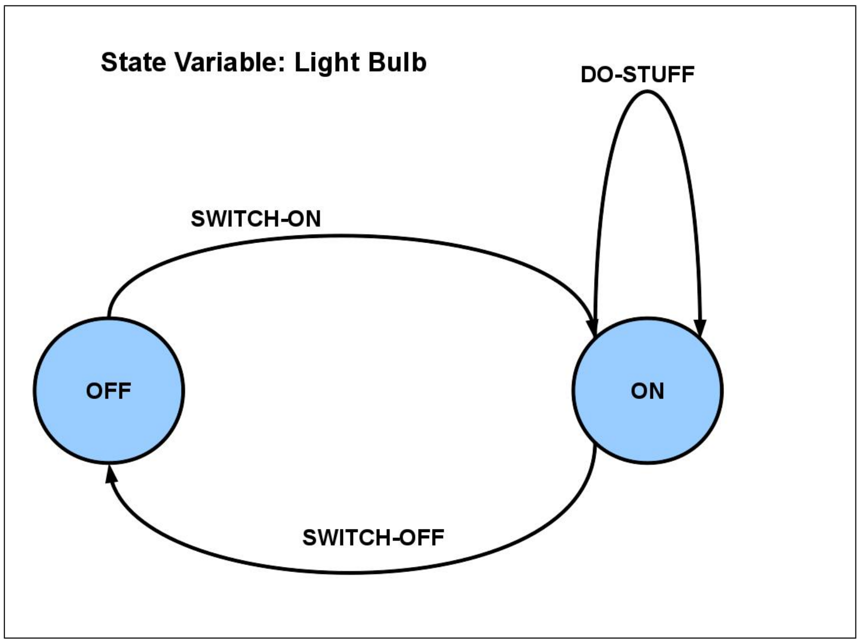


Figure 2.2: State Variable: Bulb

$v \in \mathrm{dom}(sv) \cup \{\emptyset\}$.

2.2.3 Actions and Transitions

Actions[3] are the components that manipulate the states of the domain objects. An action can change states of a subset of state variables from one state to another or requires a particular state, and consume, produce or borrow resources. An action can have effects on one or more state variables and resources simultaneously, and these effects can have different durations. This means that in our representation an action doesn't have duration, its effects have durations. Since we assign durations to the individual effects of actions, these effects become important reasoning entities on their own right. We call each effect of an action a *Transition*.

Before we describe transitions of action, we want to make distinction between an action and an action instance. An **action instance** is an occurrence of an action at a particular time. An action can occur more than once in a plan. For example, consider an action *Move(truck,A,B)*, representing moving a truck from location A to location B in some logistics domain. Now if we say the action *Move(truck,A,B)* starts at time point t, then we mean that an instance of the action *Move(truck,A,B)* starts at t. There may be another instance of that action that starts at other time point. Note that multiple instances of an action can start at the same time. For example consider an action *PRODUCE_COAL* that produces 10 units of coal from a coal mine. Now if

[3]By action we mean grounded action

our goal is to produce 30 units of coal, then we can execute 3 instances of *PRODUCE_COAL* action simultaneously. We always execute an action instance not an action. It means start time is only defined for an action instance, not for an action. When we talk about the start time of an action, we mean the start time of an instance of that action. Note that multiple instances of an action can have same start time. We consider each action instance as a unique entity.

We first describe transitions and how they affect evolution of the state variables and resource, then we describe the relationship between actions and its transitions.

2.2.3.1 Transition

A transition T is always associated with an action instance. For each transition T, let $\mathtt{act}(T)$ denote the action instance this transition is part of, $\mathtt{dur}(T)$ denote the duration of transition T, $\mathtt{req}(T)$ represent the requirement of transition T, $\mathtt{start}(T)$ and $\mathtt{end}(T)$ represent the start time and end time of T respectively. All transitions are non-preemptive, which means that they can't be stopped after they start executing. This means that the following relation holds:

$$\mathtt{end}(T) = \mathtt{start}(T) + \mathtt{dur}(T)$$

Depending on if a transition is executed on a resource or on a state variable, we define two main types of transitions: **State Variable Transitions** and **Resource Transitions**. In the following section we describe how state variable transitions and resource transitions are further categorized depending on how they affect state variables and resources.

2.2.3.2 Transitions on State Variables

Each state variable transition can be either an **EFFECT Transition** that causes a state change, or a **PREVAIL Transition** that represents a persistent state requirement on the corresponding state variable. In the following we describe these two types of transitions in details.

EFFECT Transitions: If a transition changes states of a state variable from one to another, we will call this transition an EFFECT transition. On each state variable sv, the requirement of each EFFECT transition T^E_{sv} is a pair of states of the state variable, i.e. $\mathtt{req}(T^E_{sv}) =< s_{from}, s_{to} >$, where $s_{from} \neq s_{to}$ and $s_{from}, s_{to} \in \mathtt{dom}(sv)$. It represents the fact that T^E_{sv} achieves the state s_{to} from the state s_{from}. The pre-condition of the EFFECT transition T^E_{sv} is the state s_{from}, denoted as $\mathtt{pre}(T^E_{sv}) = s_{from}$, and the post-condition is the state s_{to}, denoted as $\mathtt{post}(T^E_{sv}) = s_{to}$. As described before for each state variable sv, $\mathtt{state}(sv, t)$ describes the state of sv at time point t. If the EFFECT transition T^E_{sv} is executed on the state variable sv, the following three conditions must be satisfied.

1. At the start of the execution the state of sv must be the pre-condition of T^E_{sv}.

$$\text{state}\left(sv, \text{start}(T^E_{sv})\right) = \text{pre}(T^E_{sv}) \tag{2.1}$$

2. At the end of the execution the state of the state variable sv must be the post-condition of T^E_{sv}.

$$\text{state}(sv, \text{end}(T^E_{sv})) = \text{post}(T^E_{sv}) \tag{2.2}$$

3. During the execution at all intermediate time points between the start and the end of T^E_{sv} the state of the state variable must be undefined i.e. $\emptyset$. It represents that an EFFECT transition is executing on sv.

$$\forall t \;\; s.t. \;\; \text{start}(T^E_{sv}) < t < \text{end}(T^E_{sv}) : \text{state}(sv, t) = \emptyset \tag{2.3}$$

Figure 2.3 describes the effect of the execution of an EFFECT transition $T1$ on a state variable sv that has $\text{dur}(T1) = 6$ and achieves the state s' from s where $s, s' \in \text{dom}(sv)$. Transition $T1$ starts its execution at the time point t and finishes at the time point $t + 6$. The state of the state variable is s at time point t, and s' at the time point $t + 6$, and all intermediate time points (from $t + 1$ to $t + 5$) the state of the state variable sv is *undefined*. Note that if an EFFECT transition $T2$, on the state variable sv, has unit duration (i.e. $\text{dur}(T2) = 1$), and has the same requirement as $T1$, then there will be no intermediate time point where the state variable sv would be *undefined* as described in Figure 2.4.

PREVAIL Transitions: If a transition does not change states of a state variable, but instead requires a particular state for the duration of its execution, we will call this transition a PREVAIL transition. Each PREVAIL transition T^P_{sv} on a state variable sv has the requirement $\text{req}(T^P_{sv}) =< s >$, where s is a possible domain value of the state variable sv, meaning that the state variable must have the state s during the execution of T^P_{sv}. That means at all time points during the execution of T^P_{sv} the the state of sv must be $\text{req}(T^P_{sv})$.

$$\forall t' \;\; s.t. \;\; \text{start}(T_{sv}) \leq t' \leq \text{end}(T_{sv}) : \text{state}(sv, t') = \text{req}(T^P_{sv}) \tag{2.4}$$

Example: Recall the example of the state variable BULB given before. Now, consider that there are two actions: SWITCH-ON-1 and SWITCH-ON-2 that have one EFFECT transition each that changes the state of the BULB from OFF to ON. When the state of the bulb is changed from OFF to ON, then there are 3 possible explanations of the cause of the change: either EFFECT transition of SWITCH-ON-1 action was executed, or the EFFECT transition

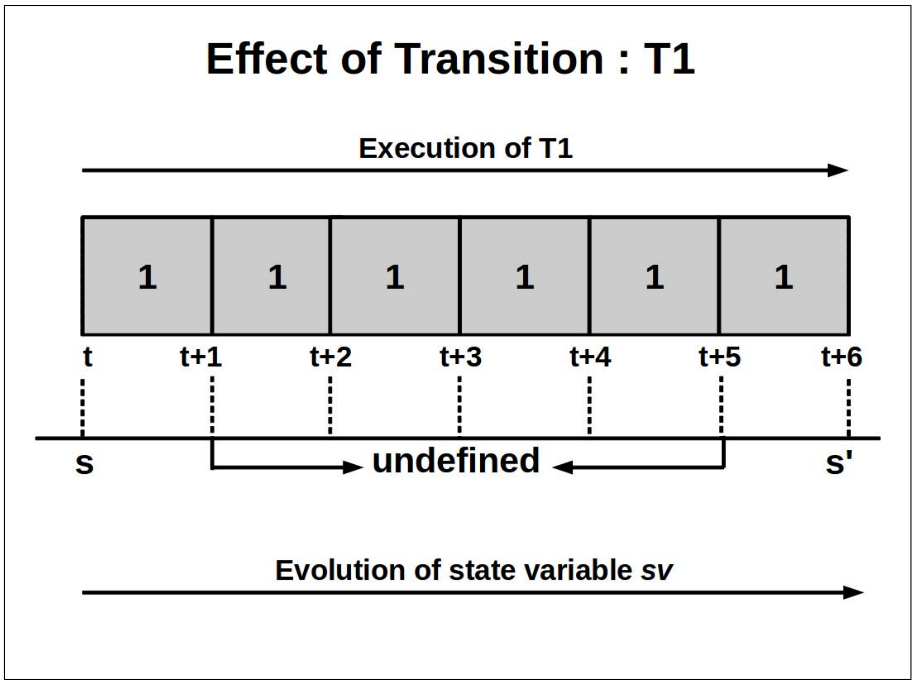


Figure 2.3: EFFECT Transition execution

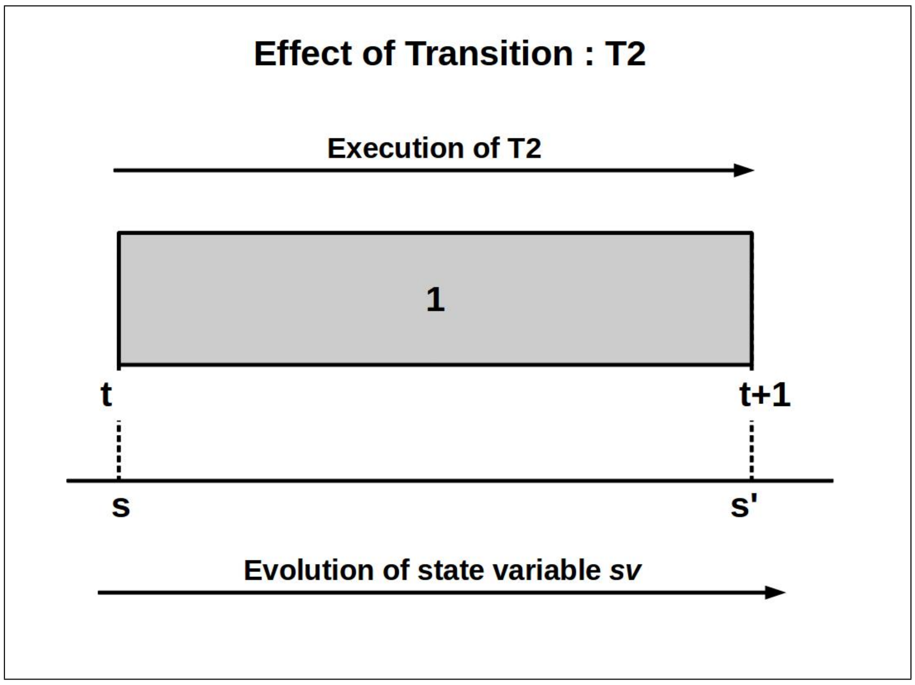


Figure 2.4: Unit Duration EFFECT Transition execution

of SWITCH-ON-2 action was executed or both EFFECT transitions of SWITCH-ON-1 and SWITCH-ON-2 are executed simultaneously. Since our planning model is deterministic, we assume that on each state variable, **only one EFFECT transition can change states of the state variable at any given time**. That means, if the bulb changes its state from OFF-to-ON, then either one of the EFFECT transitions of SWITCH-ON actions must be executed, not both. This assumption forces all EFFECT transitions that change states of the state variable to be totally ordered. Given a transition pair T_{sv} and T'_{sv} on the state variable sv, we say that they are totally ordered *iff* either T'_{sv} finishes its execution on sv before T_{sv} starts its execution or T'_{sv} starts after T_{sv} finishes, i.e. either $\mathtt{start}(T) \geq \mathtt{end}(T')$ or $\mathtt{start}(T') \geq \mathtt{end}(T)$ holds. On the other hand, a PREVAIL transition requires a state variable to be in a particular state during its execution. If there are other PREVAIL transitions that requires the same state, then they can be executed parallely. Since EFFECT transitions change state of state variables, each PREVAIL transition on a state variable, must be totally ordered with all EFFECT transitions on the state variable.

State Variable as Resource: Each state variable, where all transitions on a state variable have to be totally ordered, except for the PREVAIL transitions on the same state, can be viewed as discrete-state resource, as described by Smith et al [49]. In a unit-capacity resource, only one action can use the resource at time, meaning all actions that need the resource must be totally ordered in the final solution. In case of a state variable, at most one EFFECT transition can change the state of state variable at any time. In the final plan, all EFFECT transitions that change states of the same state variable must be totally ordered.

2.2.3.3 Transitions on Resources

For a resource r and a given time point t, $\mathtt{level}(r,t)$ represents the amount of resource available for use, and its dual $\mathtt{free\text{-}space}(r,t)$[4] represents the amount of free-space on the resource. The level of resources (also the amount of free-space) changes only via execution of transitions on resources. We describe the effects of transitions at each time point during their execution intervals on resources via three resource events : *production*, *consumption*, and *reservation*. Production and consumption events have their usual meaning, that is production (increament of level) and consumption (decreament of level) of resources. When we say *"a transition T **reserves*** $\mathtt{req}(T)$ *amount of free-space on the resource r at the time point t"*, we mean that at time point t the amount of free-space on r must be greater than or equal to $\mathtt{req}(T)$. That means:

$$reserve(req(T), t) \Rightarrow \mathtt{free\text{-}space}(r,t) \geq \mathtt{req}(T)$$

[4]Note that $\mathtt{level}(r,t) + \mathtt{free\text{-}space}(r,t) = \mathtt{capacity}(r)$ at any time point t.

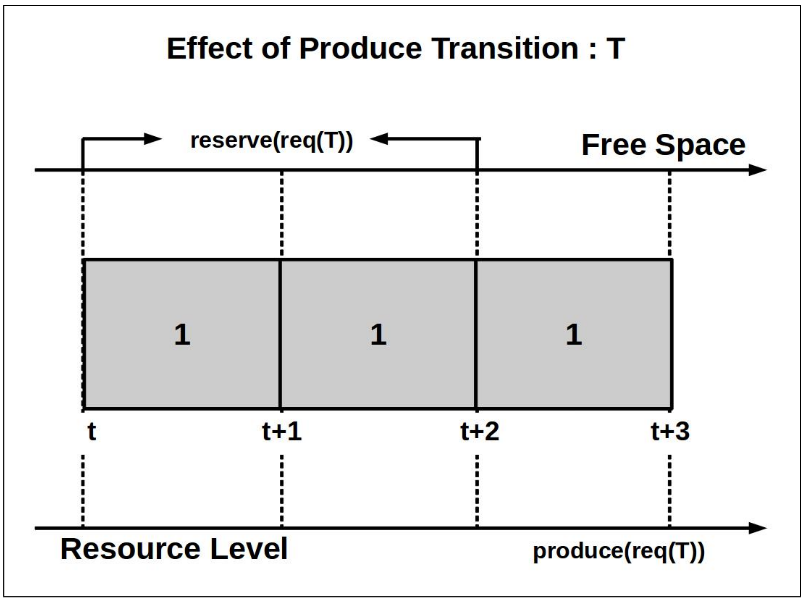


Figure 2.5: PRODUCE Transition execution

Note that a reservation request for a transition T at the time point of t doesn't change the amount of free-space on r but posts the above condition that must hold if T is executed on r. The reservation requests of transitions are additive. For example, if there exists two transitions T and T' such that at time point t they reserve $\mathrm{req}(T)$ and $\mathrm{req}(T')$ amount of free-space on r, then $\text{free-space}(r,t)$ must be greater than or equal to $\mathrm{req}(T) + \mathrm{req}(T')$.

In our representation, a resource r can be either a *reservoir* resource or a *reusable* resource. On reservoir resources, transitions can either produce or consume resource, and on resuable resource transitions can only borrow resource during their execution intervals. First we describe how transitions consume and produce resources on reservoir resources, and then based on the transitions defined on reservoir resources we describe how reusable resources are affected by the transitions on them.

On a reservoir resource r, a transition T_r can either produce or consume $\mathrm{req}(T_r)$ amount of resource during their execution. Note that production events increase the level of resource and consumes free-space, and similarly consume events decrease resource level and produces free-space. On each reservoir resource we define two types of transitions as following.

PRODUCE transition: Each PRODUCE transition T_r^P that starts its execution at $\mathrm{start}(T_r^P)$, *reserves* $\mathrm{req}(T_r^P)$ amount of free-space at each time point from $\mathrm{start}(T_r^P)$ to $\mathrm{end}(T_r^P) - 1$, and *produces* $\mathrm{req}(T_r^P)$ amount of resource at the time point $\mathrm{end}(T_r^P)$. Figure 2.5 describes the effect of a PRODUCE transition T that has duration 3. T *reserves* $\mathrm{req}(T)$ amount of free-space from the starting time point t to the time point $t+2$, and *produces* $\mathrm{req}(T)$ amount of resource at $t+3$.

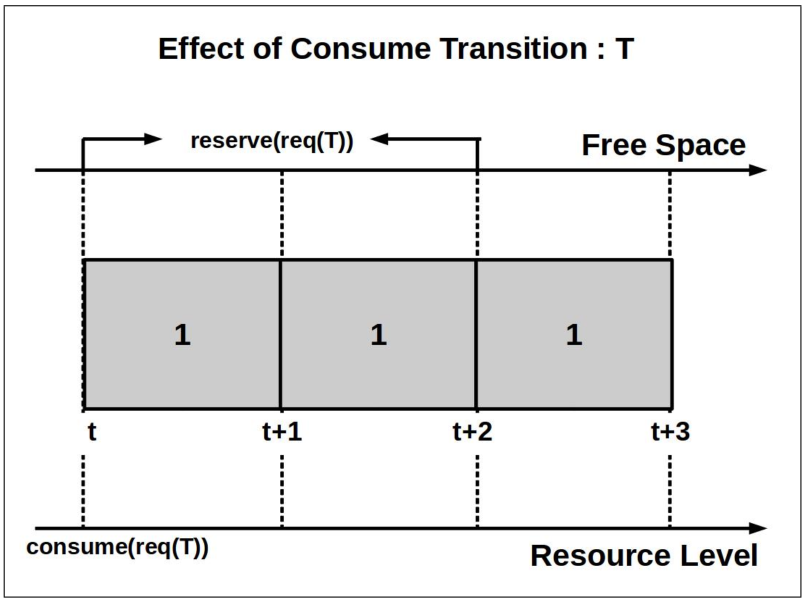


Figure 2.6: CONUSUME Transition execution

CONSUME transition: Each CONSUME transition T_r^C *consumes* $\mathtt{req}(T_r^C)$ amount of resource at the time point $\mathtt{start}(T_r^C)$, and **reserves** $\mathtt{req}(T_r^C)$ amount of free-space at each time point from $\mathtt{start}(T_r^C)$ to $\mathtt{end}(T_r^C) - 1$. Figure 2.6 describes the effect of execution of a CONSUME transition T that has duration 3. T *consumes* $\mathtt{req}(T)$ amount of resource at the starting time point t and *reserves* $\mathtt{req}(T)$ amount of free-space from the time point t to the time point $t+2$.

Transitions on reusable resources can't consume or produce resources separately as in reservoir resource. Transitions on reusable resources can only **borrow** resources during their execution. We define the type for transitions on reusable resources as the following:

BORROW transition: Each transition T_r^B on a reusable resource r is called a BORROW transition that borrows $\mathtt{req}(T_r^B)$ amount of resource at start and returns the same amount at the end. Each BORROW transition with duration d can be seen as a sequence of two (non-overlapping) consecutive transitions, where the first one is a CONSUME transition with duration $d-1$ and the second one is a PRODUCE transitions with unit duration. Each BORROW transition T_r^B *consumes* $\mathtt{req}(T_r^B)$ amount of resource at $\mathtt{start}(T_r^B)$, *reserves* $\mathtt{req}(T_r^B)$ amount of free-space at each time point from $\mathtt{start}(T_r^B)$ to $\mathtt{end}(T_r^B) - 1$, and at $\mathtt{end}(T_r^B)$ it *produces* $\mathtt{req}(T_r^B)$ amount of resource. Figure 2.7 describes the effect of a BORROW transition T that has duration 3. T *consumes* $\mathtt{req}(T)$ amount of resource at t, *reserves* $\mathtt{req}(T)$ amount of free-space from the time point t to the time point $t+2$, and *produces* $\mathtt{req}(T)$ amount of resource at $t+3$.

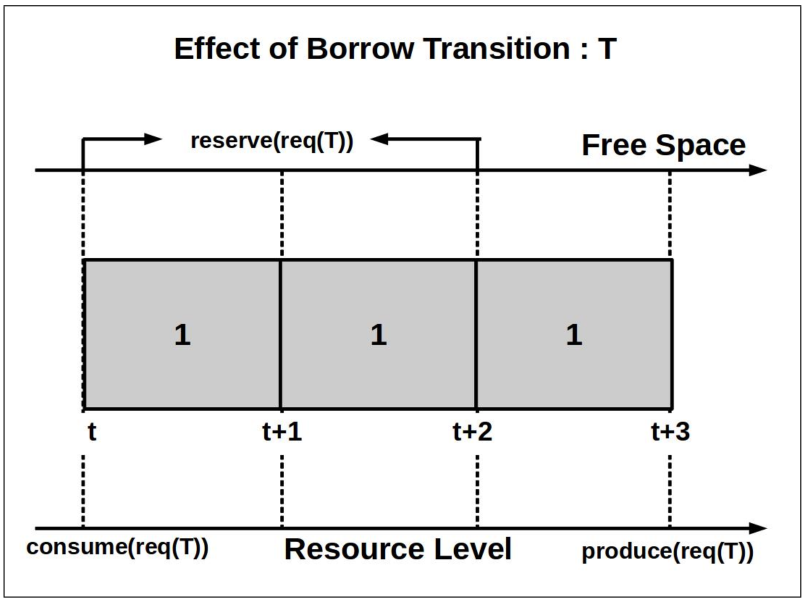


Figure 2.7: BORROW Transition execution

2.2.3.4 Safe execution of transitions on resources

Although transitions have positive non-zero durations, we assume that consumption events occur at the start of the transition execution and production events occur at the end of the transition execution and these events change the level of resource available. Note that at any time point t the level of a resource r must be within the capacity of the resource, i.e. $\forall t : 0 \leq \mathrm{level}(r,t) \leq \mathrm{capacity}(r)$. Each production event consumes free-space and each consumption events produces free-space on the resource. At any time point t, $\text{free-space}(r,t)$ that represents the amount of free-space on the resource at t, which is the dual of $\mathrm{level}(r,t)$. By allowing transitions to **reserve** free-space, we enable safe execution of the durative transitions on resources.

Definition 1. ***Safe Execution***
A safe execution of transitions on a resource r, is a execution where at each time point t, the invariant $0 \leq \mathrm{level}(r,t) \leq \mathrm{capacity}(r)$ *holds and the total reservation request for free-space on the resource is less than or equal to* $\text{free-space}(r,t)$.

Consider a reusable resource with capacity 4 that has three BORROW transitions $T1$, $T2$, and $T3$, where all of them require 2 units of resource, $T1$ and $T3$ have duration 3, and $T2$ has duration 2. Figure 2.8(top) describes an execution situation where $T1$ starts its execution at t, and $T2$ starts it execution at $t+1$. At start (time point 0) total resource available is 4. At time point t resource level is 2, because $T1$ consumes 2 units of resource at t, at $t+1$ $T2$ consumes the remaining 2 units of resource. At $t+3$, $T1$ and $T2$ finish their execution and produce 2

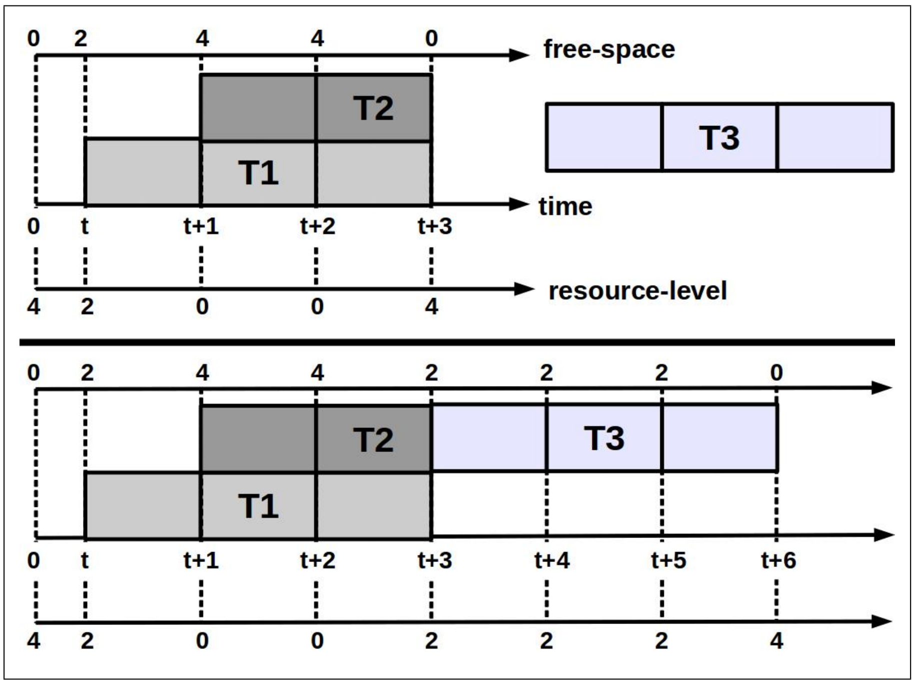


Figure 2.8: Safe execution on reusable resource

units of resource each. At each time point, total required free-space reservation is satisfied and at each time point sum of amount of free-space and resource level is 4 (capacity of the resource). It means that execution of $T1$ and $T2$ on the resource is a safe execution. Now if we want to execute $T3$ on the resource, we can see that if $T3$ starts at any time point between t and $t+2$, then the execution would not be safe. Because if $T3$ starts at any time point between t and $t+2$, then at $t+1$ and $t+2$ the total reservation of free-space would be 6 (each transition would reserve 2 units of free-space), which would be greater than the available free-space on the resource. So $T3$ can not start at any time point t', where $t-2 \leq t' \leq t+3$. The bottom part of the Figure 2.7 describes one possible safe execution of $T1$, $T2$ and $T3$ on the reusable resource, where $T3$ starts at $t+3$.

Figure 2.9 describes an example of transition execution on a reservoir resource. The reservoir resource has capacity 6, at start (time point 0) 3 units of resource is available for use, and a CONSUME transition Tc starts its execution at t and a PRODUCE transition starts at $t+1$. At t the level of resource reduced to 0 because Tc consumes 3 units of resource at start, and at $t+4$ Tp produces 3 units of resource. At the top part of the Figure 2.9 describes a safe execution of Tc and Tp on the resource. Given another PRODUCE transition $T'p$ that has duration 2 and produces 3 units of resource, we can see that the $T'p$ can start earliest at $t+3$ time point. $T'p$ can't start at $t+1$ or $t+2$ because there is no free-space available to reserve. Also note that since $T'p$ is a PRODUCE transition, it actually consumes free-space at the end

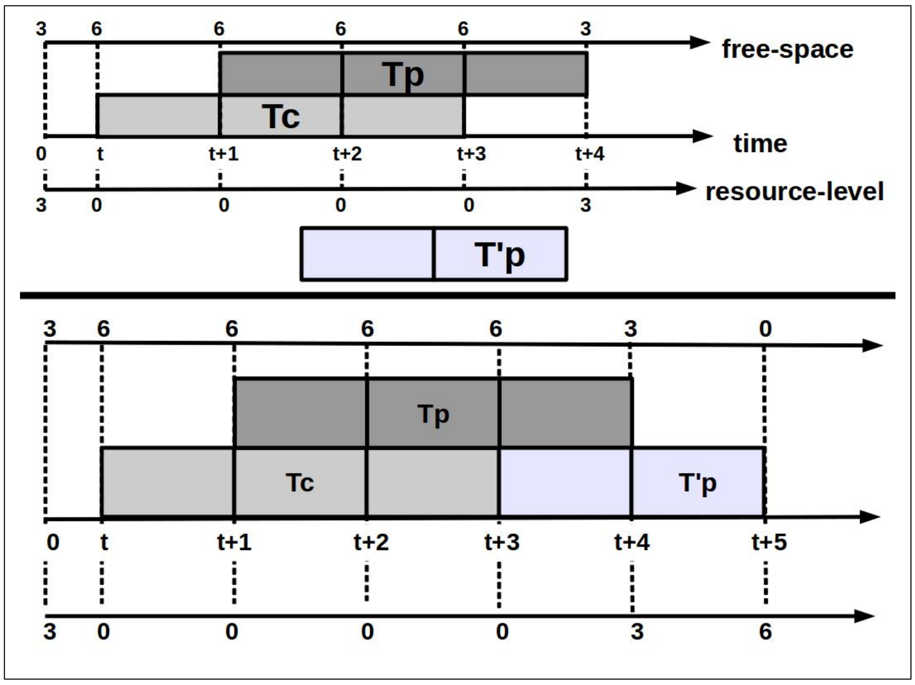


Figure 2.9: Safe execution on reservoir resource

when it produces resource. $T'p$ can't finish at $t+1$ or $t+2$, because then it will consume 3 units of free-space at those time points, and the execution of Tc and Tp will not be safe. If $T'p$ finishes at time point t or before, then amount of free-space in the resource will be 3. Since there are no other CONSUME transition (that can produce free-space) to execute on the resource, execution of Tc and Tp would become unsafe at $t+1$ and $t+2$. Which means that $T'p$ can't finish at any time point before $t+2$ (including $t+2$) such that the execution Tc and Tp remains safe. This means that $T'p$ can't start any time point before t (since $T'p$ has duration 2). So we can conclude on the reservoir resource the execution of transitions where $T'p$ start earliest at $t+3$ and produce 3 units of resource at $t+6$, is a safe execution as described in the bottom part of the Figure 2.9.

2.2.3.5 Conservative Modeling of Resource Transitions

In general, each resource transition of actions consumes or produces at a different or fixed rate during their execution. For example, consider two actions A_C and A_P that consumes and produces resource r respectively, where r is a reservoir resource with $\mathtt{capacity}(r) = 10$. Figure 2.10 describes the consumption of A_C and production of A_P on the resource r. Durations of both these effects are 4. Action A_C consumes resource at different consumption rate, during 0-1, it consumes 2 units, during 1-2 it consumes 1 unit, from 2-3 it consumes 5 units and 3-4 it consumes 2 units of resource. In total A_C consumes 9 units of resource during

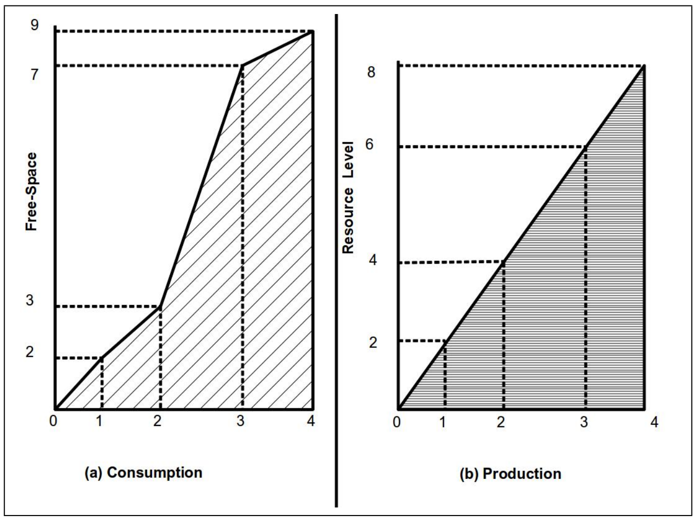


Figure 2.10: Conservative Modeling of Action Resource Usage

its execution. Similarly, action A_P produces total 8 units of resource at a fixed rate of 2 units per unit of time during its execution.

In our representation we model consumption effect of A_C as a CONSUME transition Tc that consumes the total consumption amount at the start of its execution, i.e. $\text{req}(Tc) = 9$. Similarly we model the production effect of A_P as a PRODUCE transition Tp that produces the total production amount at the end of its execution, i.e $\text{req}(Tp) = 8$. Both transitions Tc and Tp have duration 4. This way of approximating the effects of actions can be seen as **coarse-grained discretization**, where we lose the information about the consumption or production process at each time-step during the execution interval of transitions. Because of this coarse-grained discretization, we have the **conservative** assumption that each type of transition reserves free-space during its execution. This conservative assumption ensures that execution of transitions on a resource will be *safe*.

One disadvantage of the conservative assumption is that it delays the start of transitions, where the transition could be executed earlier time in real world on *reservoir* resources. For example, consider the execution situation described in Figure 2.11, where there are no other transitions except for Tc and Tp are executing on r, and at the beginning r is full, i.e. $\text{level}(r, 0) = \text{capacity}(r) = 10$. If we assume $\text{start}(Tc) = t$, then at t, 9 units of resource is being consumed by Tc. It means at t there exists 1 unit of resource and 9 units of free-space on the resource. Note that Tc reserves 9 unit of free-space until $t + 3$. For Tp to start it needs to reserve 8 unit of free-space. Since the free-space created by Tc is available at $t + 4$, earliest Tp can start is at the time point $t + 4$, where it reserves 8 unit of free-space from $t + 4$ to $t + 7$, and produces 8 unit of resource at $t + 8$. The level of available resource from 0 to $t - 1$ is 10, from t to $t + 7$ is 1 and from $t + 8$ onwards is 9.

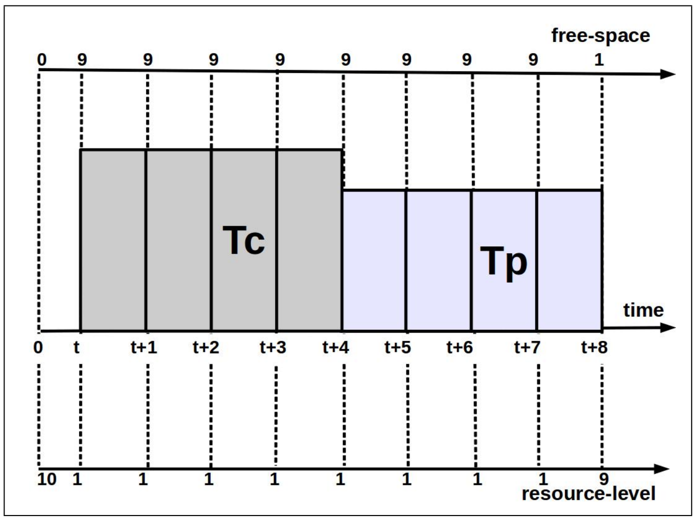


Figure 2.11: Coarse-Grained Discretization

If we look at the Figure 2.10, we can see if Tc starts executing at t, then earliest Tp can start executing is at the time point $t+2$ without overflowing or under-flowing the resource r. We will describe latter how can we achieve this under the same conservative assumption using our flexible action model that allows modeling delayed effects and multiple effects on same state variable and resource as described below. This enables discretization of the consumption and production effects of actions at a finer level.

2.2.3.6 Action-Transition Model

As stated earlier that actions have transitions on state variables and resources. Each action has a set (possibly empty) of transitions on each state variable and resource. Recall that, each transition is associated with an action instance. Here when we say that that an action a has a transition T, we mean that each instance of the action a have a transition T. For each action a, for each state variable sv and for each resource r, let $\mathsf{effect}(sv, a)$ and $\mathsf{effect}(r, a)$ represent the set of transitions that action a has on sv and r respectively. For each action a, let $\mathsf{trans}(a) = \mathcal{T}^a_{state} \cup \mathcal{T}^a_{res}$ represent the set of transitions of the action, where $\mathcal{T}^a_{state}$ represents the set of transitions on state variables, and $\mathcal{T}^a_{res}$ represents the set of transitions on resources.

$$\mathcal{T}^a_{state} = \bigcup_{\forall sv} \mathsf{effect}(sv, a)$$
$$\mathcal{T}^a_{res} = \bigcup_{\forall r} \mathsf{effect}(r, a)$$

Note that, if $T \in \mathsf{trans}(a)$, then $\mathsf{act}(T) = a$.

Each transition of an action a, $T^a \in \mathsf{trans}(a)$ has a non-negative offset value, $\mathsf{offset}(T^a)$

that describes the delay between start of the action a and start of the transition T^a. Start time of transitions are synchronized with the start time of their actions via these offset values. This means that, for each transition T^a, where $T^a \in \mathtt{trans}(a)$, the start time of T^a is related with the start time of a as the following:

$$\mathtt{start}(T) = \mathtt{start}(a) + \mathtt{offset}(T) \tag{2.5}$$

It means that after action a starts, transition T starts after $\mathtt{offset}(T)$ units of time.

In the PDDL-based representation each action has its duration and all effects take place either at start or at the end of the action. Our action model differs from PDDL-based action representation in two main ways: here actions do not have durations, instead the transitions of actions have durations, and the start of each transition can be delayed from the start of the actions, which enables us to model complex actions that appear in many real world problems in a straight-forward way. Consider the example of the turning of a spacecraft in order to point at a target, described by Smith [46]. To turn a space craft from one direction to other, the reaction control system (RCS), must fire the thrusters to provide angular velocity, then the spacecraft coasts until it points to the destination target, then the RCS thrusters are fired again to stop the angular motion of the spacecraft. It means the firing of the thrusters happens in the beginning and the end, and is controlled by the controller. Each time the thrusters are fired, propellants are consumed and it creates vibrations which may prevent some other operation on the spacecraft.

Assume that we have two state variables: *Pointing* that represents the location of the pointing device, whose domain consists of a finite set of targets, and *Vibration* representing the vibration status of the spacecraft which can take two possible values $\{V, S\}$, where V stands for "vibrating" and S stands for "stable". There are two resources: a multi-capacity reservoir resource *Propellant*, and an unit capacity reusable resource, *Controller*. In our representation we can model the **Turn** action of the space craft from *Target-1* to *Target-2* with the following sets of transitions. Each transition's name is followed by its offset value and duration.

$$\begin{aligned}
\mathtt{effect}(Pointing, Turn) &= \{T_1(0,25)\}\\
\mathtt{effect}(Vibration, Turn) &= \{T_2(0,3), T_3(7,3), T_4(20,3), T_5(27,3)\}\\
\mathtt{effect}(Controller, Turn) &= \{T_6(0,10), T_7(20,10)\}\\
\mathtt{effect}(Propellant, Turn) &= \{T_8(0,10), T_9(20,10)\}
\end{aligned}$$

Figure 2.12 describes the pictorial view of the transitions of the Turn action. Transition T_1 on the state variable *Pointing* changes the pointing location from *Target-1* to *Target-2* and takes 25 units of time, and starts at the same time when the action starts. Transitions T_2 and T_4 on the state variable *Vibration* change the vibration status from "stable" to "vibrating". Similarly,

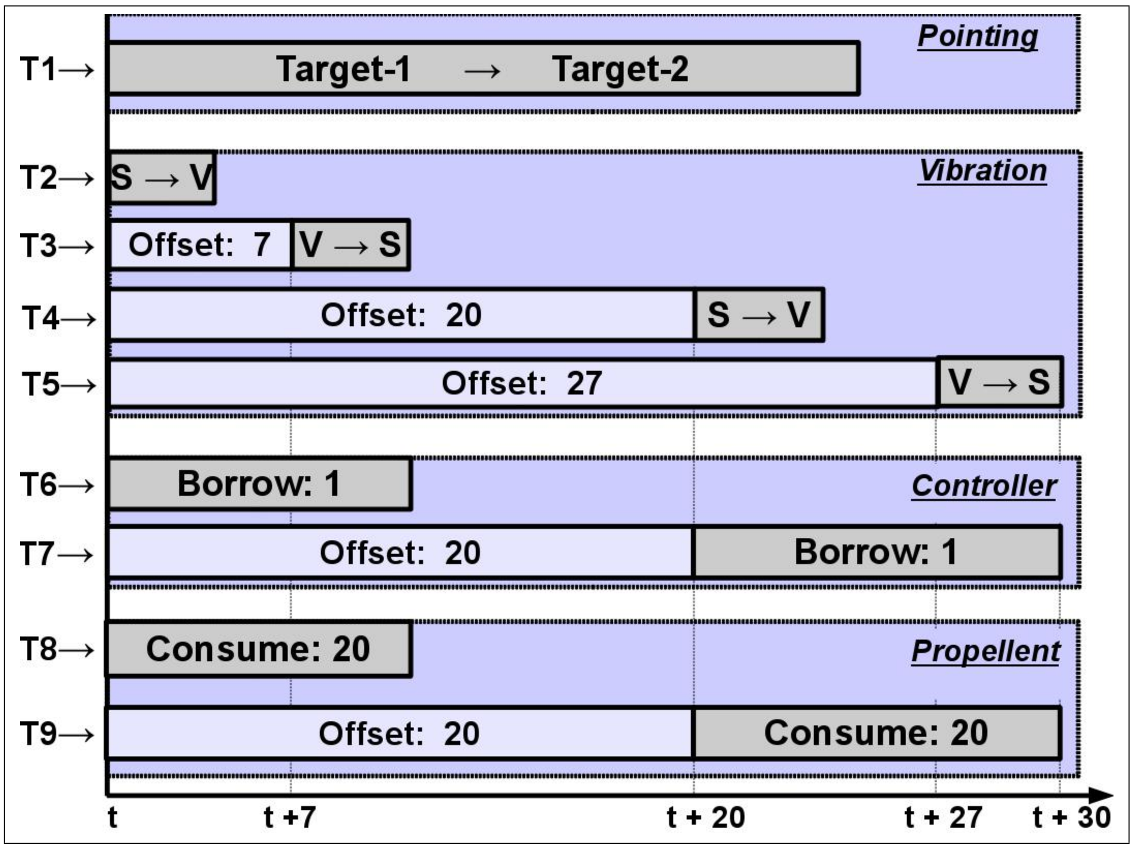


Figure 2.12: Transitions of Turn Action

transitions T_3 and T_5 change the the status from "vibrating" to "stable". Each transition on the state variable *Vibration* takes 3 units of time, and have 0, 7, 20, and 27 as their offset values respectively. On the resource *Controller* the *Turn* action has two borrow transitions T_6 and T_7, each borrows the *Controller* for 10 units of time. T_6 starts when the action starts, and T_7 starts 20 units of time after the action starts. On the reservoir resource *Propellant*, it has two consume transitions that consume 20 units of propellant while executing for 10 units of time. Note that T_7 on *Controller* and T_9 on *Propellant* starts after 20 units of time from the start of the action *Turn*.

In the PDDL2.1 durative action model, the assumption is that all the effects of an action, must happen either at start of the action or at the end of the action. Due to this assumption, as Smith has argued [46], complex actions that have intermediate effects on state variables and resources, like the Turn action described above, are not very easy to model with PDDL2.1. Our action model allows delays between the start of transitions and their corresponding actions that provides a straight-forward way to model complex realistic actions.

Valid Action Model: Recall that on each state variable all transitions must be totally ordered, except for PREVAIL transitions that require same state of the state variable. For each action we can assume without loss of generality that all its PREVAIL transitions on a state variable are non-overlapping, because if it has two PREVAIL transitions on different states, then those two PREVAIL transitions will be totally ordered, and if they are on the same state and overlapping then we can always model them as a single PREVAIL transition.

This means, that for an action model to be valid, for each state variable, all transitions of the action on the state variable must be sequenced, such that for each two transition T_1 and T_2, where T_1 appears before T_2 in the sequence, the following relation must hold.

$$\mathrm{offset}(T_2) \geq \mathrm{offset}(T_1) + \mathrm{dur}(T_1) \tag{2.6}$$

Similarly if an action has a set of overlapping resource transition on a resource, then total requirement of the overlapping transitions must be less than or equals to the capacity of the resource. Note that, if for an action the above conditions do not hold, then we can trivially say that the action will not be part of any solution.

2.3 Planning Problem Specification

A planning problem is defined as $\mathcal{P} =< R_{reuse}, R_{reserve}, SV, A, H, init, goal >$, where R_{reuse} and $R_{reserve}$ are the sets of reusable and reservoir resources respectively; SV is the set of state variables; A is the set of actions; H is the planning horizon; $init$ is the initial configuration of the problem, and $goal$ is the goal description. Each resource $r \in R_{reuse} \cup R_{reserve}$ has an integer capacity $\mathrm{capacity}(r)$. Each state variable $sv \in SV$ has a domain of states $\mathrm{dom}(sv)$. Each action $a \in A$ has a set of transitions (possibly empty) on each state variable and each resource, where each transition T is a 5-tuple

$$T :< \mathrm{obj}(T), \mathrm{type}(T), \mathrm{dur}(T), \mathrm{req}(T), \mathrm{offset}(T) >$$

where $\mathrm{obj}(T)$ represents the state variable or resource that T requires, and $\mathrm{type}(T)$ can assume one of the 5 types of transitions EFFECT, PREVAIL, BORROW, PRODUCE, and CONSUME. $\mathrm{dur}(T)$, $\mathrm{req}(T)$, and $\mathrm{offset}(T)$ have their usual meaning as defined before. Note that if a transition is a type of EFFECT, then $\mathrm{req}(T)$ is a pair of states, and if PREVAIL, then $\mathrm{req}(T)$ is a state from the domain of $\mathrm{obj}(T)$, which is a state variable. If T is a resource transition, then $\mathrm{req}(T)$ is a positive integer.

We assume that our planning horizon starts from time 0 and ends at H. Note that the horizon is the time by which all goals of the problem must be achieved. In general, the horizon can be set to infinity.

Initial Condition: The initial configuration of the problem $init$ defines the resource availability for each resource and the state of each state variable at the start. For each resource $r \in R_{reuse} \cup R_{reserve}$, $init(r)$ represents the amount of available resource initially, i.e. $0 \leq init(r) \leq \mathrm{capacity}(r)$. For reusable resources, since no transition can produce resource without consuming it first or consume it without producing it at the end, the capacity of resource must be same as the initial level of the resource throughout the planning horizon. This means that for each reusable resource r, $\mathrm{init}(r) = \mathrm{capacity}(r)$.

For each state variable $sv \in SV$, $\mathrm{init}(sv)$ denotes the initial state of sv, i.e. $\mathrm{init}(sv) \in \mathrm{dom}(sv)$. Initial state description is defined for all resource and state variables.

Goal Condition: For each resource $r \in R_{reuse} \cup R_{reserve}$ goal configuration $goal(r)$ defines the amount of resource left in r at the end of planning horizon H. Each $\mathrm{goal}(r)$ is an interval $[min_r, max_r]$, where

$$0 \leq min_r \leq max_r \leq \mathrm{capacity}(r)$$

It represents that at the end of the planning horizon the amount of resource in r must be atleast min_r and at most max_r. Note that, if for a resource r $min_r = 0$, and $max_r = \mathrm{capacity}(r)$, then it means that we don't care about the level of r at the end, in other words r does not have any goal. As discussed above, on reusable resources there is no transition that can produce or consume resource separately, all transitions borrow some resource. Since all transitions must finish their execution before the end of the planning horizon, at the end the total amount resource that will be left in a reusable resource is the total capacity of the resource. This means that for each reusable resource r, $\mathrm{goal}(r) = [\mathrm{capacity}(r), \mathrm{capacity}(r)]$.

Unlike resources, goal is defined for only a subset of state variables in the planning problem. For a state variable sv, $\mathrm{goal}(sv)$ denotes the state of the state variable sv that sv must have at the end of the planning horizon. We will call a state variable sv a **goal** state variable if $\mathrm{goal}(sv)$ is defined, otherwise a **non-goal** state variable.

Note that all states are a possible end state of a non-goal state variable sv. But there are cases when for a non-goal state variable, all states can be the potential end state except for some states. The notation $\mathrm{non\text{-}final}(sv)$ describes such a set of states that can not be the end state of the state variable sv.

2.4 Solution to Planning Problem

Traditionally scheduling problems are solved by assigning start times to the actions. Recently, focus on Partial Order Schedules (POS) [42] has increased. In a POS, instead of assigning fixed start times to the actions, new precedence relations are added to resolve resource contentions among the actions, where each precedence relation implies temporal constraint. Interest in POS can be attributed to the fact that POSs are more resilient to uncertainty than fixed time schedules, and can have longer execution life. To exploit the advantages of POS like solutions in planning, we define a solution to a planning problem in our representation as a **flexible plan**.

A flexible plan for a planning problem $\mathcal{P}$ given in the representation described in the previous section, is defined as the following:

$$FlexiPlan(\mathcal{P}) =< ActIns, StartTimeIntv, DistCons >$$

where $ActIns$ is a set of action instances, $StartTimeIntv$ is the set of start time intervals of each action instance in $ActIns$, and $DistCons$ is a set of difference constraints between each pair of start times of the action instances in Act. Each difference constraint is described as the following:

$$\mathtt{start}(b) - \mathtt{start}(a) \in dist(a, b)$$

Where $a, b \in ActIns$ and $dist(a, b) = [dist(a, b)_{min}, dist(a, b)_{max}]$ represents the minimum and maximum distance between $\mathtt{start}(a)$ to $\mathtt{start}(b)$ respectively. For all action instances in $ActIns$, start times are consistent with the constraints in $DistCons$. Each flexible plan represent the same structure as a consistent Simple Temporal Network (STN)[18]. A STN is a directed graph where nodes are time points and edges between nodes represents the distance between time points. If two time points t and t' have a constraint $Min \leq t' - t \leq Max$, then there exists an edge from t to t' with weight Max, and there exists an edge from t' to t with weight $-Min$. A STN is consistent if it does not have any negative weight cycle. Here the nodes in STN represent the action start times, and the edges represent the distance constraints between the actions start times.

2.4.1 Realization of a Flexible Plan

In a flexible plan for a planning problem, start times of actions are intervals, representing the possible time windows within which actions can start execution. While executing, the plan execution engine chooses a fixed time point for each action from its start time window to execute the action. Given a flexible plan, we define **realization** of the flexible plan as the following.

Definition 2. ***Realization of a flexible plan***
Given a flexible plan for a planning problem $\mathcal{P}$, $\mathrm{FlexiPlan}(\mathcal{P})$, a realization, denoted by $R(\mathrm{FlexiPlan}(\mathcal{P}))$, is an instantiation of the start times of the actions in the flexible plan with particular values from the intervals, such that all distance constraints of the flexible plan are satisfied.

This means that each flexible plan $\mathrm{FlexiPlan}(\mathcal{P})$ represents a set of realizations, where each realization is a concrete possible execution of the plan.

Given a flexible plan, for simplicity we will denote a realization of a flexible plan as R. We will denote a fixed start time as $\mathtt{start}[*]$ and the start time interval $\mathtt{start}(*)$ for both actions and transitions. We call each transition and action instance a time-stamped transition and action instance if their start times are fixed. Note that, each realization of flexible plan is a set of time-stamped action instance from the $\mathrm{FlexiPlan}(\mathcal{P})$. It means each action instance $a \in ActIns$ has a fixed start time $\mathtt{start}[a]$ such that $\mathtt{start}[a] \in \mathtt{start}(a)$. Since an action instance and its transitions start times are synchronized, in each realization transitions of the time-stamped action instances have a fixed start time, and also fixed end times, because transitions are uninterruptable. Given a transition T, $\mathtt{start}[T]$, a nd $\mathtt{end}[T]$ denote the fixed start and end time of T respectively, where $\mathtt{start}[T] = \mathtt{start}[\mathrm{act}(T)] + \mathrm{offset}(T)$ and $\mathrm{act}(T) \in ActIns$.

Each realization R, creates a schedule on each state variable and resource, where each **schedule** is a set of transitions of the action instances in the realization.

Definition 3. ***Schedule on State Variables and Resources***
Given a realization R, schedule on a state variable sv is a set of time-stamped state variable transitions and schedule on a resource r is a set of time-stamped resource transitions, such that for each transition T in these sets, the corresponding action instance $\mathrm{act}(T)$ is included in the flexible plan.

$$schedule(sv, R) = \{(T, t) |\, \mathrm{obj}(T) = sv, \mathrm{act}(T) \in ActIns, \mathtt{start}[T] = t\}$$

$$schedule(r, R) = \{(T, t) |\, \mathrm{obj}(T) = r, \mathrm{act}(T) \in ActIns, \mathtt{start}[T] = t\}$$

2.4.2 Valid Schedule of a State Variable

Given a schedule of a state variable sv for a realization R, $schedule(sv, R)$, let $schedule(sv, R)^E$ represent the subset of $schedule(sv, R)$, where all transitions are EFFECT transitions, and similarly, $schedule(sv, R)^P$ represent the subset where all transitions are PREVAIL transitions. Note that these two set are disjoint, i.e.

$$schedule(sv, R) = schedule(sv, R)^E \cup schedule(sv, R)^P$$

For each state variable sv, $\mathtt{state}(sv, t)$ describes the state of the state variable at time point t. A state variable at any time point must be either in one of its possible states defined in $\mathtt{dom}(sv)$ or $\emptyset$ which represents the transition of state variable from one state to other. Given a realization R, **evolution** of a state variable over an interval $[t, t']$ describes how the state variable changes its states over the interval.

Definition 4. ***Evolution of a State Variable***
For each state variable sv, $\mathrm{evolution}(sv, [t, t'])$ represents the evolution of the state variable sv as a set of $\mathtt{state}$ values for sv at each time point in $[t, t']$. That means:

$$\mathrm{evolution}(sv, [t, t']) = \{\mathtt{state}(sv, t'') | t \le t'' \le t'\} \tag{2.7}$$

Where $\forall t'' : \mathtt{state}(sv, t'') \in \mathtt{dom}(sv) \cup \{\emptyset\}$.

Constructing the Evolution on a State Variable: Recall that only EFFECT transitions change states of state variable from one to other during its execution. Given an EFFECT transition T^E_{sv} that is in the schedule of the state variable sv, i.e. $T^E_{sv} \in \mathit{schedule}(sv, R)^E$, at the start of T^E_{sv}, the state of the state variable sv must be $\mathtt{pre}(T^E_{sv})$, and at the end of its execution the state of sv must be $\mathtt{post}(T^E_{sv})$. If $\mathtt{dur}(T^E_{sv}) > 1$, then at the each time point from $\mathtt{start}[T^E_{sv}] + 1$ to $\mathtt{end}[T^E_{sv}] - 1$ the state of sv must be $\emptyset$, denoting the intermediate state while sv is transitioning from $\mathtt{pre}(T^E_{sv})$ to $\mathtt{post}(T^E_{sv})$. Given a schedule on a state variable sv, $\mathit{schedule}(sv, R)$, we can construct the evolution of the state variable sv for the interval $[0, H]$ as the following:

- At time point $t = 0$, the state of sv must be the initial state $\mathtt{init}(sv)$. Note that for each state variable the initial state is always defined.

$$\mathtt{state}(sv, 0) = \mathtt{init}(sv) \tag{2.8}$$

- Since on each state variable all EFFECT transitions are totally ordered, at any given time point t such that $t > 0$, then there must be at most one EFFECT transition that either executing at t or ends at t. For all time point t, where $0 < t \leq H$, the state of sv is defined by exactly one of the following cases:

  Case-I If there exists an EFFECT transition $T^E_{sv} \in \mathit{schedule}(sv, R)^E$, such that T^E_{sv} is ending at t, i.e. $\mathtt{end}[T^E_{sv}] = t$, then the state of sv must be $\mathtt{post}(T^E_{sv})$ at t.

$$\mathtt{state}(sv, t) = \mathtt{post}(T^E_{sv})$$

  Case-II If there exists an EFFECT transition $T^E_{sv} \in \mathit{schedule}(sv, R)^E$, such that it started its execution at the previous time point, meaning $\mathtt{start}[T^E_{sv}] \leq t - 1$, and its not ending at t, $\mathtt{end}(T^E_{sv}) > t$, then the state of sv at t must be the intermediate state $\emptyset$ at t.

$$\mathtt{state}(sv, t) = \emptyset$$

  Case-III If none of the cases above holds at the time point t, then due to the assumption of inertia, the state of sv stays same as the state at the previous time point.

$$\mathtt{state}(sv, t) = \mathtt{state}(sv, t - 1)$$

Note that the evolution of a state variable created from a given realization always achieves the initial condition of the planning problem. We will call a schedule on a state variable a **valid schedule** if it produces a evolution for the state variable, and the evolution will satisfy the pre-conditions of EFFECT transitions and overall conditions of PREVAIL transitions in the schedule and achieves the goal conditions if specified for the state variable.

Definition 5. ***Valid Schedule on State Variable***
*For each state variable sv, if a given schedule, $schedule(sv, R)$, created by realization R, satisfies the following three conditions, then we call it a **valid** schedule.*

- *All EFFECT transition pairs T^E_{sv} and $T^{E'}_{sv}$, in $schedule(sv, R)^E$, are totally ordered, i.e. either $\text{start}[T^E_{sv}] \geq \text{end}[T^{E'}_{sv}]$ or $\text{start}[T^{E'}_{sv}] \geq \text{end}[T^E_{sv}]$. This condition ensures the creation of a correct evolution of the state variable.*

- *Given the correct evolution of sv the pre-conditions of the EFFECT transitions in the schedule must be satisfied, i.e. $\forall T^E_{sv} \in schedule(sv, R)^E$:*

$$\text{state}(sv, \text{start}[T^E_{sv}]) = \text{pre}(T^E_{sv}) \tag{2.9}$$

  For all PREVAIL transitions in the schedule, the overall conditions must hold for each time point their execution, i.e. $\forall T^P_{sv} \in schedule(sv, R)^P$,

$$\forall t \in [\text{start}[T^P_{sv}], \text{end}[T^P_{sv}]] : \text{state}(sv, t) = \text{req}(T^P_{sv}) \tag{2.10}$$

- *If the state variable sv has a goal condition $\text{goal}(sv)$ defined in the goal description of the planning problem, then the final state of sv must satisfy the goal condition.*

$$\text{state}(sv, H) = \text{goal}(sv) \tag{2.11}$$

- *Each non-goal state variable sv must have an end state that is not included in their $\text{non-final}(sv)$ set.*

$$\text{state}(sv, H) \notin \text{non-final}(sv) \tag{2.12}$$

2.4.3 Valid Schedule of a Resource

There are two types of resource in our planning representation, reservoir and reusable resource. The main difference between these two type of resource is the way they can be accessed. On a reservoir resource only two types of transitions can be executed: PRODUCE transitions that produce resource at end of their execution and CONSUME transitions that consume resource at start of their execution. On a reusable resource only BORROW transitions are allowed to execute, where each BORROW transition can be described as a sequence of a CONSUME and a PRODUCE transition (see BORROW Transition in Section 2.2.3.3). Since each transition's behavior can be described via three types of resource events, production, consumption and reservation, we describe the effects of transitions on a resource (either reservoir or reusable) in a unified way in this section, where we do not distinguish between reservoir or reusable resource.

For each resource r, given a schedule, $schedule(r, R)$ for the realization R, let the subset $schedule(r, R)^C$ contains the transitions that consume resource r at the start of their execution, and $schedule(r, R)^P$ contains the transitions that produce resource r at the end of their execution. Note that if r is a reusable resource then all transitions in $schedule(r, R)$ are of type BORROW, which consumes at the start and produces same amount at the end. This means that in the case of reusable resource, sets $schedule(r, R)^C$ and $schedule(r, R)^P$ contain the same sets of transitions. If r is a reservoir resource then $schedule(r, R)^C$ and $schedule(r, R)^P$ are disjoint sets containing the CONSUME and PRODUCE transitions respectively. For any resource r,

$$schedule(r, R) = schedule(r, R)^C \cup schedule(r, R)^P$$

Given a schedule for a resource r, let $usage(r, t)$ represent the total amount of resource consumption at time point t which is defined as the total consumption minus the total production at t. Note that consumption events occur at the start and production event occur at the end.

$$usage(r, t) = \sum_{\substack{T_r^C \in schedule(r,R)^C \\ s.t.\, \mathrm{start}[T_r^C]=t}} \mathrm{req}(T_r^C) - \sum_{\substack{T_r^P \in schedule(r,R)^P \\ s.t.\, \mathrm{end}[T_r^P]=t}} \mathrm{req}(T_r^P) \tag{2.13}$$

For each resource r, $\mathrm{level}(r, t)$ describes the amount of resource available at the time point t. Given a realization R, similar to the evolution of state variable, a **evolution** of the resource over an interval $[t, t']$ describes the resource availability over the time interval.

Definition 6. ***Evolution of a Resource***
For each resource r, $\mathrm{evolution}(r, [t, t'])$ *represents the evolution of the resource r as a set of* level *values for r at each time point in* $[t, t']$. *That means:*

$$\mathrm{evolution}(r, [t, t']) = \{\mathrm{level}(r, t'') | t \le t'' \le t'\} \tag{2.14}$$

Where $\forall t'' : 0 \le \mathrm{level}(r, t'') \le \mathrm{capacity}(r)$.

Constructing the Evolution on a Resource: Given a schedule on r, $schedule(r, R)$, we can construct the evolution of r over planning horizon $[0, H]$ as the following:

- At time point $t = 0$, the level of resource is initial amount available of resource level minus the resource usage at 0.

$$\mathrm{level}(r, 0) = \mathrm{init}(r) - usage(r, 0) \tag{2.15}$$

- At each time point t, where $0 < t \le H$ the level of resource is calculated as the amount of resource available at $t-1$ minus the total consumption at t. Note that, negative consumption means production.

$$\text{level}(r,t) = \text{level}(r,t-1) - \mathit{usage}(r,t) \tag{2.16}$$

Given any time point t, where $t \geq 0$, we can determine the value of $\text{level}(r,t)$ using the above two rules. Note that given a realization R each transition starts at or after time point 0. Since all transitions have non-zero positive duration, and only CONSUME transitions consume resource at start, at time point 0 for each resource r the value of $\mathit{usage}(r,0)$ will be always non-negative. This means that in the evolution created from the realization R for each resource the level of the resource at time point 0 always be less than or equal to the initial level, i.e. $\text{level}(r,0) \leq \text{init}(r)$, i.e. each *evolution* of a resource respects the initial condition on the resource.

For each resource r since at each time point t, $\text{level}(r,t) + \text{free-space}(r,t) = \text{capacity}(r)$ must hold, where $\text{free-space}(r,t)$ describes the amount of free-space available, we can calculate the amount of free-space available as the following:

$$\text{free-space}(r,t) = \text{capacity}(r) - \text{level}(r,t)$$

Each resource transition T_r, independent of their type (BORROW, PRODUCE or CONSUME), reserves $\text{req}(T_r)$ amount of free-space on r during their execution. Note that reservation requests for free-space on same time points are additive. Let $\mathit{reserve}(r,t)$ represent the total amount reservation requests for free-space on the resource r at the time point t, which can calculated as the following:

$$\mathit{reserve}(r,t) = \sum_{\substack{T_r \in \mathit{schedule}(r,R) \\ s.t.\ \text{start}[T_r] \leq t < \text{end}[T_r]}} \text{req}(T_r) \tag{2.17}$$

Recall that we call a execution of transitions on a resource r *safe*, if at each time point t the level of the resource $\text{level}(r,t)$ lies within the range $[0, \text{capacity}(r)]$, and at each time point the total required reservation of free-space is satisfied. Each schedule on a resource r for a given realization R, $\mathit{schedule}(r,R)$, represents an execution of transitions on the resource r.

Definition 7. ***Safe Schedule on Resource***
A schedule $\mathit{schedule}(r,R)$ on a resource r given a realization R is called safe, if it satisfies the following two conditions:

1. *The $\mathit{schedule}(r,R)$ creates an evolution of the resource r, meaning the at each time point $t \in [0,H]$, $\text{level}(r,t) \in [0, \text{capacity}(r)]$.*
2. *At each time point $t \in [0,H]$, the total reservation of free-space must be less than or*

equal to the available amount of free-space on the resource.

$$\text{free-space}(r, t) \geq \mathit{reserve}(sv, t) \tag{2.18}$$

We will call a schedule on a resource a **valid schedule** if it is a *safe schedule* and achieves the goal conditions if specified for the resource.

Definition 8. ***Valid Schedule on Resource***
We call a schedule $schedule(r, R)$ of realization R on the resource r a ***valid schedule*** *if it satisfies the following conditions.*

1. *it is a safe schedule as defined above.*
2. *at the end of the planning horizon the amount of resource in r is within the range defined in* $\text{goal}(r)$.

$$\text{level}(r, H) \in \text{goal}(r) \tag{2.19}$$

2.4.4 Solution To a Planning Problem

Given a realization of a flexible plan, $R(\text{FlexiPlan}(\mathcal{P})$, if it creates valid schedules for each state variable and resource, then it is called a **valid realization**.

Given a flexible plan for a planning problem $\text{FlexiPlan}(\mathcal{P})$, let $\mathcal{R}^{FP}$ represents the set of all possible realizations. If each realization $R \in \mathcal{R}^{FP}$ of a flexible plan is a *valid realization*, then the flexible plan is called a **valid flexible plan**.

Definition 9. ***Solution***
A solution to a planning problem $\mathcal{P}$, is a ***valid flexible plan*** $\text{FlexiPlan}(\mathcal{P})$.

2.5 Modeling Resource Usage of Actions

Our representation language allows expressive modeling capabilities for planning and scheduling problems. Specially our representation of an action via transitions, which have different durations and can have delay from the start of the action allows us to model complex actions. In this section we discuss how we can discretize non-monotonic resource usage of actions into monotonic resource usage, and how we can model monotonic resource usage in a fine-grained discrete model.

2.5.1 Discretization of Non-Monotonic Resource Usage

A general model of resource usage can be arbitrarily complex. For example consider a move action of a car that consumes some battery power and recharges the battery during its execution.

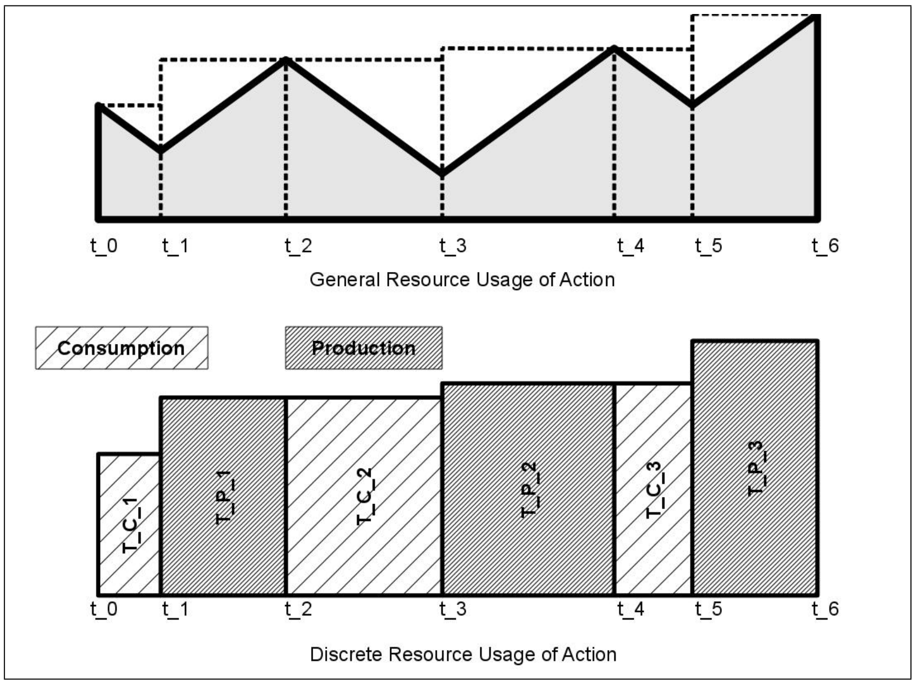


Figure 2.13: Discretization of Action Resource Usage

Figure 2.13 describes the resource usage of such an action over an interval on a resource r, where it consumes resource from t_0 to t_1, from t_2 to t_3, and from t_4 to t_5 at a fixed rate $rate_c$, and produces resource from t_1 to t_2, from t_3 to t_4, and from t_5 to t_6 at a fixed rate $rate_p$. This resource usage can be discretized as a sequence of CONSUME and PRODUCE transitions of the action on the resource r as follows:

$$\text{effect}(r, Act) = \{T_r^{C1}, T_r^{P1}, T_r^{C2}, T_r^{P2}, T_r^{C3}, T_r^{P3}\}$$

where $T_r^{C1}, T_r^{C2}, T_r^{C3}$ are the CONSUME transitions and $T_r^{P1}, T_r^{P2}, T_r^{P3}$ are the PRODUCE transitions on r. Table 2.1 describes the transitions with their type, offset values, durations, and resource requirements. In this way we can model actions that have non-monotonic resource usage in a discrete monotonic action model. Reasoning with monotonic action usage is simpler, because it can be conveniently discretized as shown above.

2.5.2 Fine-grained Discretization of Monotonic Resource Usage

We have taken a conservative approach for modeling effects of resource transitions where production and consumption events happen at the end and at the beginning of transitions respectively, and transitions reserve free-space during their execution. This model, although it makes pessimistic assumption about when a transition can start, guarantees the correctness

Name	Type	offset	duration	req
T_r^{C1}	CONSUME	0	t_1-t_0	$rate_c$*dur(T_{C1})
T_r^{P1}	PRODUCE	t_1-t_0	t_2-t_1	$rate_p$*dur(T_{P1})
T_r^{C2}	CONSUME	t_2-t_0	t_3-t_2	$rate_c$*dur(T_{C2})
T_r^{P2}	PRODUCE	t_3-t_0	t_4-t_3	$rate_c$*dur(T_{P2})
T_r^{C3}	CONSUME	t_4-t_0	t_5-t_4	$rate_c$*dur(T_{C3})
T_r^{P3}	PRODUCE	t_5-t_0	t_6-t_5	$rate_c$*dur(T_{P3})

Table 2.1: Description of Transitions of Discretized Action

of the resource usage over time. Consider the example discussed in the Transition subsection (page 23), where actions A_C and A_P are modeled with one CONSUME and one PRODUCE transition each and the duration of both these transitions are 4. Action A_C consumes resource at different consumption rates: during 0-1 it consumes 2 units, during 1-2 it consumes 1 unit, from 2-3 it consumes 4 units and during 3-4 it consumes 2 units of resource. Action A_P produces resource at fixed rate of 2 units per unit of time. We modeled these transition in a conservative coarse-grain discretized model, where if A_C starts at t then A_P could start earliest at $t + 4$, and the amount of resource available to other actions from t to $t + 8$ is 1.

To estimate better lower bounds on action start times, we can further discretize these transitions in a more fine-grained manner by expressing these transitions as a sequence of **unit duration** CONSUME and PRODUCE transitions on the resource r for actions. In this fine-grained discretized model, action A_C has the following sequence of CONSUME transitions on r:

$$\text{effect}(r, A_C) = \{Tc1, Tc2, Tc3, Tc4\}$$

and action A_P has the following sequence of PRODUCE transitions on r:

$$\text{effect}(r, A_P) = \{Tp1, Tp2, Tp3, Tp4\}$$

Table 2.2 lists the offset values and resource requirements of the transitions for action A_C and A_P. Note that all transitions have unit duration.

A_C Transition	Offset	Req	A_P Transitions	Offset	Req
$Tc1$	0	2	$Tp1$	0	2
$Tc2$	1	1	$Tp2$	1	2
$Tc3$	2	4	$Tp3$	2	2
$Tc4$	3	2	$Tp4$	3	2

Table 2.2: Transitions of Fine-grained Action

Given this fine-grained discrete model of action A_C and A_P, we can see that if A_C starts

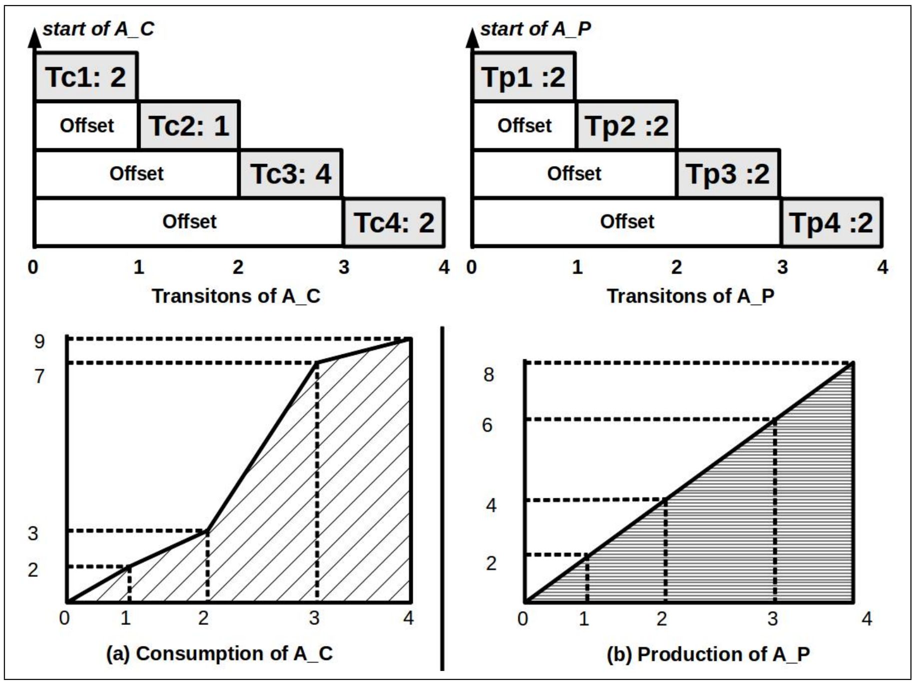


Figure 2.14: Fine-Grained Discretization

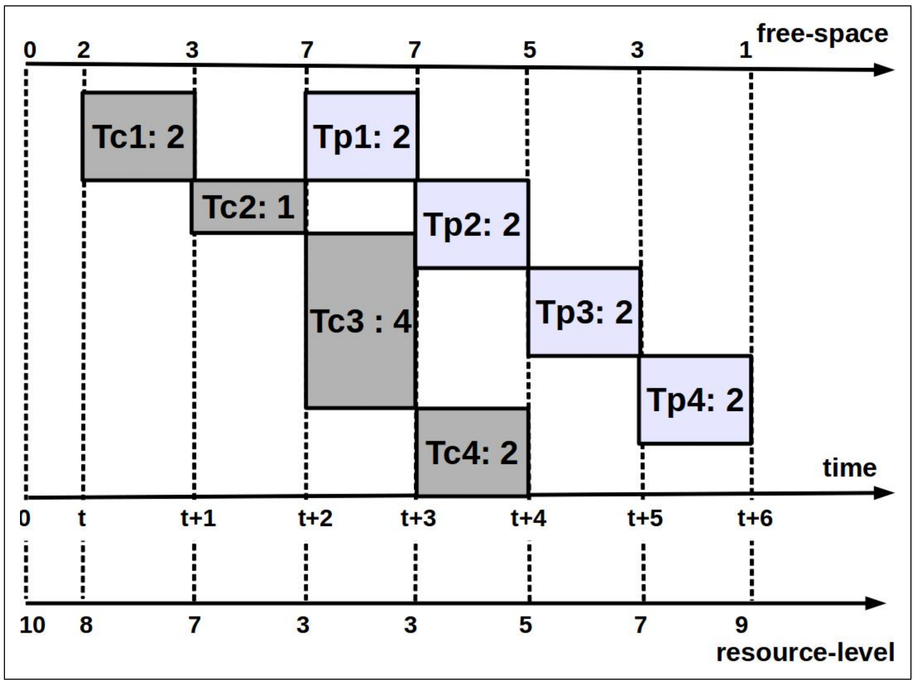


Figure 2.15: Execution of Fine-Grained Discretized Transitions

executing at time point t then A_P can start executing as early as $t+2$. This means that actions A_C and A_P can overlap. This was not possible in the coarse-grained model. Fine-grained discretization leads to a better mode of concurrency between action executions. The model of the actions and their execution on the resource r is described in Figure 2.15. Also note that the amount of resource available from t to $t+6$ is always better than in the coarse-grained model, which allows more actions to execute concurrently.

2.6 Summary

Our problem representation is based on multi-valued state variable representation of planning problems with two extensions: explicit representation of two types of resources and richer action model, where each effect of an action is typed, has variable duration and can start after a delay from start of the action. As a solution to the planning problem we create a flexible plan. Flexible plans are important in the sense that they are more resilient to uncertainties like delay of action starts exogenous events etc. A flexible plan represents a set of possible execution of the plan. The more possible executions a plan represents the more flexible, i.e resilient to the uncertainty it is.

As discussed in the beginning of this chapter, our representation is closely related to ANML [15]. Our representation is less expressive than ANML in sense that it can't represent HTN planning problems fully (but can represent certain aspect of it using action inclusion and exclusion constraints, see Section 5.4.2), and only allows actions to have certain types of effects on state variables and resources. Note that although ANML is a very expressive language, there exists no planner that can solve a problem described in ANML. In the next chapter we show that how a planning problem described in our representation, can be complied to a constraint satisfaction problem (CSP). We also show how a flexible plan can be extracted from the solution to the compiled CSP.

Chapter 3

Compilation: Transition-based Formulation

Compilation is a technique that converts a given problem in one representation to another representation, such that a solution to the problem in latter representation is also a solution to the problem in first representation. The main reason for compilations from one representation to another is that we have efficient solving methods available for problems described in the complied representation. In this chapter we will describe how to compile a problem represented in the problem description language represented in the previous chapter to a CSP by bounding the number of instances for each action. Given the relative simplicity of temporal constraint representation and efficient inference techniques developed in constraint-based scheduling, we believe CSP is best suited for solving the problems that lie inbetween planning and scheduling.

3.1 Background and Related Work

Kautz and Sellman [35] were the first to describe a reduction technique that converts a planning problem into a SAT problem. The main idea was to bound the planning problem by its makespan or number of time-steps, convert the bounded problem to a SAT problem, and solve it using a SAT solver. If no solution is found given the bound on makespan, the bound is increased and the process repeats until the first solution is found. There are other planners ([22, 57, 30]) that have been developed based on this basic idea. Similar compilation techniques are developed to convert planning problems to other constraint-based search problems. Planners like CPlan [51] and GP-CSP [19] converts a planning problem to a CSP, and ILP-Plan [34] converts to a integer linear program (ILP). The CTN [43] planner converts a planning problem described in a timeline-based representation to a CSP based on the time-step bound approach as the planners above. The main difference here is that CTN encodes the size of the bound as a decision variable as well.

An alternative bounding approach is taken in CPT [52] that compiles a planning problem into a CSP by bounding the planning problem by number of times an action can occur in the final plan. This means, first it assumes each action will occur at most once, and compiles the problem into a CSP. If the resultant CSP has no solution it increases the bound and converts the problem again. CPT's constraint model is based on the working of a partial-order planner (POP) [55] where **causal links** play a central role.

A POP planner starts with a partial plan containing two dummy actions: START and END. The START action represents the achievement of the initial state. It has no pre-condition and its add-effects are the initial conditions of the state variables. Similarly, the FINISH action represents the achievement of the goals. It has goal conditions as its pre-condition and no effects. At each planning step a POP algorithm refines the partial plan by adding actions into the plan and maintaining consistency among them, until it achieves a complete plan. The core of a POP algorithm is a concept called **causal link**. A causal link is written as $a[p]a'$. It represents that the action a supports the action a' by achieving one of its pre-condition p. Each causal link implies a precedence relation $a \rightarrow a'$, which means that a must finish executing before a' starts. The consistency is maintained by resolving **threat** to a causal link $a[p]a'$. An action b is a threat to the causal link if b makes p false as its effect and there is no precedence relation between a and b, and no precedence relation between a' and b. A POP algorithm at each step adds a causal link to provid support to pre-conditions of actions already added to the partial plan, and resolves any threats by posting additional precedence constraints. The algorithm stops when there are no pre-conditions left to be supported and there are no threats to any causal links.

Our constraint model is based on the ideas explored in CPT. We also bound a planning problem by the number of instances for each action, and compile the planning model into a CSP. Our constraint model for the state variable transitions is based on the concept of causal links, and the constraint model for the resource transitions is based on **support links** which can be thought of causal links for resources.

3.2 Overview of the Constraint Model

We want to produce a flexible plan, where start times of actions are not fixed, but constrained by distance constraints between each pair of actions. Recall that a flexible plan is called a solution to the planning problem, if and only if all its possible realizations are **valid**. A realization is valid, if it generates a **valid schedule** for each state variable and resource in the problem. This means that each solution flexible plan represents a set of *valid schedules* on each state variable and resource.

To build a solution for a given planning problem we take a bottom-up approach by posting ordering/precedence constraints between transition pairs on each resource and state variable. Each precedence constraint (or ordering) implies a temporal constraint, that the transitions can't overlap during execution. Each temporal separation due to a precedence constraint between a pair of transitions puts a distance constraint between their corresponding actions' start times. This is the case because the start times of transitions are synchronized with the start times of their actions. This approach is known as Precedence Constraint Posting (PCP)[48] in the constraint-based scheduling literature.

This chapter describes the constraint model for planning problems. The model assumes the number of occurrence of each action is bounded (as in CPT [52]). It includes constraint models for actions, state variables and resources. The main idea of the constraint model is to find out what actions should be in the plan and for state variables and resources how transitions are going to be supported. The constraint model has two layers : the first layer includes the constraint model for actions, where we decide which actions are included in the plan, and maintain the distance constraints between the start times of the included actions, and the second layer includes a constraint model for each state variable and resource. For each domain object, the constraint model contains the included transitions on the domain object and maintains the precedence constraints between them. The constraint models for resources and state variables in the second layer do not interact with each other directly, they interact via the constraint model for actions in the first layer.

Note that on each state variable, the transitions are totally ordered, except for PREVAIL transitions that require the same state, and all transitions pre-conditions must be satisfied. The constraint model for each state variable can be thought of as the constraint model for **causal links** between pairs of state variable transitions. Recall that in partial order planning (POP)[37], a *causal link* $a[p]a'$, represents the fact that action a achieves the pre-condition p for action a'. In our representation, given a state variable sv, a causal link is represented as $T_{sv}[s]T'_{sv}$, where T_{sv} is an EFFECT transition and T'_{sv} is either an EFFECT or a PREVAIL transition on sv, and s is a state of sv, i.e. $s \in \mathrm{dom}(sv)$. It represents that the EFFECT transition T_{sv} achieves the state s, where s is the pre-condition of T'_{sv}. When we say that a causal link $T_{sv}[s]T'_{sv}$ holds, it means that in the final plan T_{sv} achieves the pre-condition of T', which implies a precedence constraint between T_{sv} and T'_{sv}. For each EFFECT transition T_{sv} on sv that achieves state s, there can be more than one EFFECT transition T'_{sv} such that $T_{sv}[s]T'_{sv}$ is a possible causal link. For any given EFFECT transition T_{sv} that must execute on the state variable sv, at most one such causal link will hold in the final plan, because all EFFECT transitions must be totally ordered. To solve the constraint problem on each state variable, we decide which causal links hold in the final plan, and impose the precedence constraints between those pairs of transitions.

Given a resource and the set of activities (transitions in our case), resource reasoning in the PCP approach is generally done in two steps [21]: first find a subset of activities that overlap in time and together produce or consume more resource than the capacity of the resource (this set is also known as **peak**), and second, put enough precedence constraints between pairs of activities (note that each precedence constraint makes the pair of activities non-overlapping) in the peak such that the subset of overlapping transitions' total production or consumption lies between 0 and the capacity of the resource.

In our constraint model for resources, we have chosen a different resource reasoning approach than the two step procedure described above. Our resource reasoning is based on the idea of modelling causal links on state variables. The constraint model for each resource is based on deciding the **support links** between pairs of transitions on the resource. On a resource r, a support link, $T_r[\delta]T'_r$, represents that transition T_r provides δ amount of resource towards the requirement of T'_r. If $\delta = 0$, it means T_r does not provide any support to T'_r. If $\delta > 0$, then the support link implies a precedence relation between T_r and T'_r. By deciding how transitions provide support to other transitions, i.e. creating support links, we build a schedule on each resource.

Each *causal* and *support link* implies a precedence or ordering relation between a pair of transitions. A precedence relation between two transitions $T \rightarrow T'$ means that T' starts after T finishes its execution. Since each transition is non-preemptive, and the start times of transitions and their corresponding actions are synchronized, each precedence constraint puts a distance constraint between the start times of the actions of the transitions. The constraint model for actions maintains the consistency of these distance constraints.

As described before, our constraint formulation has two layers, where the first layer includes the constraint model for actions, and the second layer includes a constraint model for each resource and state variable. Each constraint model in the second layer interacts with the action constraint model in the first layer via two types of constraints:

- If an action is included in the plan, then all its transitions are also included in the plan/schedule of the state variables/resources that the transition affects, and vice versa.
- Start times of actions and their transitions are synchronized.

Note that in the second layer, the constraint models for state variables and resources do not interact with each other directly.

Since the constraint model of transitions plays a central role here, we call it **The Transition-based Constraint Formulation** of the planning problem. An overview of the Transition-based Constraint Formulation is shown in Figure 3.1. The action level maintains the distance constraints between the action start times, and at the domain object level, each state variable and resource constraint model maintains the precedence constraints between transition pairs on them. To solve a planning problem, we first compile the given problem into a CSP, and solve the CSP by selecting actions to include in the plan, and selecting causal and support links such a way that

- The initial and goal conditions are satisfied

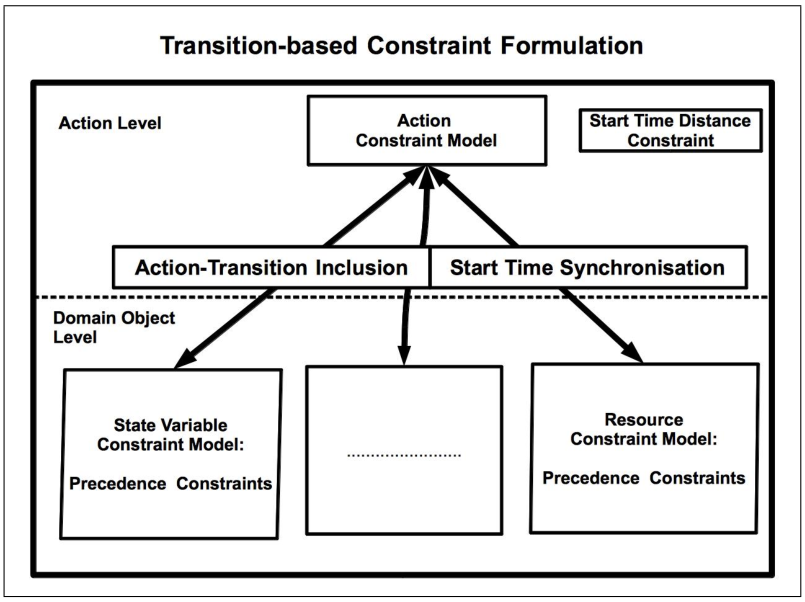

Transition-based Constraint Formulation
Action Level
Action Constraint Model
Start Time Distance Constraint
Action-Transition Inclusion
Start Time Synchronisation
Domain Object Level
State Variable Constraint Model:
Precedence Constraints
....
Resource Constraint Model:
Precedence Constraints

Figure 3.1: Overview of The Transition-based Constraint Formulation

- On each state variable, all transitions are totally ordered, except for the PREVAIL transitions on the same state, and each state variable transitions' pre-conditions are satisfied.
- On each resource all resource transitions requirements are fulfilled and the resource is not over or under consumed produced or borrowed at any time point.

A solution to the constraint model, that includes constraint models for state variables, resources and actions, represents a solution to the planning problem and the action layer represents the solution flexible plan for the given problem.

In this chapter we describe how we compile a planning problem to a transition-based constraint formulation. Before we describe the constraint model, the next section describes a pre-compilation step that adds some dummy actions and states to state variables to the original problem to make the compilation intuitive and straightforward. Then we describe the constraint variables for actions and transitions. Lastly we list the constraints relating the variables in the transition-based formulation.

3.3 Pre-Compilation of a Planning Problem

Recall that a planning problem is defined as $\mathcal{P} =< R_{reuse}, R_{reserve}, SV, A, H, init, goal >$, where R_{reuse} and $R_{reserve}$ are the sets of reusable and reservoir resources respectively; SV is the set of state variables; A is the set of actions; H is the planning horizon; $init$ is the initial configuration of the problem, and $goal$ is the goal description. Note that for each state variable and resource the initial state is defined in $init$. For each resource the goal condition is defined in $goal$. Unlike resources, goal states are defined only for a subset of state variables. We will call the subset of state variables that have goal states *goal state variables*, and the subset of state variable that have no goal states, *non-goal state variables*.

To compile a planning problem into a constraint problem, we need to ensure that each action (transition), that is included in the plan is supported. In addition to that the constraint problem should also reflect the achievement of initial conditions (described in $init$) and goal conditions ($goal$). Before we compile a planning problem into the transition-based formulation, we add new states to the domain of state variables, and new actions and transitions, such that the new planning problem (old problem representation + additional states,actions and transitions) corresponds to the original problem but make our compilation to transition-based formulation, that ensures the achievement of initial and goal conditions, easier.

3.3.1 Additional States for State Variables

To provide a uniform represenation for state variable evolution, for each state variable $sv \in SV$ we add two additional states to its domain of possible states: $start_{sv}$ and end_{sv}. Each state variable evolution would start from the $start_{sv}$ and end at end_{sv}. The only possible transition from $start_{sv}$ is to the initial state $\text{init}(sv)$, and the only possible transition to end_{sv} is from $\text{goal}(sv)$. Since for non-goal state variables $\text{goal}(sv)$ is not defined, for each non-goal state variable sv' we introduce a state, **pseudo goal**, or $PG_{sv'}$ in short, into the domain of sv', where $\text{goal}(sv') = PG_{sv'}$. Figure 3.2 describes the changed domains of two state variables: one with a goal state (top) and one without a goal state (bottom). The dotted circles represent the additional states.

3.3.2 Dummy Start and End actions

We add two dummy actions Start and End into the set of actions A, where Start and End mark the achievement of the initial state and goal respectively. The Start action is constrained to appear at the beginning of the plan, before any other action in the plan, and the End action is constrained to appear at the end of the plan, after every other action. Introducing these dummy Start and End actions is a standard practice in modeling partial order causal link (POCL) planning [37, 52], as well as in resource constrained project scheduling problems (RCPSP) [9]. Note that all transitions of all dummy actions, Start, End and other dummy actions that we introduce below, have duration 0. Dummy action Start starts at time point 0 and the End actions starts at the time point H.

On each state variable $sv \in SV$, the Start action has an EFFECT transition T_{sv}^{start} that changes the state of sv from the dummy $start_{sv}$ state to the initial state $\text{init}(sv)$, representing the achievement of the initial state of sv. That means $\text{pre}(T_{sv}^{start}) = start_{sv}$ and $\text{post}(T_{sv}^{start}) =$

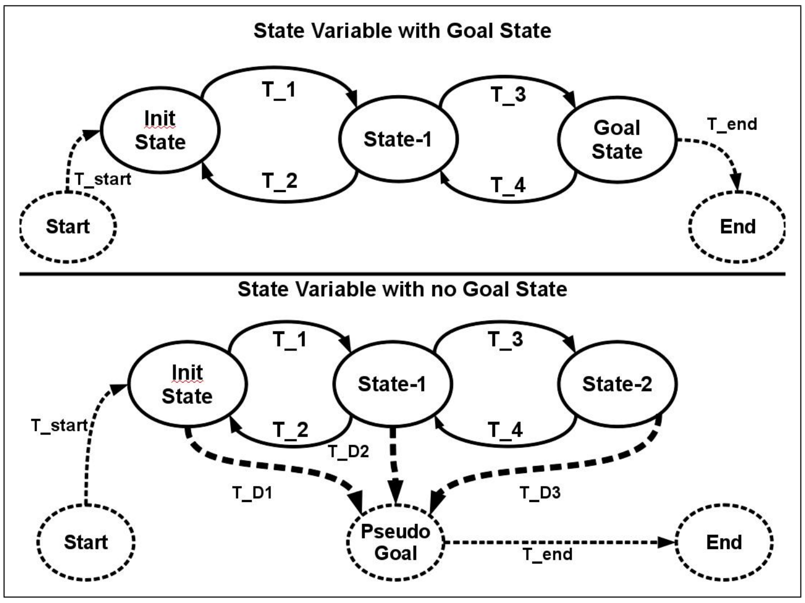


Figure 3.2: Additional States and Transitions on State Variables

$\text{init}(sv)$. The End action has an EFFECT transition T_{sv}^{end} on each state variable sv that changes its state form either the goal or the pseudo goal state to the end_{sv} state. For each goal state variable sv, $\text{pre}(T_{sv}^{end}) = \text{goal}(sv)$ and $\text{post}(T_{sv}^{end}) = end_{sv}$, and for each non-goal state variable sv', $\text{pre}(T_{sv}^{end}) = PG_{sv'}$ and $\text{post}(T_{sv}^{end}) = end_{sv'}$, $T_{sv'}^{end}$ that changes the state $PG_{sv'}$ to $end_{sv'}$. All other transitions on each state variable sv must start *after* T_{sv}^{start} and finish *before* T_{sv}^{end} as described in the Figure 3.2.

Similar to state variables, on each resource $r \in R_{reserve} \cup R_{reuse}$, the Start action has a resource transition T_r^{start}, and the End action has a resource transition T_r^{end}. All other transitions on the resource r are constrained to start their execution *after* T_r^{start} and finish their execution *before* T_r^{end}. The dummy start transition T_r^{start} on a resource r represents the availability of $\text{capacity}(r)$ amount of **space** in r at the time point 0, and the dummy end transition T_r^{end} represents that the **space** in r becomes unavailable at the time point H, i.e. at the end of the planning horizon. This means that, for each resource r, $\text{req}(T_r^{start}) = \text{req}(T_r^{end}) = \text{capacity}(r)$. The types of the dummy start and end transitions on a resource depends on the type of resource. On reusable resources, dummy start and end transitions are BORROW transitions. On reservoir resources, dummy start and end transitions have a special type, where they are considered as both PRODUCE and CONSUME transitions.

3.3.3 Dummy Actions on Reservoir Resources

For a reusable resource the initial and goal level has no meaning. This is because transitions can only borrow a reusable resource, and can not consume or produce separately. So for a reusable

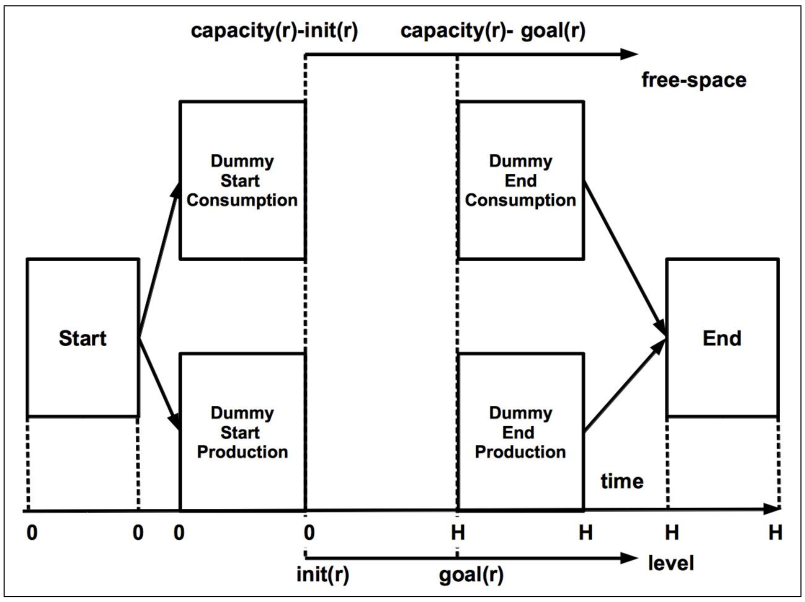


Figure 3.3: Dummy Consumption and Production transitions on reservoir resource

resource r, $\text{init}(r) = \text{capacity}(r) = \text{goal}(r)$. On the other hand, for all reservoir resource $r \in R_{reserve}$, $init(r)$ defines how much resource is available to use at the start of the plan. Note that the dummy start transition T_r^{start} on a reservoir resource r marks the availability of $\text{capacity}(r)$ amount of space, and is a special type of transition that can be both a PRODUCE and CONSUME transition. To model that on the resource r, at the beginning the amount of resource available to consume is $\text{init}(r)$ and in the resource $\text{capacity}(r) - \text{init}(r)$ amount of free-space exits at start, we create two dummy actions: a consume action $StartConsume_r$, and a production action $StartProduce_r$. Action $StartProduce_r$ has a PRODUCE transition $T_r^{StartProduce}$ on the reservoir resource r that produces $\text{init}(r)$ amount of resource. Action $StartConsume_r$ has a CONSUME transition on r $T_r^{StartConsume}$ that consumes $\text{capacity}(r) - \text{init}(r)$ amount of resource r. This means that:

$$\text{req}(T_r^{StartProduce}) = \text{init}(r) \ \& \ \text{req}(T_r^{StartConsume}) = \text{capacity}(r) - \text{init}(r)$$

On each reservoir resource these dummy actions are always included in the final plan, and their transitions are constrained to appear immediately after the dummy start transition. This means that on each reservoir resource, there will be no other transition executing in between T_r^{start} and $T_r^{StartConsume}$ and $T_r^{StartProduce}$ as described in Figure 3.3. Note that all these dummy transitions have duration 0. By introducing these dummy consumption and production transitions at the start we simulate the situation where at the start, on a reservoir resource r the resource level is $\text{init}(r)$ and the amount of free-space on r is $\text{capacity}(r) - \text{init}(r)$.

For each reservoir resource r $\mathrm{goal}(r) = [min_r, max_r]$ defines that at the end r should have atleast min_r amount of resource left, and at least $\mathrm{capacity}(r) - max_r$ amount of free-space. This means that there must be a set of PRODUCE transitions finishing their execution before the dummy end transition T_r^{end}, such that their total production is min_r, and there will be no other transition executing between this set of transitions and T_r^{end}. Similarly, there must be a set of CONSUME transitions, that consumes $\mathrm{capacity}(r) - max_r$ amount of resource, finishing immediately before T_r^{end}. To make sure that this condition holds, for each reservoir resource r we create another two dummy actions: $EndProduce_r$, and $EndConsume_r$. Action $EndConsume_r$ has a CONSUME transition on r $T_r^{EndConsume}$ that consumes min_r amount of resource, and action $EndProduce_r$ has a PRODUCE transition $T_r^{EndProduce}$ that produces $(\mathrm{capacity}(r) - max_r))$ amount of resource. This means that:

$$\mathrm{req}(T_r^{EndConsume}) = min_r \ \& \ \mathrm{req}(T_r^{EndProduce}) = \mathrm{capacity}(r) - max_r$$

Like before, these dummy actions are always included in the plan, and the transitions are constrained to execute immediately before T_r^{end} as described in the Figure 3.3. Note that these transitions always start executing at the end of the planning horizon H and have duration 0. If a CONSUME transition is included in the plan (it means if its corresponding action is included in the plan), then there must be enough resource available to consume at start of its execution. Similarly, if a PRODUCE transition must execute on a resource, there must be enough free-space available to reserve. By introducing these end dummy actions on a reservoir resource r, we make sure that each solution must produce at least min_r amount resource and at least $\mathrm{capacity}(r) - max_r$ amount of free-space before the time point H. Note that by requiring that each solution must produce $\mathrm{capacity}(r) - max_r$ amount free-space, we make sure that each solution creates at most max_r amount of resource before H, not more than that.

Note that if $\mathrm{capacity}(r) - \mathrm{init}(r) = 0$, then we don't create the dummy action $StartConsume_r$. Similarly, if $min_r = 0$, then we don't create the action $EndConsume_r$ and if $\mathrm{capacity}(r) - max_r = 0$, then we don't create the dummy action $EndProduce_r$.

3.3.4 Dummy Actions on Non-Goal State Variables

For a non-goal state variable sv, $\mathrm{goal}(sv) = PG_{sv}$, and the dummy transition T_{sv}^{end} changes PG_{sv} to end_{sv}. The non-goal state variable sv is allowed to end at any state $s \in \mathrm{dom}(sv)$, except for the states in $\text{non-final}(sv)$. To represent the transition from the end state of sv to PG_{sv}, for each non-goal state variable sv we add an action $act_{sv}^{PG\_s}$ for each of its possible states s, i.e. $s \in \mathrm{dom}(sv)$, where s is not $start_{sv}$, or end_{sv} or PG_{sv}, and $s \notin \text{non-final}(sv)$. Each action $act_{sv}^{PG\_s}$ has an EFFECT transition, $T_{sv}^{PG\_s}$, shown as the bold dotted lines (T_{D1}, T_{D2}, T_{D3}) in Figure 3.2, that changes the state s to PG_{sv}, i.e. $\mathrm{pre}(T_{sv}^{PG\_s}) = s$ and $\mathrm{post}(T_{sv}^{PG\_s}) = PG_{sv}$. These dummy actions force the end of the evolution of states for the non-goal state variables by changing current state to PG_{sv}, which then supports towards the activation of the End action, which marks the end of the plan. For each valid plan exactly one of these dummy actions must execute before the End action. In other words exactly one of these actions will be included in the final plan.

The aim of these additional states is to give a uniform structure to each state variables state-evolution where each evolution starts with the dummy start transition T_{sv}^{start} and ends with the dummy end transition T_{sv}^{end}, and on each resource the first and last transition will be T_{r}^{start} and T_{r}^{end} respectively. Note that all dummy transitions on state variables and resources, described in this section, have $\mathrm{dur}(T) = 0$.

3.4 The Constraint Model: Transition-based Formulation

Given a planning problem, its not known in advance that how many actions we need to solve the problem. To compile a planning problem to a constraint satisfaction problem, we need to bound the planning problem in some way. Because we need a finite number of variables and constraints. Constraint variables are based on action instances and their transitions. Since we don't know the number of possible occurrence of actions we need to estimate it before we can compile them to transition-based constraint formulations.

Planning problems that lie on the border of planning and scheduling often need an action at most once in solution [52]. This observation leads us to start with a CSP encoding where **each action can occur at most once**. If the encoding proved to be insoluble, then we increase the bound by one. This means that we add an extra copy of each action and search again. This process will be repeated until a solution is found. This is similar to other constraint-based planning approaches where, generally, the bound is on the makespan of the plan ([35],[19]). In other words by limiting each action occurrence to maximum one, we eliminate the difference between an action and its instance.

Given this bound, that is each action can occur at most once, we can compile the bounded planning problem into the *Transition-based Formulation*. The transition-based formulation can be seen as consisting of two layers: constraint layer for actions, and each state variable and resource having their own constant model based on the transitions on them, which are constrained via support relations. Each support relation between two transitions on state variables and resources implies an temporal distance constraint between their corresponding action start times and each action start times are synchronized with the start time of the transitions. Figure 3.1 shows an overview of the Transition-based formulation, and how different layers are related via start time constraints between transitions and actions.

In the following sections we first describe the constraint variables for actions and transitions for the transformed planning problem. Then we list the constraints on these variables capturing the interaction between transitions and actions.

3.4.1 Action Variables

For each action $a \in A \cup \{\text{DUMMY Actions}\}$, we create the following variables:

- $\texttt{start}(a)$ denoting the start time of the action a which is an integer interval.
- $\texttt{inplan}(a)$ is a boolean variable, meaning domain of $\texttt{inplan}(a)$ is $\{true, false\}$, where $\texttt{inplan}(a) = true$ means the action a is included in the final plan, and $\texttt{inplan}(a) = false$ means the action a is excluded from the plan. We will denote $\texttt{inplan}(a) = true$ as $\texttt{inplan}(a)$, and $\texttt{inplan}(a) = false$ as $\neg\,\texttt{inplan}(a)$. We will say action a is *Undecided* if $\texttt{inplan}(a)$ not assigned to be true or false.

3.4.2 Transition Variables

For each transition, either it be a state variable transition or a resource transition we create three variables similar to actions variables: $\texttt{start}(T)$, $\texttt{end}(T)$, and $\texttt{inplan}(T)$.

3.4.2.1 Variables for State Variable Transition

Given an EFFECT transition T_{sv} on a state variable, $\texttt{pre}(T_{sv})$ denotes the pre-condition, and $\texttt{post}(T_{sv})$ denotes the post-condition of T_{sv}. This means that when T_{sv} starts its execution the state of sv must be $\texttt{pre}(T_{sv})$, and the state of sv must be $\texttt{post}(T_{sv})$ when T_{sv} finishes its execution. We say the T_{sv} achieves the state $\texttt{post}(T_{sv})$ from the state $\texttt{pre}(T_{sv})$. For a PREVAIL transition T^p_{sv}, $\texttt{req}(T^p_{sv})$ denotes the state that sv must be in during the execution of T^p_{sv}. This means that for each PREVAIL transition T^p_{sv}, $\texttt{pre}(T^p_{sv}) = \texttt{post}(T^p_{sv}) = \texttt{req}(T^p_{sv})$.

Each pair of state variable transitions $< T_{sv}, T'_{sv} >$ on a state variable sv, where T_{sv} is an EFFECT transition and it achieves the pre-condition of T'_{sv}, is called an **achieve-relevant** pair. Recall that both T^{start}_{sv} and T^{end}_{sv} are EFFECT transitions. In each achieve-relevant pair $< T_{sv}, T'_{sv} >$, T_{sv} is an EFFECT transition, and T'_{sv} can be either a PREVAIL transition, or an EFFECT transition. This is because, only EFFECT transition causes a state variable state to change, thus achieve a state. PREVAIL transitions doesn't achieve any state. Each *achieve-relevant* pair of transitions represents a possible *causal link.*

The dummy start transition on sv, T^{start}_{sv}, is constrained to appear before any other transition on sv and it does not needed to be supported. This means that each pair of state variable transitions $< T_{sv}, T^{start}_{sv} >$ is **not** *achieve-relevant*. Similarly, since the end transition on sv, T^{end}_{sv} is constrained to appear after all other transitions on sv, T^{end}_{sv} does not support any transition. This means that each pair $< T^{end}_{sv}, T_{sv} >$ is **not** *achieve-relevant*. We define an *achieve-relevant* pair of state variable transitions as the following.

Definition 10. ***Achieve-Relevant Pair***
A pair of transitions $< T_{sv}, T'_{sv} >$ on a state variable sv, is called an achieve-relevant *pair if the following conditions are true:*

- *T_{sv} is an EFFECT transition, because only EFFECT transitions can cause change of state.*
- *T_{sv} achieves the pre-condition of T'_{sv}, i.e. $\text{post}(T_{sv}) = \text{pre}(T'_{sv})$.*
- *$T'_{sv} \neq T^{start}_{sv}$, because no transition can achieve the pre-condition of T^{start}_{sv}.*
- *$T_{sv} \neq T^{end}_{sv}$, because T^{end}_{sv} can't achieve any other transitions' pre-condition.*

Achieve Variables: Let $\text{AC}(sv)$ represent the set of *achieve-relevant* pairs of state variable transitions on the state variable sv. For each pair $< T_{sv}, T'_{sv} >$, such that $< T_{sv}, T'_{sv} > \in \text{AC}(sv)$ we create a constraint variable: $\text{achieve}(T_{sv}, T'_{sv})$, which can assume two possible values 0, or 1. $\text{achieve}(T_{sv}, T'_{sv}) = 0$ means that T_{sv} doesn't achieve pre-condition of T'_{sv}, and $\text{achieve}(T_{sv}, T'_{sv}) = 1$ means that T_{sv} achieves the pre-condition of T'_{sv}. Recall that each *achieve-relevant* pair represents a possible *causal link*. If $\text{achieve}(T_{sv}, T'_{sv}) = 1$, then the *causal link* $T_{sv}[\text{pre}(T'_{sv})]T'_{sv}$ holds in the final plan.

Note that on a state variable sv all EFFECT transitions must be totally ordered. Each PREVAIL transition on sv, T^p_{sv}, must execute between two EFFECT transitions, where the first EFFECT transition achieves the state $\text{req}(T^p_{sv})$, and the second EFFECT transition changes the state $\text{req}(T^p_{sv})$ to another state. This means that each PREVAIL transition must have an EFFECT transition that executes immediately after it in the final plan, as illustrated in Figure 3.4 (page 56).

Definition 11. ***Can-Follow Pair***
On a state variable sv, each pair of state variable transitions $< T^p_{sv}, T^e_{sv} >$, where T^p_{sv} is a PREVAIL transition and T^e_{sv} is an EFFECT transition, where $T^e_{sv} \neq T^{start}_{sv}$, is called a can-follow pair, if $\text{pre}(T^e_{sv}) = \text{req}(T^p_{sv})$. This means that T^e_{sv} can follow T^p_{sv} immediately after.

Follow Variables: Let $\text{FL}(sv)$ represent the set of *can-follow* pairs of transitions on the state variable sv. For each pair of transitions T^p_{sv}, and T^e_{sv}, such that $< T^p_{sv}, T^e_{sv} > \in \text{FL}(sv)$, we create a constraint variable $\text{follow}(T^p_{sv}, T^e_{sv})$ that can either be 1 meaning T_e immediately follows T_p in the final plan, or 0 otherwise. This means that, if $\text{follow}(T^p_{sv}, T^e_{sv}) = 1$, then no other EFFECT transition can execute between the end of T^p_{sv} and the start of T^E_{sv}.

3.4.2.2 Variables for Resource Transition

Similar to the *achieve-relevant* pairs of state variable transitions, a pair of resource transitions $< T_r, T'_r >$ on a resource r, is called a **support-relevant** pair if T_r can provide some amount of resource towards the fulfillment of T'_r's requirement.

On each resource $r \in R_{reuse} \cup R_{reserve}$, the dummy start transition T_r^{start} must appear before any other transition and its requirement is always satisfied. This means that each pair of transitions on r, $< T_r, T_r^{start} >$, is **not** a *support-relevant* pair. Similarly, since the dummy end transition T_r^{end} is constrained to appear after all transitions on r, thus does not provide any support to other transitions, each pair of transitions $< T_r^{end}, T_r >$ is **not** a *support-relevant* pair.

Recall that all resource transitions on a reusable resource $r \in R_{reuse}$ are of type BORROW, including the dummy start transition T_r^{start} and the dummy end transition T_r^{end}. Each Borrow transition T_r on r needs $\mathtt{req}(T_r)$ amount of resource at start and can provide $\mathtt{req}(T_r)$ amount of resource to other BORROW transitions on the same resource when it finishes its execution. That means each pair of BORROW transitions on a reusable resource are *support-relevant*.

Definition 12. ***Support-Relevant Pair on Reusable Resources***
On a reusable resource r, each pair of transitions $< T_r, T'_r >$, where $T_r \neq T_r^{end}$, $T'_r \neq T_r^{start}$, and $T_r \neq T'_r$, is a support-relevant pair.

On a reservoir resource $r \in R_{reserve}$, transitions can be either PRODUCE or CONSUME resource transitions, except for the dummy start and end transitions. The dummy start and end transitions are considered as both PRODUCE and CONSUME transitions. Each PRODUCE transition consumes $\mathtt{req}(T_r^p)$ amount of free-space at start and produces $\mathtt{req}(T_r^p)$ amount of resource at end, and each CONSUME transition consumes $\mathtt{req}(T_r^c)$ amount of resource at start and produces $\mathtt{req}(T_r^c)$ amount of free-space at end. This means that a PRODUCE transition produces resource that can be consumed by CONSUME transitions, and a CONSUME transition produces free-space that can be used by PRODUCE transitions. Form this point of view we can see that each pair of transitions on a reservoir resource, where one is a PRODUCE transition and the other one is a CONSUME transition, is a *support-relevant* pair, but any pair of transitions where both transitions have the same type is **not** *support-relevant*.

Definition 13. ***Support-Relevant Pair on Reservoir Resources***
Each pair of transitions $< T_r, T'_r >$ on a reservoir resource r, where $T_r \neq T_r^{end}$, $T'_r \neq T_r^{start}$, and $T_r \neq T'_r$, is called a support-relevant pair if the following conditions are true:

- *If T_r is a PRODUCE transition, then either T'_r is a CONSUME transition or $T'_r = T_r^{end}$*
- *If T_r is a CONSUME transition, then either T'_r is a PRODUCE transition or $T'_r = T_r^{end}$*

Support Variables: For each resource $r \in R_{reuse} \cup R_{reserve}$, let $\mathrm{SUP}(r)$ represent the set of *support-relevant* transition pairs on the resource. For each pair of resource transitions $< T_r, T'_r > \in \mathrm{SUP}(r)$, we create a variable $\mathtt{support}(T_r, T'_r)$ which represents the amount of resource (a non-negative integer) T_r provides to T'_r. For any given *support-relevant* pair of transitions $< T_r, T'_r >$, if δ_{T_r,T'_r} denotes the maximum amount of resource that T_r can provide to T'_r, then

$$\delta_{T_r,T'_r} = \min\left(\mathrm{req}(T_r), \mathrm{req}(T'_r)\right)$$

The domain of each $\mathtt{support}(T_r, T'_r)$ is the interval $[0, \delta_{T_r,T'_r}]$, where 0 indicates that T_r does not support T'_r.

3.4.3 Constraints

In the following we describe the constraints for the transition-based formulation based on the variables described above.

3.4.3.1 Non-preemptive Transitions

Transitions are non-preemptive, this means that they can't be preempted once they start execution. The following non-preemptive constraint holds for each transition T.

Constraint 1. *For each transition T, end time of T is the sum of the start time of T and duration of T.*

$$\mathrm{end}(T) = \mathrm{start}(T) + \mathrm{dur}(T) \tag{3.1}$$

3.4.3.2 Action Synchronization Constraints

The start time of each transition of an action is synchronized with the start time of the action. Recall that each transition has an offset value (non-negative) that represents the delay between the start of the action and start of the transition.

Constraint 2. *For each action $a \in A$, for all transition T, such that $T \in \mathtt{trans}(a)$, the start time of T is the sum of the start time of a and the offset value between the start of a and the start of T.*

$$\mathrm{start}(T) = \mathrm{start}(a) + \mathrm{offset}(T) \tag{3.2}$$

3.4.3.3 Horizon Constraints

Each plan must be executed within the interval $[0, H]$. This means that each action (including the dummy actions) in the plan must start after time point 0.

Constraint 3. *Each action a, including the dummy actions, must start after time point 0*

$$\forall a : 0 \leq \mathrm{start}(a) \tag{3.3}$$

Since transitions start times are delayed from the start of their corresponding actions (Constraint 2), this constraint implies that each transition must start after 0.

All actions in a plan must end before time point H. An action ends when all its transitions end.

Constraint 4. *Each transition must end before H.*

$$\forall T : \mathrm{end}(T) \leq H \tag{3.4}$$

Note that each transition (including the dummy transitions) has a non-negative duration. This constraint with Constraint 1 and Constraint 2 implies that each transition and action must start before H.

3.4.3.4 Action Activation Constraint

If an action is in the plan, i.e. $\mathrm{inplan}(a) = true$, then all its transitions are also in the plan and if an action is excluded from the plan, $\mathrm{inplan}(a) = false$, then all its transitions are also excluded from the plan.

Constraint 5. *For each action $a \in A$ and for all transition $T \in \mathrm{trans}(a)$ the following constraint holds:*

$$\mathrm{inplan}(a) \Leftrightarrow \mathrm{inplan}(T) \tag{3.5}$$

3.4.3.5 State Variable Support Constraints

Let $\mathrm{trans}(sv)^E$, and $\mathrm{trans}(sv)^P$ represent the set of EFFECT and PREVAIL transitions respectively on a state variable sv, and $\mathrm{trans}(sv) = \mathrm{trans}(sv)^E \cup \mathrm{trans}(sv)^P$. Note that the dummy start T_{sv}^{start} and the dummy end transition T_{sv}^{end} on sv are EFFECT transitions, i.e. $T_{sv}^{start}, T_{sv}^{end} \in \mathrm{trans}(sv)^E$. On each state variable sv, all EFFECT transitions that are included in the plan must be sequenced, since only one action changes states at a time. PREVAIL transitions that are included in the plan must be sequenced in between two consecutive EFFECT transitions. This means that each included EFFECT transition achieves the pre-condition of exactly one EFFECT transition and zero or more PREVAIL transitions, all included transitions' pre-conditions must be achieved by exactly one EFFECT transition, and for each included PREVAIL transition there will be exactly one EFFECT transition that will follow it. The following three constraints achieves these facts.

Recall that $\mathrm{AC}(sv)$ represents the set of *achieve-relevant* pairs on the state variable sv, and for each pair $< T_{sv}, T'_{sv} > \in \mathrm{AC}(sv)$, $T_{sv} \neq T_{sv}^{end}$, $T'_{sv} \neq T_{sv}^{start}$, and $T_{sv} \neq T'_{sv}$. Also note that in each achieve-relevant pair $< T_{sv}, T'_{sv} >$, T_{sv} is always an EFFECT transition.

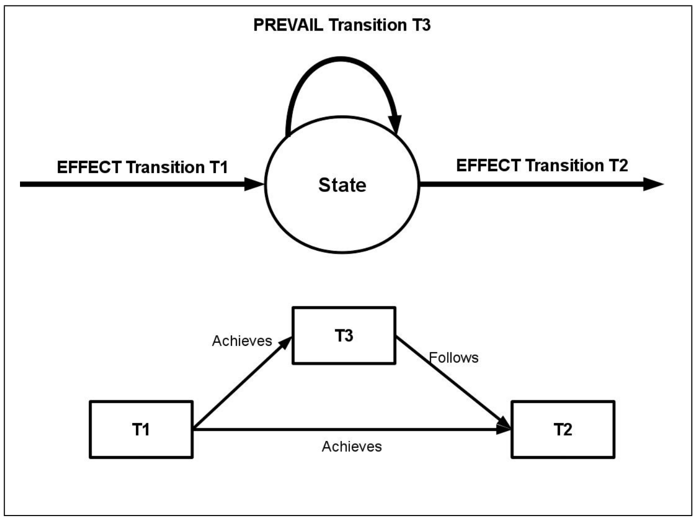


Figure 3.4: EFFECT-PREVAIL Contiguous Relation

Constraint 6. *For each transition $T_{sv} \in \text{trans}(sv)$, where $T_{sv} \neq T_{sv}^{start}$, and T_{sv} is included in the plan, there exists exactly one EFFECT transition the achieves its pre-condition.*

$$\text{inplan}(T_{sv}) \Leftrightarrow \sum_{<T'_{sv},T_{sv}>\in \text{AC}(sv)} \text{achieve}(T'_{sv}, T_{sv}) = 1 \tag{3.6}$$

Constraint 7. *For each EFFECT transition $T_{sv}^E \in \text{trans}(sv)^E$, where $T_{sv}^E \neq T_{sv}^{end}$, if T_{sv}^E is included in the plan, then it must achieve the pre-condition of exactly one other EFFECT transition.*

$$\text{inplan}(T_{sv}^E) \Leftrightarrow \sum_{<T_{sv}^E,T_{sv}^{E'}>\in \text{AC}(sv)\ s.t. T_{sv}^{E'}\in \text{trans}(sv)^E} \text{achieve}(T_{sv}^E, T_{sv}^{E'}) = 1 \tag{3.7}$$

For a state variable sv, $\text{FL}(sv)$ represents the set of *can-follow* pairs of transitions on sv. For each pair $< T_{sv}, T'_{sv} > \in \text{FL}(sv)$, T_{sv} is a PREVAIL and T'_{sv} is an EFFECT transition, where $T'_{sv} \neq T_{sv}^{start}$.

Constraint 8. *For each PREVAIL transition $T_{sv}^P \in \text{trans}(sv)^P$ on a state variable sv, if T_{sv}^P is included in the plan then there exists exactly one EFFECT transition on sv that will follow it immediately.*

$$\text{inplan}(T_{sv}^P) \Leftrightarrow \sum_{<T_{sv}^P,T_{sv}>\in \text{FL}(sv)} \text{follow}(T_{sv}^P, T_{sv}) = 1 \tag{3.8}$$

An EFFECT transition can achieve the pre-condition of exactly one EFFECT transition and zero or more PREVAIL transitions as described above. If an EFFECT transition T_{sv} achieves the pre-conditions of an EFFECT transition T'_{sv} and a PREVAIL transition T^P_{sv}, then the PREVAIL transition T^P_{sv} must execute between T_{sv} and T'_{sv}, and T'_{sv} must follow T^P_{sv} immediately. Consider the situation in Figure 3.4, where EFFECT transition T_1 achieves the pre-conditions of EFFECT transition T_2 and PREVAIL transition T_3. In this case T_2 must immediately follow T_3.

Constraint 9. *For each triplet $< T_{sv}, T^P_{sv}, T'_{sv} >$, where $T_{sv}, T'_{sv} \in \text{trans}(sv)^E$, $T^P_{sv} \in \text{trans}(sv)^P$, $< T_{sv}, T^P_{sv} >, < T_{sv}, T'_{sv} > \in \text{AC}(sv)$, and $< T^P_{sv}, T'_{sv} > \in \text{FL}(sv)$, if T_{sv} achieves the pre-conditions of T^P_{sv} and T'_{sv}, then T'_{sv} must immediately follow T^P_{sv}.*

$$\Big(\text{achieve}(T_{sv}, T'_{sv}) = 1 \wedge \text{achieve}(T_{sv}, T^P_{sv}) = 1\Big) \Rightarrow \text{follow}(T^P_{sv}, T'_{sv}) = 1 \tag{3.9}$$

When an action is decided to be excluded from the plan, $\neg\,\text{inplan}(a)$, then all its transitions are also excluded from the plan, as stated above. Each excluded EFFECT transition can't achieve the pre-condition of any other state variable transition and no other EFFECT transition achieves its pre-condition. Also, for each excluded PREVAIL transition, no EFFECT transition follows it.

Constraint 10. *For each transition $T_{sv} \in \text{trans}(sv)$, if T_{sv} is excluded from the plan, then no EFFECT transition achieves its pre-condition.*

$$\neg\,\text{inplan}(T_{sv}) \Leftrightarrow \sum_{<T'_{sv}, T_{sv}> \in \text{AC}(sv)} \text{achieve}(T'_{sv}, T_{sv}) = 0. \tag{3.10}$$

Constraint 11. *For each EFFECT transition $T^E_{sv} \in \text{trans}(sv)^E$, if T^E_{sv} is excluded from the plan, then it can't achieve the pre-condition of any transition.*

$$\neg\,\text{inplan}(T^E_{sv}) \Leftrightarrow \sum_{<T^E_{sv}, T'_{sv}> \in \text{AC}(sv)} \text{achieve}(T^E_{sv}, T'_{sv}) = 0. \tag{3.11}$$

Constraint 12. *For each PREVAIL transition $T^P_{sv} \in \text{trans}(sv)^P$, if T^P_{sv} is excluded from the plan, then no EFFECT transition can follow it.*

$$\neg\,\text{inplan}(T^P_{sv}) \Leftrightarrow \sum_{<T^P_{sv}, T'_{sv}> \in \text{FL}(sv)} \text{follow}(T^P_{sv}, T'_{sv}) = 0. \tag{3.12}$$

3.4.3.6 Resource Support Constraints

For each resource $r \in R_{reuse} \cup R_{reserve}$, if a transition T_r is included in the plan, then its requirement must be satisfied, except for the dummy start transition T^{start}_r. Also each included transition on a resource provides same amount of support to fulfill other transitions' requirements, except for the dummy end transition T^{end}_r. Note that here the requirement of a transition can be either resource (for BORROW and CONSUME transitions) or free-space (for PRODUCE transitions).

On each resource r, $\text{SUP}(r)$ represents the set of *support-relevant* transition pairs on the resource, and for each pair $< T_r, T'_r > \in \text{SUP}(r)$, $T_r \neq T_r^{end}$, $T'_r \neq T_r^{start}$, and $T_r \neq T'_r$.

Constraint 13. *For each resource transition $T_r \in \text{trans}(r)$, where $T_r \neq T_r^{start}$, if T_r is included in the plan, then its requirement must be satisfied.*

$$\text{inplan}(T_r) \Rightarrow \sum_{<T'_r, T_r> \in \text{SUP}(r)} \text{support}(T'_r, T_r) = \text{req}(T_r) \tag{3.13}$$

Constraint 14. *For each resource transition $T_r \in \text{trans}(r)$, where $T_r \neq T_r^{end}$, if T_r is included in the plan, then it should provide support of amount $\text{req}(T_r)$ to other transitions on the resource.*

$$\text{inplan}(T_r) \Rightarrow \sum_{<T'_r, T_r> \in \text{SUP}(r)} \text{support}(T_r, T'_r) = \text{req}(T_r) \tag{3.14}$$

For each transition T_r on a resource r, except for the dummy start and end transition, if T_r is included in the plan, then the above two constraints imply the following *support conservation* rule:

$$\sum_{<T'_r, T_r> \in \text{SUP}(r)} \text{support}(T'_r, T_r) = \text{req}(T_r) = \sum_{<T_r, T''_r> \in \text{SUP}(r)} \text{support}(T_r, T''_r)$$

Similar to the state variable transitions, if a resource transition T_r is excluded from the plan, then it can't support any other transitions and no transition can support its requirements.

Constraint 15. *For each resource transition T_r on a resource r, if T_r is exuded from the plan then the total support from other transitions to T_r and the total support from T_r to other transitions must be 0.*

$$\neg\,\text{inplan}(T_r) \Leftrightarrow \sum_{<T'_r, T_r> \in \text{SUP}(r)} \text{support}(T'_r, T_r) = 0 = \sum_{<T_r, T''_r> \in \text{SUP}(r)} \text{support}(T_r, T''_r) \tag{3.15}$$

3.4.3.7 Transition Ordering Constraints

For each pair of transitions T and T' we define a precedence constraint, $T \to T'$, to specify that T' is ordered after T, meaning T' starts after T finishes its execution. This ordering relation is only valid between transitions that execute on same state variable or resource, and is implied by assignments of achieve, follow, and support variables.

Constraint 16. *Assignment of each* achieve *and* follow *variable to the value 1 implies the following precedence constraint between the transitions.*

$$\forall < T_{sv}, T'_{sv} >\in \mathrm{AC}(sv) : \mathrm{achieve}(T_{sv}, T'_{sv}) = 1 \Rightarrow T_{sv} \rightarrow T'_{sv} \tag{3.16}$$
$$\forall < T^P_{sv}, T^E_{sv} >\in \mathrm{FL}(sv) : \mathrm{follow}(T^P_{sv}, T^E_{sv}) = 1 \Rightarrow T^P_{sv} \rightarrow T^E_{sv} \tag{3.17}$$

Constraint 17. *Assignment of each* support *variable to any value greater than 0 implies the following precedence constraint between the transitions.*

$$\forall < T_r, T'_r >\in \mathrm{SUP}(r) : \mathrm{support}(T_r, T'_r) > 0 \Rightarrow T_r \rightarrow T'_r \tag{3.18}$$

Constraint 18. *For each pair of transitions T and T' such that $T \rightarrow T'$ holds, the start time of T' must be greater or equal to the end time of T.*

$$T \rightarrow T' \Rightarrow \mathrm{start}(T') \geq \mathrm{end}(T) \tag{3.19}$$

As we have stated before, each precedence constraint between a pair of transitions is also a constraint on the distance between the start times of their corresponding actions. Here we will show how Constraint 18 constrains the distance between the corresponding action start times. Note that Constraint 1 specifies that the transitions are non-preemptive, meaning:

$$\mathrm{end}(T) = \mathrm{start}(T) + \mathrm{dur}(T) \tag{3.20}$$

The start time of each transition is synchronized with the start time of its action (Constraint 2), i.e.

$$\mathrm{start}(T) = \mathrm{offset}(T) + \mathrm{start}(\mathrm{act}(T)) \tag{3.21}$$

Substituting $\mathrm{start}(T)$ in equation 3.20, from equation 3.21 we get the following equation for the end time of T.

$$\mathrm{end}(T) = \mathrm{offset}(T) + \mathrm{start}(\mathrm{act}(T)) + \mathrm{dur}(T) \tag{3.22}$$

From Constraint 18, we know that $T \rightarrow T'$ means that $\mathrm{start}(T') \geq \mathrm{end}(T)$. Using equation

3.21, and equation 3.20 we can rewrite Constraint 18, as follows:

$$\begin{aligned} T \to T' &\Rightarrow \mathtt{start}(T') \geq \mathtt{end}(T) \\ &\Rightarrow \mathtt{offset}(T') + \mathtt{start}(\mathtt{act}(T')) \geq \mathtt{offset}(T) + \mathtt{start}(\mathtt{act}(T)) + \mathtt{dur}(T) \\ &\Rightarrow \mathtt{start}(\mathtt{act}(T')) - \mathtt{start}(\mathtt{act}(T)) \geq \mathtt{offset}(T) + \mathtt{dur}(T) - \mathtt{offset}(T') \end{aligned} \tag{3.23}$$

This means that each precedence constraint between a pair of transitions on same state variable or resource puts a constraint on the distance between their corresponding actions.

3.4.3.8 DUMMY Action Constraints

We have two dummy actions: Start and End, that mark the beginning and end of a plan respectively. Recall that for each resource r, $\mathtt{goal}(r) = [min_r, max_r]$. For each reservoir resource $r \in R_{reserve}$, there are four dummy actions:$StartProduce_r$, that has a PRODUCE transition which produces $\mathtt{init}(r)$ amount of resource, and $StartConsume_r$, that has a CONSUME transition which creates $\mathtt{capacity}(r) - \mathtt{init}(r)$ amount of free-space, $EndProduce_r$ that has a PRODUCE transition which produces $\mathtt{capacity}(r) - max_r$ amount of resource, and $EndConsume_r$ that has a CONSUME transition which consumes min_r amount of resource.

All dummy actions are constrained to be always in the plan.

Constraint 19. *All dummy actions must be included in the plan.*

$$\forall a \in \{DUMMYactions\} : \mathtt{inplan}(a) = true \tag{3.24}$$

Recall that the Start and End actions are constrained to appear before and after all other actions in the plan, respectively. This means that the minimum distance between Start and all other actions must be 0, and the minimum distance between all actions to End must be 0.

Constraint 20. *For all actions a and a' such that $a \neq$ Start and $a' \neq$ End:*

$$\mathtt{start}(a) - \mathtt{start}(\mathtt{Start}) \geq 0 \textit{ and } \mathtt{start}(\mathtt{End}) - \mathtt{start}(a') \geq 0 \tag{3.25}$$

For each reservoir resource the dummy start production and consumption transitions can only be supported by the dummy start transition of the resource.

Constraint 21. *For each reservoir resource $r \in R_{reserve}$, the dummy start production and consumption transitions are supported only by the dummy start transition of r.*

$$\begin{aligned} \mathtt{support}(T_r^{start}, T_r^{StartCosnume}) &= \mathtt{req}(T_r^{StartConsume}) \\ \mathtt{support}(T_r^{start}, T_r^{StartProduce}) &= \mathtt{req}(T_r^{StartProduce}) \end{aligned} \tag{3.26}$$

Note that the way we model the dummy start transitions on reservoir resources, the following is always true on each reservoir resource r.

$$\mathsf{req}(T_r^{StartConsume}) + \mathsf{req}(T_r^{StartProduce}) = \mathsf{capacity}(r) = \mathsf{req}(T_r^{start})$$

Similarly, the dummy end production and consumption transitions on a reservoir resource can only provide support to the dummy end transition of the resource.

Constraint 22. *For each reservoir resource $r \in R^g_{reserve}$, the dummy end production and consumption transitions must provide support only to the dummy end transition of r.*

$$\begin{aligned} \mathsf{support}(T_r^{EndConsume}, T_r^{end}) &= \mathsf{req}(T_r^{EndConsume}) \\ \mathsf{support}(T_r^{EndProduce}, T_r^{end}) &= \mathsf{req}(T_r^{EndProduce}) \end{aligned} \tag{3.27}$$

Due to our modeling of the dummy end transitions on reservoir resources, the following is always true for each reservoir resource r.

$$\mathsf{req}(T_r^{EndConsume}) + \mathsf{req}(T_r^{EndProduce}) \leq \mathsf{req}(T_r^{end}) = \mathsf{capacity}(r)$$

3.5 Solution To the Constraint Model

A solution to the transition-based constraint model assigns values to the inplan, $\mathsf{achieve}$, follow and $\mathsf{support}$ variables such that all constraints are satisfied. The assignments of inplan variables indicate which actions and transitions are included in the plan, and assignments of the $\mathsf{achieve}$ and follow indicates how pre-conditions of the state variable transitions are supported, and assignments of the $\mathsf{support}$ variables represent how resource requirements of the resource transitions are fulfilled. Each assignment of $\mathsf{achieve}$, follow and $\mathsf{support}$ variables implies a precedence ordering between transitions. Each precedence constraint between a pair of transitions implies a distance constraint between the corresponding actions of the transitions. Given a solution of the transition-based constraint model of the planning problem, we can extract a *flexible plan* from the solution by selecting the included actions (excluding the dummy actions) and their start times. Recall that we call a flexible plan a solution to the planning problem, if and only if all possible *realizations* of the flexible plan are *valid*. Each *valid realization* creates a *valid schedule* on each state variable and resource. In other words, each flexible plan that is a solution to the planning problem, creates a set of *valid schedules* on each domain object.

In this section we show that the flexible plan extracted from the solution to the transition-based constraint formulation of a planning problem is indeed a solution to the planning problem. We first show that on each state variable and resource, the solution to the transition-based constraint formulation creates a partially ordered set of transitions that must execute on the state variable and resource. Then we show that each possible execution of this partially ordered set of transition creates a *valid schedule* on the corresponding domain object. Lastly we show that each realization of the flexible plan extracted from the solution, creates an execution of the partially ordered set of transitions on each domain object that creates a valid schedule on the domain object. That is, we show that all realizations of the extracted flexible plan are valid, thus the flexible plan is a solution to the planning problem.

On each domain object, the solution to the transition-based constraint model creates a partially ordered set of active transitions on the domain object. We call this partially ordered set a **Partial Order Schedule (POS)** on the domain object. Note that solution of the constraint model doesn't assign the start time of transitions. We define an *execution* of a POS, as follows:

Definition 14. ***Execution of POS***
An execution of a POS assigns values to the start times of the transitions (excluding the dummy transitions) such that the following two constraints are satisfied:

1. *for each transition T,* $\text{end}(T) = \text{start}(T) + \text{dur}(T)$
2. *for each pair of transition T and T' where the precedence constraint $T \rightarrow T'$ exists in the POS:* $\text{end}(T) \leq \text{start}(T')$

Since execution of a POS refers to actual execution of transitions, we exclude the dummy transitions. We will show below that each execution of a partial order schedule represents a *valid schedule* on the corresponding domain object.

3.5.1 Partial Order Schedule On State Variables

Given a solution to the transition-based constraint formulation, for each state variable sv we create a partial order schedule, $POS(sv)$, which is a directed graph, as follows:

- For each *achieve-relevant* transition pair T_{sv} and T'_{sv}, if $\text{achieve}(T_{sv}, T'_{sv}) = 1$, then add T_{sv} and T'_{sv} as nodes in $POS(sv)$, and add a directed edge from T_{sv} to T'_{sv}.
- For each *can-follow* transition pair T_{sv} and T'_{sv}, if $\text{follow}(T_{sv}, T'_{sv}) = 1$, then add T_{sv} and T'_{sv} as nodes in $POS(sv)$, and add a directed edge from T_{sv} to T'_{sv}.

We only add a node for a transition in $POS(sv)$ above if it doesn't exist before. Note that each edge represents a precedence constraint between transitions. Given a state variable sv, the $POS(sv)$ created by the solution has the following properties:

1. For each transition T_{sv} on the state variable sv, where $\text{inplan}(T_{sv}) = true$, there exists a corresponding node for T_{sv} in $POS(sv)$. This is the case because we add an achieve-relevant pair of transitions $< T_{sv}, T'_{sv} >$ as nodes if $\text{achieve}(T_{sv}, T'_{sv}) = 1$.

This means that we need to show that if $\text{achieve}(T_{sv}, T'_{sv}) = 1$, then

$$\text{inplan}(T_{sv}) = \text{inplan}(T'_{sv}) = true$$

If $\text{achieve}(T_{sv}, T'_{sv}) = 1$, then Constraint 6 ensures that $\text{inplan}(T'_{sv}) = true$. Note that T'_{sv} can be either an EFFECT transition or a PREVAIL transition.

If T'_{sv} is an EFFECT transition, then Constraint 7 ensures that $\text{inplan}(T_{sv}) = true$.

If T'_{sv} is a PREVAIL transition, then Constraint 8 makes sure that there exists an EFFECT transition T''_{sv} such that $\text{follow}(T'_{sv}, T''_{sv}) = 1$,. Since $\text{achieve}(T_{sv}, T'_{sv}) = 1$ and $\text{follow}(T'_{sv}, T''_{sv}) = 1$, and T'_{sv} is a PREVAIL transition, Constraint 9 derives that $\text{achieve}(T_{sv}, T''_{sv}) = 1$. Since T_{sv} and T''_{sv} both are EFFECT transition, it implies that T''_{sv} is included in the plan (Constraint 6), and T_{sv} is included in the plan (Constraint 7), i.e. $\text{inplan}(T_{sv}) = true$.

In addition to this, Constraint 19, includes the dummy actions in the plan, and Constraint 5 ensures that all dummy start and end transitions are included in the plan.

2. The dummy start transition on sv, T^{start}_{sv}, has no incoming edge. This is due to the fact that each pair $< T_{sv}, T^{start}_{sv} >$ is not *achieve-relevant*. Similarly, since each pair $< T^{end}_{sv}, T_{sv} >$ is not *achieve-relevant*, the dummy end transition T^{end}_{sv} has no outgoing edge.

3. All EFFECT transitions in $POS(sv)$, except for the dummy start transition T^{start}_{sv}, have exactly one incoming edge from one other EFFECT transition in $POS(sv)$ that achieves its pre-condition. This is the case because for each EFFECT transition T_{sv} that is included in $POS(sv)$, Constraint 6 implies that there exists an EFFECT transition that achieves its pre-condition. Similarly, all EFFECT transitions, except for the dummy end transition T^{end}_{sv}, have exactly one outgoing edge to another EFFECT transition in $POS(sv)$, due to Constraint 7. Since each action and its transitions can occur at most once, Constraint 6 and Constraint 7 together imply that in $POS(sv)$ all EFFECT transitions are sequenced, and since no transition occur more than once this also implies that the sequence is acyclic.

4. Each PREVAIL transition in $POS(sv)$ appears between two consecutive EFFECT transitions in $POS(sv)$. This means that each PREVAIL transition has exactly one incoming edge from an EFFECT transition T_{sv} and exactly one out going edge to another EFFECT transition T'_{sv}, where $T_{sv} \neq T'_{sv}$, and there exists an edge from T_{sv} to T'_{sv}. This is due to the fact that each transition in $POS(sv)$ must have their pre-condition satisfied (Constraint 6) and if the EFFECT transition T_{sv} achieves the pre-condition of the PREVAIL

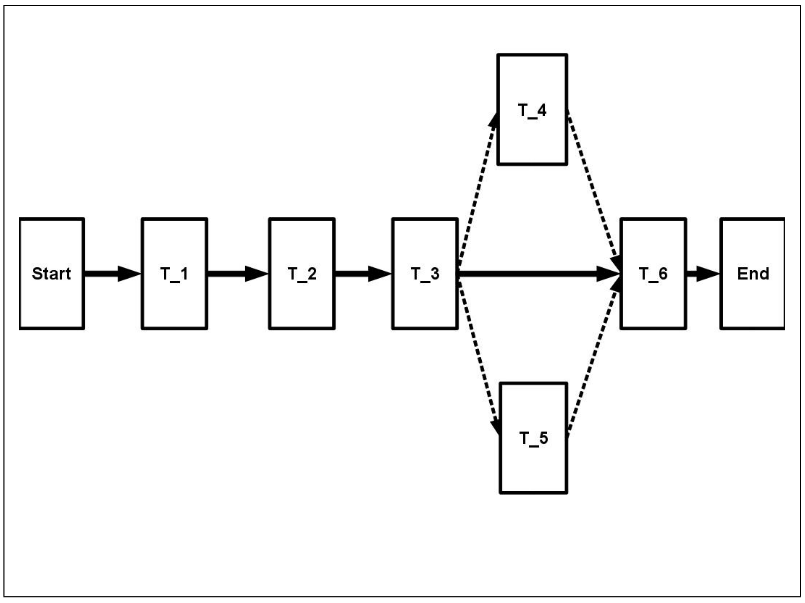


Figure 3.5: Example of POS for a state variable

transition T_{sv}^{P}, and the EFFECT transition T'_{sv} follows T_{sv}^{P}, then Constraint 9 ensures that T_{sv} must achieve the pre-condition of the EFFECT transition T'_{sv}.

For each state variable sv, $POS(sv)$ represents a set of ordering relations between the active transitions on the state variable sv, such that the EFFECT transitions are sequenced, where T_{sv}^{start} appears in the first position and T_{sv}^{end} is at the last position, and each PREVAIL transition appears in between two EFFECT transitions, i.e. is ordered with the EFFECT transitions that appear immediate before and after it. Figure 3.5 describes an example of such $POS(sv)$, where $Start$ and End represents the dummy start and end transitions on sv, T_1, T_2, T_3, and T_6 are EFFECT transitions and T_4 and T_5 are PREVAIL transitions that appear in between two EFFECT transitions T_3 and T_6.

For each state variable sv, the $POS(sv)$ created by the solution to the constraint model, each *execution* of the $POS(sv)$ achieves the conditions of a *valid schedule* on a state variable, as described in Definition 5 in the previous chapter (page 33). This is because each *execution* of $POS(sv)$

- Ensures the correct *evolution* of the state variable sv, because all EFFECT transitions are sequenced.

- Achieves the pre-conditions of all EFFECT transitions, and satisfies the overall conditions of the PREVAIL transitions. Each included PREVAIL transition appears in between two EFFECT transitions, meaning that the PREVAIL transition starts execution after the EFFECT transition that achieves its required state and finishes before the EFFECT transition that changes the state to another.

- Starts the evolution of the state variable sv from the initial state $\mathrm{init}(sv)$. The first transition in $POS(sv)$ is T_{sv}^{start} that achieves the initial state. The EFFECT transition T_{sv} that executes immediately after T_{sv}^{start} must have $\mathrm{pre}(T_{sv}) = \mathrm{post}(T_{sv}^{start}) = \mathrm{init}(sv)$. That means every execution of $POS(sv)$ starts from the state $\mathrm{init}(sv)$.

- Achieves the goal state $\mathrm{goal}(sv)$, if sv is a goal state variable. In $POS(sv)$ the last transition is T_{sv}^{end}. Since all transitions' pre-conditions are satisfied, the second to last EFFECT transition T_{sv} in the sequence, must have $\mathrm{post}(T_{sv}) = \mathrm{pre}(T_{sv}^{end}) = \mathrm{goal}(sv)$. This means that each execution of $POS(sv)$, where sv is a goal state variable, ends with the state $\mathrm{goal}(sv)$.

This means that each *execution* of $POS(sv)$ represents a valid schedule on the state variable sv. In other words each $POS(sv)$ represents a set of *valid schedules* on sv.

3.5.2 Partial Order Schedule On Resources

Given a solution, for each resource r we create a $POS(r)$ which is a directed weighted graph, as follows:

- For each *support-relevant* transition pair T_r and T'_r on the resource r, if $\mathrm{support}(T_r, T'_r) > 0$, then we add T_r and T'_r as nodes in $POS(r)$ and add a directed edge from T_r to T'_r that has the weight $\mathrm{support}(T_r, T'_r)$.

We only add a transition as a node in $POS(r)$, if it doesn't exist before. Note that each edge represents a precedence relation between the transitions. For each resource r, the $POS(r)$ has the following properties:

1. All active transitions T_r, i.e. $\mathrm{inplan}(T_r) = true$, are included as a node in $POS(r)$. This is the case because, for each transition T_r on the resource r, if $\mathrm{inplan}(T_r) = false$ then Constraint 15 ensures that for each support relevant pair, either $< T_r, T'_r >$ or $< T'_r, T_r >$, $\mathrm{support}(T_r, T'_r) = 0$ or $\mathrm{support}(T'_r, T_r) = 0$. We only add pair of transitions $< T_r, T'_r >$ where $\mathrm{support}(T_r, T'_r) > 0$. This means that all the transitions we add in $POS(r)$ are active transitions.

2. The node representing the dummy start transition T_r^{start} has no incoming edge (or support). This is due the fact that each pair $< T_r, T_r^{start} >$ is not *support-relevant*. Similarly, the node representing the dummy end transition T_r^{end} has no outgoing edge because each pair $< T_r^{end}, T_r >$ is not *support-relevant*.

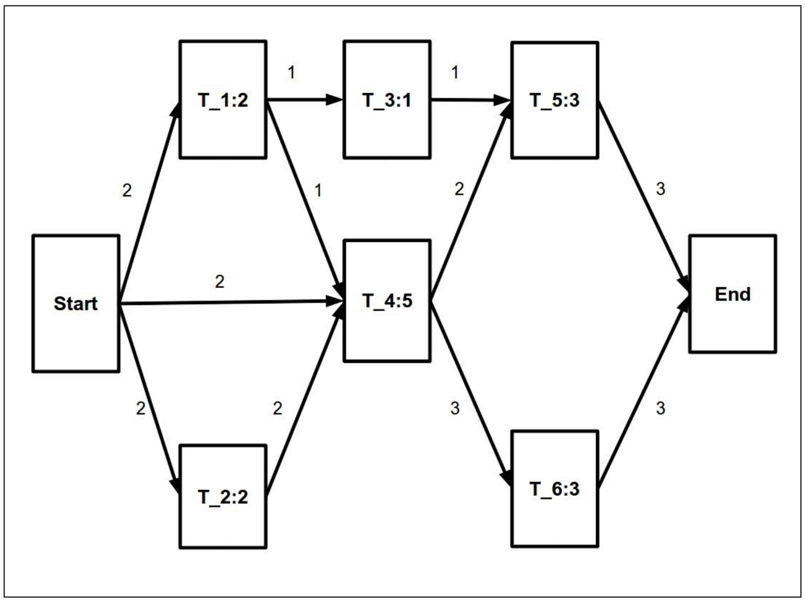


Figure 3.6: Example of $POS(r)$ for resource r

3. For each T_r included in the $POS(r)$, where $T_r \neq T_r^{start}$, the total weight of the incoming edges is equal to the $\text{req}(T_r)$. Since all transitions included in the $POS(r)$ are active transitions, Constraint 13 ensures that each active transitions requirements are satisfied. Similarly, for each transition T_r in $POS(r)$, where $T_r \neq T_r^{end}$, the total weight of the outgoing edges is equal to the $\text{req}(T_r)$, due to Constraint 14.

Given the above properties, its easy to see that each $POS(r)$ represents a *Flow Network*, where T_r^{start} represents the *source node* and T_r^{end} is the *sink node*, and all other transitions in $POS(r)$ represent the internal nodes. For each transition in the flow network, the corresponding $\text{req}(T_r)$ represents the capacity of the node, and each edge weight represents the flow from one node to other. Each internal node has the flow conservation property, i.e. total inflow is equal to total outflow. Note that each $POS(r)$ is a special sort of flow network where total in and out flow is equal to the capacity of each internal node. The flow of the network is the sum of all out-going flow from the source node, i.e. from T_{sv}^{start} in this case. Note that total out-going flow from each node is equal to $\text{req}(T_r)$ where the node represents the transition T_r, and we know that $\text{req}(T_r^{start}) = \text{capacity}(r)$. This means that each $POS(r)$ represents a flow network which has the flow equal to $\text{capacity}(r)$.

Figure 3.6 describes a $POS(r)$ for a resource r, where start and end represent the dummy start transition T_r^{start} and dummy end transition T_r^{end} on r respectively. For example, the transition T_4 in the Figure 3.6, has $\text{req}(T_4) = 5$, and there are 3 incoming edges from T_1, T_2, and T_r^{start}, with total weight of 5. Similarly, T_4 has two outgoing edges to T_5 and T_6 with total weight of 5 as well.

On each resource r the partial order schedule, $POS(r)$, that represents a flow network that has a flow equal to $\mathtt{capacity}(r)$, has the following property.

- For any given transition set $k \subset POS(r)$ of transitions, such that there are no edges between the corresponding nodes in $POS(r)$, the total requirement of the set k is always less than or equal to $\mathtt{capacity}(r)$.

For example, consider the set $\{T_1, T_2\}$, where T_1 and T_2 have no precedence constraint between them, and total requirement of them $\mathtt{req}(T_1) + \mathtt{req}(T_2)$, 2+2=4, which is less than 6 which is the capacity of r. The other independent set $\{T_5, T_6\}$, has the total requirement of 6.

For each resource r, each edge in the $POS(r)$ represents a precedence constraint between a pair of transitions. Each precedence constraint implies a temporal constraint that prevents the transition pair to overlap during execution, but if a pair of transition has no precedence constraint, then the transitions in the pair can overlap during their execution. The above property ensures that each set of transitions in the $POS(r)$, where transitions in the set can pairwise overlap, can execute in parallel without over-consuming, -producing or -borrowing the resource.

For each resource r, each *execution* of the $POS(r)$ represents a *valid schedule* as described in Definition 8 in the previous chapter (page 36). This is because each execution:

- Creates an evolution of the resource r, where at each time point $t \in [0, H]$, $\mathtt{level}(r, t) \in [0, \mathtt{capacity}(r)]$.

- Satisfies the requirement of reservation of free-space for each transition. Each CONSUME transition reserves the free-space it creates and release the free-space at the end. Each PRODUCE transition is supported by CONSUME transitions. This means that CONSUME transitions provide unreserved free-space to PRODUCE transition to reserve. If a transition's requirement is fulfilled, then it has enough free-space to reserve. Since each execution creates an evolution of r, and all transitions requirements are fulfilled, the total amount of reservations of free-space will be always less than or equal to the amount of free-space available.

- Creates an evolution of r, such that that it respects the initial condition of the resource r, i.e. at time point 0, $\mathtt{level}(r) \leq \mathtt{init}(r)$. In the case of reusable resources, since each $POS(r)$ starts with T_r^{start}, this means that at 0 $\mathtt{req}(T_r^{start}) = \mathtt{capacity}(r)$ amount of resource is available for use. Since each BORROW transition consumes at the start, and each transition have non-zero positive duration, condition $\mathtt{level}(r) \leq \mathtt{init}(r)$ holds.

  In case of reservoir resources, T_r^{start} immediately followed by dummy transitions $T_r^{StartConsume}$ and $T_r^{StartProduce}$ which simulates production of initial level of resource, and creation of initial free-space respectively. All CONSUME transitions can only starts after $T_r^{StartProduce}$

because, in each reservoir resource, T_r^{start} does not support any transitions other than the dummy start consume and produce transitions. This means that on each reservoir resource, each execution of $POS(r)$ satisfies the initial condition.

- Achieves the goal condition, i.e. at time point H, $\text{level}(r, H) \in \text{goal}(r)$. Note that goal condition on each resource r is an interval $\text{goal}(r) = [min_r, max_r]$. In the case of reusable resources, the last transition in $POS(r)$ is T_r^{end}. Since each transition's precondition is satisfied in $POS(r)$, there must be a set of BORROW transitions B, such that the following condition must hold:

$$\sum_{T \in B} \text{support}(T, T_r^{end}) = \text{req}(T_r^{end}) = \text{capacity}(r)$$

  Note that for each reusable resource r, $\text{goal}(r) = [\text{capacity}(r), \text{capacity}(r)]$, and there are no other transitions that execute in between the set B and T_r^{end}, i.e. transition in B are the last transitions to execute on r in each execution of $POS(r)$. This means that each execution of $POS(r)$, where r is a reusable resource, satisfies the goal condition.

  In the case of a reservoir resource r, the last transition in $POS(r)$, T_r^{end}, is immediately preceded by the dummy transitions $T_r^{EndProduce}$ and $T_r^{EndConsume}$. Recall that $\text{req}(T_r^{EndConsume}) = min_r$ and $\text{req}(T_r^{EndProduce}) = \text{capacity}(r) - max_r$. In $POS(r)$ there is a set of CONSUME transitions C, that provides supports to the CONSUME transition $T_r^{EndConsume}$, and a set of PRODUCE transitions P, that provides support to the PRODUCE transition $T_r^{EndProduce}$. This means that the set P produces min_r amount of resource and the set C produces $\text{capacity}(r) - max_r$ amount of free-space. Note that for each execution of $POS(r)$, transitions in C and P are among the last transitions that execute on r. There can be other PRODUCE and CONSUME transitions additional to these sets of transition executing last on r. Note that by executing last, the transitions in C, that produces $\text{capacity}(r) - max_r$ amount free-space, ensures that maximum amount of resource available in r at H is max_r. Similarly, the transition set P ensures that at H amount of resource available in r is at least min_r. This means that, each execution of $POS(r)$, where r is a reservoir resource, achieves the goal condition.

This means that, similar to the POS of state variables, each realization of $POS(r)$ represents a *valid schedule* on the resource r. In other words, for each resource r, $POS(r)$ represents a set of *valid schedules* on r.

3.5.3 Solution Extraction

As described in the previous chapter, a plan for a planning problem $\mathcal{P}$, is defined as a triplet:

$$\text{FlexiPlan}(\mathcal{P}) = < ActIns, StartTimeIntv, DistCons >$$

where $ActIns$ is a set of action instances where for each $a \in ActIns$: $\text{inplan}(a) = true$, $StartTimeIntv$ is the set of start times, i.e. $\forall a \in ActIns : \text{start}(a) \in StartTimeIntv$, and $DistCons$ is a set of start time differences where $\forall a, a' \in ActIns, a \neq a' : \text{dist}(a, a') \in DistCons$. We can extract a flexible plan from the solution as the following:

- Select each action a such that $\text{inplan}(a) = true$ and a is not a dummy action, put it in $ActIns$.

- Select each action a such that $\text{inplan}(a) = true$ and a is not a dummy action, put $[lb(\text{start}(a)), ub(\text{start}(a))]$ in $StartTimeIntv$

- Create a set of distance constraints, $\text{dist}(a, a')$, where

  $$\text{dist}(a, a') = \text{start}(a') - \text{start}(a)$$

  for each pair of actions a and a', such that $\text{inplan}(a) = true$ and $\text{inplan}(a') = true$, and put them in $DistCons$. Note that $\text{start}(a)$ and $\text{start}(a')$ are intervals and may not be fully assigned in a solution, but instead constraint on them are represented as STN.

Given a $\text{FlexiPlan}(\mathcal{P})$, extracted from the solution to the transition-based constraint model, as described above, note the following two points:

1. Due to Constraint 5, if an action a is included in the plan, then all its transitions $T \in \text{trans}(a)$ are also included in the plan. Note that each partial order schedule on domain objects includes all active transitions on that domain object. This means that for each action included in $ActIns$, each of its transitions are part of the partial order schedule on the corresponding domain object, and for all transitions (except for the dummy transitions) that are included in the partial order schedules, their corresponding actions are included in the $ActIns$.

2. Each precedence constraint between a pair of transitions in the partial order schedules implies a temporal constraint between start and end times of the transition pair, which implies a distance constraint between their corresponding action start times. When an action's start time gets updated because of precedence constraints posted on one domain object, then via Constraint 2 that change gets propagated to transitions on other domain objects.

What follows from the above two points is that the distance constraints ($DistCons$) of the $\text{FlexiPlan}(\mathcal{P})$ among the actions are the results of precedence constraints posted on the partial order schedules on domain objects.

3.5.3.1 Solution to Planning Problem

Each *realization* R of the $\mathrm{FlexiPlan}(\mathcal{P})$, where we fix the start time of each action instances (that implies the fixed start and end times for transitions) such that each distance constraint in $DistCons$ is satisfied, creates an *execution* for each partial order schedule on each domain object. For each domain object, each execution of the partial order schedule represents a *valid schedule* on the domain object. This means that all realizations of the flexible plan are *valid realizations*. That means the flexible plan extracted from the solution of the transition-based constraint model of the planning problem is a solution to the planning problem.

3.6 Summary

In this chapter we have described how a planning problem described in transition-based representation can be complied into a CSP by bounding the number of action instances. We have shown that the solution of the CSP corresponds to a flexible solution to the planning problem.

The key assumption we have made for compilation is to bound the number of action instances in a plan. In many practical planning and scheduling problems, for most of the actions we can find a small, fixed number of their occurrence. There are usually only a few action(s) that a modeler has to guess the number of occurrences of. This key parameter plays an important role in the size of complied CSP, which in turn can affect the solving time of the planning problem (see Section 6.5 for experimental results). One possible way to overcome this restriction is to add a new copy of an action when the action is included in the plan. In the rest of the thesis we assume that each action can occur at most once.

In the next chapter we describe the solving techniques for the CSP generated in this chapter.

Chapter 4

Solving: Branching, Propagation and Inference Techniques

A constraint satisfaction problem is solved by making decisions on constraint variables, where a decision is an assignment of a value to a variable from its domain. This process of making decisions, also called search, stops when there are no more decision to make. We call a constraint problem solved when there are no more decisions to make and none of the constraints are violated. The set of decisions is called a solution to the problem. There are mainly two ways to search for a solution for a constraint problem: systematic search, and local search. In systematic search, at each search step we assign a decision variable a possible value. If the assignment fails, meaning it violates some constraint, search backtracks to the previous step and try other possible values of the variable. In local search, usually we search for a solution by assigning all decision variables at once, and checking if this satisfies all constraints. If it violates any constraint, local search tries to fix the set of assignments, by reassigning a subset of variables to new values, and checking for violation of constraints again. Once it finds a set of assignment that satisfies all constraints it returns the set as the solution. One major difference between systematic search and local search is that the first one can be *complete*, and latter one is not. By complete we mean that systematic search will find a solution if there exists one, otherwise it can detect if there are no solution for the problem. Local search can't prove unsatisfiability. But, in practice, for large scale industrial problems, local search is the first choice because in general it can find a solution faster than systematic search.

The search space of a problem is defined as the all possible variable value assignments. Given a search space of a problem, only certain parts of the search space contain solutions to the problem. The time to find a solution via systematic search is proportional to the size of the search space. The bigger the search space, the more time it takes to find a solution (if one exists). One way to improve the efficiency of search[1] is to remove parts of the search space where no solution exists. Propagation of constraints means pruning the search space by removing values from the domains of constraint variables, based on the property of given

[1]From now on we will refer to systematic search by search.

constraints. Generally, given a constraint problem, propagation of constraints is performed at each search step. Because it is invoked at each step in the search space it is usually the case that we need to compromise on how much search space it can prune and how expensive it is. A propagator is called *consistent* if it doesn't prune any value from variables domains that is part of a solution. In other words, a consistent propagator doesn't prune any solution from the search space. Also, a propagation method is called *complete* if it can remove all inconsistent value from variable domains. The effect of employing complete propagation methods at each search step is that the search can find a solution without failing (that is backtrack free way). But devising complete propagation technique is non-trivial and expensive in terms of run time. Usually, in constraint-based search, cheap (low-order polynomial time) incomplete propagation techniques are employed during search.

In this chapter we describe propagation techniques for the constraint model described in the previous section. Before that we will discuss our basic branching scheme, that will describe how we make a decision at each search step.

4.1 Branching Scheme

A branching scheme describes how we make decisions at each search step. Note that we want to generate a flexible plan, meaning start times of action instances are intervals, not fixed. Recall that our constraint model has start, end and inplan variable for each action and transition, achieve variables for each *achieve-relevant* pair of state variable transitions and follow variables for each *can-follow* pair of transitions on each state variable, and support variables for each *support-relevant* pair of resource transitions on each resource. Since we don't want to make decision on the start and end times of actions and transitions, we exclude them from being decision variables. Also, due to the action activation constraints we can see that decisions on achieve and support determines the values of inplan variables (for both actions and transitions). For this reason we branch on three decision variables: achieve, follow and support.

At each CSP search step we pick a decision variable and assign a value from its domain, then propagate the effect of assignment. If the propagation fails, i.e. if the assignment leads to an inconsistent state of CSP, we backtrack and prune that value from the domain of the variable. This means that at search step, first we select a decision variable (variable selection) and then select a value from its domain (value selection). Performance of any CSP search technique depends a lot on its variable and value selection procedure. To solve different problems, it may be necessary to employ different variable value selection techniques to solve the problems efficiently. Creating a general variable value selection heuristic that would work sufficiently well on variety of problems is a hard problem and is an active research area in the automated planning and scheduling community. This topic will not be addressed in this thesis. In this chapter we discuss how we propagate constraints and infer information after each branching decision. We assume that at each search step, either a $\mathtt{achieve}$ or a $\mathtt{follow}$ or a $\mathtt{support}$ variable is given, and below we describe how we branch on the given variable.

- For each $\mathtt{achieve}(T, T')$ variable first we assign $\mathtt{achieve}(T, T') = 1$, i.e. T achieves precondition of T'. If this decision leads to a failure, then we prune the value 1 from the domain of $\mathtt{achieve}(T, T')$. Since each $\mathtt{achieve}(T, T')$ has a binary domain $\{0, 1\}$, this means that we assign $\mathtt{achieve}(T, T') = 0$, i.e. T does not achieve precondition of T'.
- For $\mathtt{follow}$ variables we do exactly same as for $\mathtt{achieve}$ variables.
- The domain of each $\mathtt{support}(T, T')$ variable is an interval $[0, ub]$, where ub is maximum possible support that T can provide to T'. For each $\mathtt{support}(T, T')$ variable first we assign $\mathtt{support}(T, T') = ub$, i.e. T provides maximum possible support to T'. If this assignment leads to a failure, then we prune ub from the domain by putting the constraint, $\mathtt{support}(T, T') < ub$. This means that $\mathtt{support}(T, T')$ will have a new upper bound, $\mathtt{support}(T, T') = \{0, ub - 1\}$.

Each decision $\mathtt{achieve}(T, T') = 1$, $\mathtt{follow}(T, T') = 1$, and $\mathtt{support}(T, T') = ub$, where $ub > 0$, have the following implications:

- Each transition in the pair $< T, T' >$, is included in the plan, $\mathtt{inplan}(T) = \mathtt{inplan}(T') = true$
- Implies a precedence constraint between the pair of transitions $< T, T' >$, $T \rightarrow T'$

These implications can trigger other constraints. In the next section we describe how constraints are propagated. After each decision, there is constraint propagation phase. If the constraint propagation is successful, then we move to next search step, if it returns failure, then we backtrack to the previous step.

4.2 Constraint Propagation

This section describes the propagation rules that implements the constraints described in the previous chapter. All these propagation rules collectively ensure the correctness of the constraint model. The main aim of these propagation rules are to maintain consistency of the constraint model by pruning inconsistent domain values. These propagation rules are executed repeatedly after each decision is made via branching until a fixed point is reached. For each propagation rule, we will use a procedural notation: $lhs \Rightarrow rhs$ to represent that if the condition on the left hand side of $\Rightarrow$ holds (*lhs*), then apply the right hand side (*rhs*). The right hand side of $\Rightarrow$ always be either an assignment of a variable-value (*set:*) or posting a constraint (*post:*).

We maintain an additional variable during the search for each pair of actions $< a, b >$, $\mathrm{dist}(a, b)$ that maintains the difference between the start times of the actions, i.e.

$$\mathrm{dist}(a, b) = \mathrm{start}(a) - \mathrm{start}(b) \tag{4.1}$$

$\mathrm{dist}(a, b)$ gets updated when either $\mathrm{start}(a)$ or $\mathrm{start}(b)$ gets updated during the propagation.

4.2.1 Failure of Propagation

The propagation returns failure, indicating an inconsistent CSP search state, only when propagation try to assign inconsistent value to inplan variables of actions. This means that if for an action a, $\mathrm{inplan}(a) = true$ and propagation tries to assign *set:* $\mathrm{inplan}(a) = false$ or vice versa, then the propagation phase returns failure.

4.2.2 Inconsistent Temporal Interval Propagation

For each constraint variable Var, let $lb(Var)$ and $ub(Var)$ represent the lowest and highest possible value in $\mathrm{dom}(Var)$ respectively. Note that an interval variable var is consistent if $lb(var) \leq ub(var)$ holds. The transition-based constraint formulation of planning problems has 4 main temporal interval variables: start of each action, $\mathrm{start}(a)$, start and end of each transition, $\mathrm{start}(T)$ and $\mathrm{end}(T)$, and the distance between each pair of actions, $\mathrm{dist}(a, b)$.

For any action a and for any transition T, if $\mathrm{start}(a)$, or $\mathrm{start}(T)$, or $\mathrm{end}(T)$ becomes inconsistent then the corresponding action or transition must be excluded from the plan.

Propagation Rule 1. *For each action and transition, inconsistent start or end time implies that the action or the transition must be excluded from the plan.*

$$lb(\mathrm{start}(a)) > ub(\mathrm{start}(a)) \Rightarrow \ set: \ \mathrm{inplan}(a) = false \tag{4.2}$$

$$lb(\mathrm{start}(T)) > ub(\mathrm{start}(T)) \Rightarrow \ set: \ \mathrm{inplan}(T) = false \tag{4.3}$$

$$lb(\mathrm{end}(T)) > ub(\mathrm{end}(T)) \Rightarrow \ set: \ \mathrm{inplan}(T) = false \tag{4.4}$$

Recall that $\mathrm{dist}(a, b)$ represents the distance from the start of action a to the start of action b. If $\mathrm{dist}(a, b)$ becomes inconsistent, then either a or b, or both a and b, must be excluded from the plan.

Propagation Rule 2. *For each action pair $< a, b > \in A$ such that $a \neq b$:*

$$lb(\mathrm{dist}(a, b)) > ub(\mathrm{dist}(a, b)) \Rightarrow \ post: \ \neg\,\mathrm{inplan}(a) \vee \neg\,\mathrm{inplan}(b) \tag{4.5}$$

For each action we say that the action is *included* in the plan if $\text{inplan}(a) = true$, and *excluded* from the plan, if $\text{inplan}(a) = false$. If $\text{dom}(\text{inplan}(a)) = \{true, false\}$, then we say that action a is undecided. When we say that action a is **not excluded**, we mean that either a is included or a is undecided. Similar terminology applies to the status of transitions.

If an action a is included in the plan, then all other actions that have a disjunctive constraint posted by Prop Rule 2 must be excluded from the plan.

Propagation Rule 3. *For each action a, if a is included in the plan, then each action b such that $\neg\,\text{inplan}(a) \vee \neg\,\text{inplan}(b)$, is excluded from the plan.*

$$\begin{aligned} \text{inplan}(a) = true \Rightarrow & \forall b : s.t. \neg\,\text{inplan}(a) \vee \neg\,\text{inplan}(b) \textit{ holds} \\ & \textit{set}: \ \text{inplan}(b) = false \end{aligned} \tag{4.6}$$

4.2.3 Propagation of Activation Constraint

The Action Activation Constraint (Constraint 5) implies that if an action is included in (or excluded from) the plan, then its transitions are included in (or excluded from) the plan, and similarly if a transition is included in (or excluded from) the plan then its corresponding action must be included in (or excluded from) the plan.

Propagation Rule 4. *For each action a, for each transition T, where $T \in \text{trans}(a)$:*

$$\text{inplan}(a) = true \Rightarrow \ \textit{set}: \ \text{inplan}(T) = true \tag{4.7}$$

$$\text{inplan}(a) = false \Rightarrow \ \textit{set}: \ \text{inplan}(T) = false \tag{4.8}$$

$$\text{inplan}(T) = true \Rightarrow \ \textit{set}: \ \text{inplan}(a) = true \tag{4.9}$$

$$\text{inplan}(T) = false \Rightarrow \ \textit{set}: \ \text{inplan}(a) = false \tag{4.10}$$

4.2.4 Propagation of Resource Support Relations

For each support-relevant pair $< T_r, T'_r >$ on a resource r, the domain of the corresponding support variable $\text{support}(T_r, T'_r))$ is an interval $[0, \delta_{T_r, T'_r}]$, where δ_{T_r, T'_r} denotes the maximum support amount that T_r can provides to T'_r. We say the $\text{support}(T_r, T'_r)$ is *undecided* if $lb(\text{support}(T_r, T'_r)) = 0$ and $ub(\text{support}(T_r, T'_r)) > 0$. This means T_r has not yet committed any support to T'_r. When

$$ub(\text{support}(T_r, T'_r)) = lb(\text{support}(T_r, T'_r)) = \delta,$$

if $\delta > 0$, then we say that $\text{support}(T_r, T'_r)$ is *assigned*. If $\delta = 0$, then we say that $\text{support}(T_r, T'_r)$ is *excluded* from the plan to mean that T_r doesn't provide any support to T'_r.

If a resource transition T_r supports some amount of resource to T'_r, i.e. if $\text{support}(T_r, T'_r)$ is assigned, then from Constraint 13 and 14 we can deduce that both T_r and T'_r are included in the plan.

Propagation Rule 5. *For each resource $r \in R_{resuse} \cup R_{reserve}$, for each support-relevant pair $< T_r, T'_r > \in \text{SUP}(r)$:*

$$\text{support}(T_r, T'_r) \;\; \textit{is assigned} \Rightarrow \;\; \textit{set}: \; \text{inplan}(T_r) = \text{inplan}(T'_r) = \textit{true} \tag{4.11}$$

When a resource transition is included in the plan, Constraint 13 ensures that its requirement is supported. We define *remaining demand* of a transition T_r, that states how much support is still needed to fulfill its total requirement considering the amount of support it already has from other supporting transitions.

Definition 15. ***Remaining Demand:*** *For a resource transition T_r, remaining demand, denoted as $\text{RemDemand}(T_r)$, is the amount of resource that T_r still needs to be supported.*

$$\text{RemDemand}(T_r) = \text{req}(T_r) - \sum_{<T'_r, T_r> \in \text{SUP}(r)} lb(\text{support}(T'_r, T_r)) \tag{4.12}$$

For a resource transition T_r, if $\text{RemDemand}(T_r) = 0$, then we say that T_r is *fully supported*. For each transition T_r, the maximum amount of support it can get from each transition on the resource that not yet committed any support to T_r, is the amount of its remaining demand.

Propagation Rule 6. *On each resource r, for each support-relevant pair $< T'_r, T_r > \in \text{SUP}(r)$, such that T'_r not yet provides any support to T_r, the maximum support T'_r can provide to T_r is* $\text{RemDemand}(T_r)$.

$$\text{support}(T'_r, T_r) \;\; \textit{is undecided} \Rightarrow \;\; \textit{set}: \; \text{support}(T'_r, T_r) \leq \text{RemDemand}(T_r) \tag{4.13}$$

Constraint 14 ensures that if a transition is included in in the plan, then it provides support to other resource transitions on the same resource. We define *remaining support* of a transition T_r that states how much resource it can still provide to other transitions, given its current supporting commitments.

Definition 16. ***Remaining Support:*** *For a resource transition T_r, remaining support, denoted as $\text{RemSupport}(T_r)$, is the amount of resource that T_r still can provide to other transitions.*

$$\text{RemSupport}(T_r) = \text{req}(T_r) - \sum_{<T_r, T'_r> \in \text{SUP}(r)} lb(\text{support}(T_r, T'_r)) \tag{4.14}$$

For a transition T_r, if $\text{RemSupport}(T) = 0$, then we say that T_r is **totally supporting**. For each transition T_r the maximum amount of support it can provide to other transitions that are not yet supported by T_r, is the $\text{RemSupport}(T)$.

Propagation Rule 7. *For each resource r, for each support-relevant pair $< T_r, T'_r > \in \text{SUP}(r)$, such that T'_r is not yet supported by T_r, the maximum support T'_r can get from T_r is* $\text{RemSupport}(T_r)$.

$$\text{support}(T_r, T'_r) \quad \textit{is undecided} \Rightarrow \ \textit{set}: \ \text{support}(T_r, T'_r) \leq \text{RemSupport}(T_r) \tag{4.15}$$

4.2.5 Propagation of State Variable Relations

Each state variable transition's pre-condition can be achieved by only one EFFECT transition. On a state variable sv,for each *achieve-relevant* pair $< T_{sv}, T'_{sv} > \in \text{AC}(sv)$, if $\text{achieve}(T_{sv}, T'_{sv}) = 1$, then it means T_{sv} achieves the pre-condition of T'_{sv} and we say that $\text{achieve}(T_{sv}, T'_{sv}) = 1$ is *decided*. If $\text{achieve}(T_{sv}, T'_{sv}) = 0$, then we say that $\text{achieve}(T_{sv}, T'_{sv})$ is *excluded* from the plan. We say $\text{achieve}(T_{sv}, T'_{sv})$ is *undecided* if it is neither decided nor excluded.

If a state variable transition achieves the pre-condition of another transition, then Constraint 6 ensures that both transitions should be included in the plan.

Propagation Rule 8. *For each state variable sv, for each achieve-relevant pair $< T_{sv}, T'_{sv} > \in$ $\text{AC}(sv)$, if T_{sv} achieves the pre-condition of T'_{sv}, then both T_{sv} and T'_{sv} are included in the plan.*

$$\text{achieve}(T_{sv}, T'_{sv}) = 1 \Rightarrow \ \textit{set}: \ \text{inplan}(T_{sv}) = \text{inplan}(T'_{sv}) = \textit{true} \tag{4.16}$$

When T_{sv} achieves the pre-condition of T'_{sv}, no other transition can achieve T'_{sv}'s pre-condition (Constraint 6).

Propagation Rule 9. *For each state variable sv, for each achieve-relevant pair $< T_{sv}, T'_{sv} > \in$ $\text{AC}(sv)$, when T_{sv} achieves the pre-condition of T'_{sv}, no other transition T''_{sv}, such that $< T''_{sv}, T'_{sv} > \in \text{AC}(sv)$, can achieve its pre-condition.*

$$\begin{aligned} \text{achieve}(T_{sv}, T'_{sv}) = 1 \quad \Rightarrow \quad & \forall T''_{sv} : T_{sv} \neq T''_{sv} \ \textit{and} \ < T''_{sv}, T'_{sv} > \in \text{AC}(sv) \\ & \textit{set}: \ \text{achieve}(T''_{sv}, T'_{sv}) = 0 \end{aligned} \tag{4.17}$$

Note that for each achieve-relevant pair $< T_{sv}, T'_{sv} >$, T_{sv} is always an EFFECT transition, and T'_{sv} is either an EFFECT or a PREVAIL transition. Each EFFECT transition can only achieve pre-condition of one EFFECT transition (Constraint 7).

Propagation Rule 10. *For each state variable sv, for each achieve-relevant pair $< T_{sv}, T^E_{sv} > \in$ $\text{AC}(sv)$, where T^E_{sv} is an EFFECT transition, if T_{sv} achieves the pre-condition of T^E_{sv}, then*

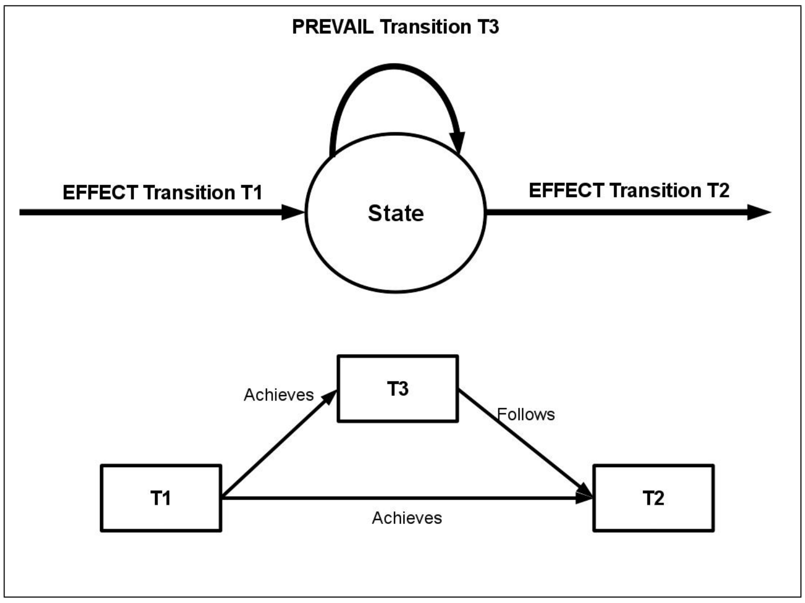


Figure 4.1: Relationship between PREVAIL and EFFECTs

T_{sv} can't achieve pre-condition of any other EFFECT transition $T_{sv}^{E'}$, where $< T_{sv}, T_{sv}^{E'} >\in \mathrm{AC}(sv)$.

$$\begin{aligned}\mathrm{achieve}(T_{sv}, T_{sv}^{E}) = 1 \quad \Rightarrow \quad & \forall T_{sv}^{E'} : T_{sv} \neq T_{sv}^{E'} \textit{ and } T_{sv}^{E'} \in \mathrm{trans}(sv)^{E} \textit{ and} \\ & < T_{sv}, T_{sv}^{E'} >\in \mathrm{AC}(sv) \\ & \textit{set}: \ \mathrm{achieve}(T_{sv}, T_{sv}^{E'}) = 0 \end{aligned} \tag{4.18}$$

For each state variable sv, for each can-follow pair $< T_{sv}^{P}, T_{sv}^{E} >\in \mathrm{FL}(sv)$, if $\mathrm{follow}(T_{sv}^{P}, T_{sv}^{E}) = 1$, then T_{sv}^{E} follows T_{sv}^{P} and both are included in the plan (Constraint 8).

Propagation Rule 11. *On each state variable sv, for each can-follow transitions pair $< T_{sv}^{P}, T_{sv}^{E} >\in \mathrm{FL}(sv)$:*

$$\mathrm{follow}(T_{sv}^{P}, T_{sv}^{E}) = 1 \Rightarrow \textit{set}: \ \mathrm{inplan}(T_{sv}^{P}) = \mathrm{inplan}(T_{sv}^{E}) = \textit{true} \tag{4.19}$$

Constraint 9 ensures that if $\mathrm{follow}(T_{sv}^{P}, T_{sv}^{E'}) = 1$, then there exists an EFFECT transition T_{sv}^{E} that achieves pre-conditions of both T_{sv}^{P} and $T_{sv}^{E'}$, as described in the Figure 4.1. On each state variable sv, for each triplet $< T_{sv}^{E}, T_{sv}^{P}, T_{sv}^{E'} >$, where T_{sv}^{E} and $T_{sv}^{E'}$ are EFFECT transitions and T_{sv}^{P} is a PREVAIL transition, such that $< T_{sv}^{E}, T_{sv}^{P} >, < T_{sv}^{E}, T_{sv}^{E'} >\in \mathrm{AC}(sv)$, and $< T_{sv}^{P}, T_{sv}^{E'} >\in \mathrm{FL}(sv)$, we propagate the following rules:

Propagation Rule 12. *If T_{sv}^{E} achieves the pre-conditions of both T_{sv}^{P} and $T_{sv}^{E'}$ then $T_{sv}^{E'}$ must* *follow* T_{sv}^{P}.

$$\text{achieve}(T_{sv}^{E}, T_{sv}^{E'}) = 1 \wedge \text{achieve}(T_{sv}^{E}, T_{sv}^{P}) = 1 \Rightarrow \textit{ set: } \text{follow}(T_{sv}^{P}, T_{sv}^{E'}) = 1 \quad (4.20)$$

Propagation Rule 13. *If* T_{sv}^{E} *achieves the pre-condition of* $T_{sv}^{E'}$ *and* $T_{sv}^{E'}$ *follows* T_{sv}^{P}*, then* T_{sv}^{E} *must achieve the pre-condition of* T_{sv}^{P}.

$$\text{achieve}(T_{sv}^{E}, T_{sv}^{E'}) = 1 \wedge \text{follow}(T_{sv}^{P}, T_{sv}^{E'}) = 1 \Rightarrow \textit{ set: } \text{achieve}(T_{sv}^{E}, T_{sv}^{P}) = 1 \quad (4.21)$$

Propagation Rule 14. *If* T_{sv}^{E} *achieves the pre-condition of* T_{sv}^{P} *and* $T_{sv}^{E'}$ *follows* T_{sv}^{P}*, then* T_{sv}^{E} *must achieve the pre-condition of* $T_{sv}^{E'}$.

$$\text{achieve}(T_{sv}^{E}, T_{sv}^{P}) = 1 \wedge \text{follow}(T_{sv}^{P}, T_{sv}^{E'}) = 1 \Rightarrow \textit{ set: } \text{achieve}(T_{sv}^{E}, T_{sv}^{E'}) = 1 \quad (4.22)$$

4.2.6 Precedence Constraint Propagation

If $\text{achieve}(T, T') = 1$ or $\text{follow}(T, T') = 1$ or $\text{support}(T, T') > 0$, then it means that both T and T' are included in the plan and there is a precedence constraint $T \rightarrow T'$. On each state variable and resource, for each transition pair $< T, T' >$, a precedence relation $T \rightarrow T'$ represents that if T and T' are included in the plan, then T' starts after T finishes its execution. This precedence relation between a pair of transitions is conditionally *transitive*. This means that if $T \rightarrow T'$ holds, and $T' \rightarrow T''$ holds, then $T \rightarrow T''$ holds, if and only if T' is included in the plan. Note that if $T \rightarrow T'$ holds, then it means that T must finish its execution before T' if and only if T and T' are included in the plan.

For each state variable and resource, for each pair of transitions $< T, T' >$, we define another relation **anti-precedence** $T \nrightarrow T'$, that represents that if T and T' are included in the plan, then T can not finish before T' starts.

Note that each precedence relation between T and T', implies an anti-precedence relation between T' and T. It means, if T must finish before T', then T' can't finish before T starts.

$$T \rightarrow T' \Rightarrow T' \nrightarrow T$$

In general, $T' \nrightarrow T$ doesn't imply that T must finish before T' starts. Note that the anti-precedence relation is not *transitive*. Consider three transitions T, T' and T'', such that all are included in the plan, and the following constraints holds:

$$T \nrightarrow T' \text{ and } T' \nrightarrow T''$$

From these relations we can't conclude that $T \nrightarrow T''$, because it might be the case that $T \rightarrow T''$, which will imply that $T'' \nrightarrow T$.

Each precedence constraint implies a temporal constraint (Constraint 18). Since effect of precedence relations is conditioned on the inclusion of the transitions in the plan, if transitions are not yet decided to be included in the plan, then precedence relations have no temporal effects. Anti-precedence relations do not have any temporal effects on transitions' start or end times. We use anti-precedence constraints for pruning domains of $\mathsf{support}$, $\mathsf{achieve}$ and follow variables.

Propagation Rule 15. *On each state variable and resource, for each pair of transitions $< T, T' >$ such that $T \rightarrow T'$ holds, if T is included in the plan, then we update the start time of T' if T' is not excluded from the plan. Similarly, if T' is included in the plan, then we update the end time of T if T is not excluded from the plan. This means that $\forall T, T' s.t.$ T and T' are not excluded, we apply the following rules*

$$\mathsf{inplan}(T) \wedge T \rightarrow T' \Rightarrow \; post: \; \mathsf{start}(T') \geq lb(\mathsf{end}(T)) \tag{4.23}$$

$$\mathsf{inplan}(T') \wedge T \rightarrow T' \Rightarrow \; post: \; \mathsf{end}(T) \leq ub(\mathsf{start}(T')) \tag{4.24}$$

Each precedence constraint between a pair of transitions implies an anti-precedence relation between the transitions.

Propagation Rule 16. *For each pair of transitions $< T, T' >$ such that neither of them is excluded from the plan, a precedence relation $T \rightarrow T'$ implies an anti-precedence relation.*

$$T \rightarrow T' \Rightarrow \; post: \; T' \nrightarrow T \tag{4.25}$$

Note that $T \nrightarrow T'$ means T can't finish its execution before T' starts its execution.

On each state variable sv, if $T_{sv} \nrightarrow T'_{sv}$ holds and T_{sv} is an EFFECT transition, then T_{sv} can't achieve the pre-condition of T'_{sv}.

Propagation Rule 17. *On each state variable sv, for each pair of achieve-relevant pair $< T_{sv}, T'_{sv} > \in \mathrm{AC}(sv)$, if $T_{sv} \nrightarrow T'_{sv}$ holds then T_{sv} can't achieve the pre-condition of T'_{sv}.*

$$T_{sv} \nrightarrow T'_{sv} \Rightarrow \; set: \; \mathsf{achieve}(T_{sv}, T'_{sv}) = 0 \tag{4.26}$$

On each state variable, if T^P_{sv} is a PREVAIL transition, and T^E_{sv} is an EFFECT transitions, and $T^P_{sv} \nrightarrow T^E_{sv}$ holds, then T^E_{sv} can't follow T^P_{sv}, if $< T^P_{sv}, T^E_{sv} >$ is a can-follow pair.

Propagation Rule 18. *On each state variable sv, for each can-follow pair $< T^P_{sv}, T^E_{sv} > \in \mathrm{FL}(sv)$, if $T^P_{sv} \nrightarrow T^E_{sv}$ holds, then T^E_{sv} can't follow T^P_{sv}.*

$$T^P_{sv} \nrightarrow T^E_{sv} \Rightarrow \; set: \; \mathsf{follow}(T^P_{sv}, T^E_{sv}) = 0 \tag{4.27}$$

Similarly, for each pair of resource transitions on a resource, T can't support T' if $T \nrightarrow T'$ holds.

Propagation Rule 19. *On each resource r, for each pair of support-relevant pair $< T_r, T'_r > \in$ $\mathrm{SUP}(r)$, if $T_r \nrightarrow T'_r$ holds then T_r can't provide support to T'_r.*

$$T_r \nrightarrow T'_r \Rightarrow \textit{ set}: \ \mathsf{support}(T_r, T'_r) = 0 \tag{4.28}$$

On each domain object to maintain the *precedence* and *anti-precedence* relation, for each transition on the domain object we maintain the following sets during the search:

- $\mathsf{before}(T)$: For each transition T' in this set, if T and T' are included in the plan, then T' must **finish before** T starts.
- $\mathsf{not\text{-}before}(T)$: For each transition T' in this set, if T and T' are included in the plan, then T' must **finish after** T starts.
- $\mathsf{after}(T)$: For each transition T' in this set, if T and T' are included in the plan, then T' must **start after** T finishes.
- $\mathsf{not\text{-}after}(T)$: For each transition T' in this set, if T and T' are included in the plan, then T' must **start before** T finishes.

When propagating, we post precedence and anti-precedence constraints between transitions. When we post $T \rightarrow T'$, then we add T' in $\mathsf{after}(T)$ and add T in $\mathsf{before}(T')$. Similarly, when we post $T \nrightarrow T'$, then we add T' in $\mathsf{not\text{-}after}(T)$ and add T in $\mathsf{not\text{-}before}(T')$. When we say the $T \rightarrow T'$ holds, we mean that $T \in \mathsf{before}(T')$, and $T' \in \mathsf{after}(T)$. Similarly, when we say $T \nrightarrow T'$ holds, we mean that $T \in \mathsf{not\text{-}before}(T')$ and $T' \in \mathsf{not\text{-}after}(T)$.

4.2.7 Non-Preemptive Temporal Constraint Propagation

Note that each transition is non-preemptive, meaning that for each transition T, $\mathsf{start}(T) + \mathsf{dur}(T) = \mathsf{end}(T)$.

Propagation Rule 20. *For each transition T, if T is not excluded from the plan, then its start and end times are updated by the following rules,*

$$\textit{(T is not excluded, and Start or End is updated)} \Rightarrow$$
$$\textit{post}: \ \mathsf{start}(T) \geq lb(\mathsf{end}(T)) - \mathsf{dur}(T) \tag{4.29}$$
$$\textit{post}: \ \mathsf{start}(T) \leq ub(\mathsf{end}(T)) - \mathsf{dur}(T) \tag{4.30}$$
$$\textit{post}: \ \mathsf{end}(T) \geq lb(\mathsf{start}(T) + \mathsf{dur}(T) \tag{4.31}$$
$$\textit{post}: \ \mathsf{end}(T) \leq ub(\mathsf{start}(T)) + \mathsf{dur}(T) \tag{4.32}$$

4.2.8 Action Start Time Distance Constraint Propagation

For each pair of actions $< a, b >$, where $a \neq b$, $\mathtt{dist}(a,b)$ represents the distance from $\mathtt{start}(a)$ to $\mathtt{start}(b)$, i.e. during search we maintain: $\mathtt{dist}(a,b) = \mathtt{start}(b) - \mathtt{start}(a)$ as described before. The value for each term can be derived from the other two terms as follows:

$$\begin{aligned}
\mathtt{dist}(a,b) &= \mathtt{start}(b) - \mathtt{start}(a) \\
\mathtt{start}(a) &= \mathtt{start}(b) - \mathtt{dist}(a,b) \\
\mathtt{start}(b) &= \mathtt{start}(a) + \mathtt{dist}(a,b)
\end{aligned}$$

Given two interval variables $[x, y]$ and $[u, v]$, from interval arithmetic we know that:

$$[x,y] + [u,v] = [x+u, y+v] \tag{4.33}$$

$$[x,y] - [u,v] = [x-v, y-u] \tag{4.34}$$

Using the above interval arithmetic formula for addition and subtraction of intervals, we can rewrite the right hand sides of the individual terms as follows:

$$\begin{aligned}
\mathtt{start}(b) - \mathtt{start}(a) &= [lb(\mathtt{start}(b)) - ub(\mathtt{start}(a)), ub(\mathtt{start}(b)) - lb(\mathtt{start}(a))] \\
\mathtt{start}(b) - \mathtt{dist}(a,b) &= [lb(\mathtt{start}(b)) - ub(\mathtt{dist}(a,b)), ub(\mathtt{start}(b)) - lb(\mathtt{dist}(a,b))] \\
\mathtt{start}(a) + \mathtt{dist}(a,b) &= [lb(\mathtt{start}(a)) + lb(\mathtt{dist}(a,b)), ub(\mathtt{start}(b)) + ub(\mathtt{dist}(a,b))]
\end{aligned}$$

For each pair of actions $< a, b >$, the following three propagation rules are applied to update the start times and the distance between them.

Propagation Rule 21. *If neither a nor b are excluded from the plan then update the distance between a and b as follows:*

$$\begin{aligned}
\textit{neither a nor b are excluded} \Rightarrow \textit{post}: \ & \mathtt{dist}(a,b) \geq X \\
\textit{post}: \ & \mathtt{dist}(a,b) \leq Y
\end{aligned} \tag{4.35}$$

where $X = lb(\mathtt{start}(b)) - ub(\mathtt{start}(a))$, *and* $Y = ub(\mathtt{start}(b)) - lb(\mathtt{start}(a))$.

Propagation Rule 22. *If a is not excluded from the plan and b is included in the plan, then the start of a is updated as follows:*

$$\begin{aligned}
\mathtt{inplan}(b) = \textit{true and a is not excluded} \Rightarrow \textit{post}: \ & \mathtt{start}(a) \geq W \\
\textit{post}: \ & \mathtt{start}(a) \leq Z
\end{aligned} \tag{4.36}$$

where $W = lb(\mathtt{start}(b)) - ub(\mathtt{dist}(a,b))$ *and* $Z = ub(\mathtt{start}(b)) - lb(\mathtt{dist}(a,b))$

Propagation Rule 23. *If a is included in the plan and b is not excluded from the plan, then we update the start time of b as follows:*

$$\begin{aligned} \text{inplan}(a) = \textit{true and b is not excluded} \Rightarrow \textit{post}:\ & \text{start}(b) \geq U \\ \textit{post}:\ & \text{start}(b) \leq V \end{aligned} \tag{4.37}$$

where $U = lb(\text{start}(a)) + lb(\text{dist}(a,b))$ *and* $V = ub(\text{start}(a)) + ub(\text{dist}(a,b))$.

In many temporal planning applications distances between action start times are represented as an Simple Temporal Network (STN), and propagated using classic STN algorithms [18]. The main difference between an STN and our proposed propagation rules are that in an STN each time point corresponds to an action that must be included in the plan, whereas our propagation rules update bounds on time points of actions that are either included in the plan or may be excluded from the plan later.

The propagation rules, described in this section, implement the constraints described in the previous chapter. In the following section we describe how we can infer more information from the propagated state of the constraint model. Mainly we infer additional psestrecedence and anti-precedence relations and use these derived relations to bound the start and end times of the actions and transitions that can be useful to make good branching choices.

4.3 Precedence Relation Inference

On a domain object, for each pair of transitions $< T, T' >$, where T and T' are included in the plan, a precedence constraint $T \rightarrow T'$ implies that T' must start after T finishes, and an anti-precedence constraint $T \nrightarrow T'$ implies that T can't finish before T' starts. In this section we describe how to infer precedence and anti-precedence relations between a pair of transitions on a domain object from the absolute values of their start and end times, from the distance between the start times of their corresponding actions, and from *mutex* (which we describe in the following) relations between the transitions. In the following when we say a pair of transitions, we mean that a pair of transitions that belongs to the same state variable or resource. Also, note that the following rules only apply to transition pairs where none of the transitions is *excluded* from the plan. This means that they are either *included* in the plan or still *undecided*.

4.3.1 via Absolute Temporal Values

For each pair of transitions, $< T, T' >$, if the earliest end time of T is greater than the latest possible start time of transition T', then we can deduce that T can't finish before T' starts, this means that we can infer $T \nrightarrow T'$.

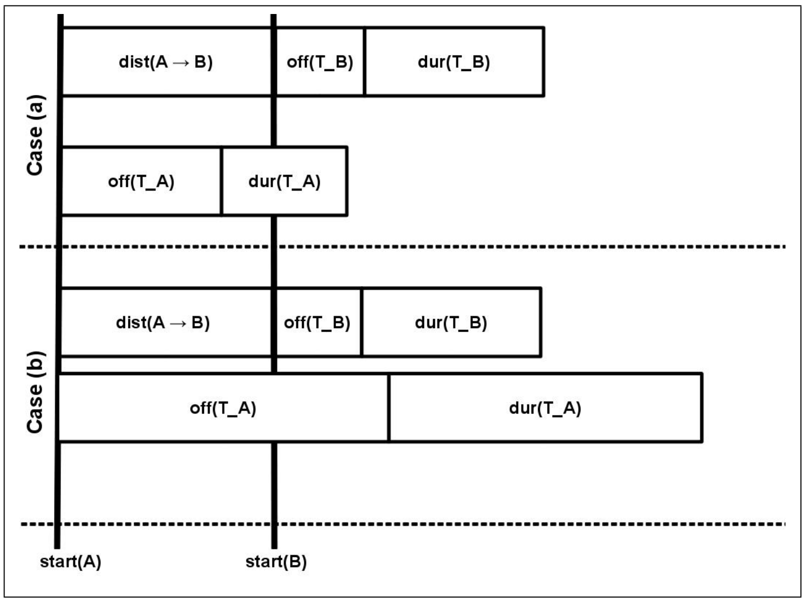


Figure 4.2: Precedence Relation Inference

Inference 1. *For each transition pair $< T, T' >$ on a state variable or a resource,*

$$lb(\text{end}(T)) > ub(\text{start}(T')) \Rightarrow \textit{post}: \; T \nrightarrow T' \tag{4.38}$$

4.3.2 via Distance Constraints

For each transition T_a, where $\text{act}(T_a) = a$, note that $\text{start}(T_a) = \text{start}(a) + \text{offset}(T_a)$ (Constraint 2), and $\text{end}(T_a) = \text{start}(T_a) + \text{dur}(T_a)$ (Constraint 1). For each pair of transitions $< T_a, T_b >$, where a and b are the corresponding actions, given the distance from a to b, $\text{dist}(a, b)$, we can infer precedence and anti-precedence relations by analyzing two cases as described in Figure 4.2.

In the first case (Case (a) in Figure 4.2), given the distance $\text{dist}(a, b)$, if the relative earliest start of T_b with respect to the start of the action a, i.e. $lb(\text{dist}(a, b)) + \text{offset}(T_b)$, is greater than or equal to the relative end of T_a, i.e. $\text{offset}(T_a) + \text{dur}(T_a)$, then we can infer that transition T_b must start after T_a ends its execution, i.e. $T_a \rightarrow T_b$ must hold.

Inference 2. *For each pair of transitions $< T_a, T_b >$, where $\text{act}(T_a) = a$ and $\text{act}(T_b) = b$, we infer the precedence relation between T_a and T_b as the following:*

$$lb(\text{dist}(a, b)) + \text{offset}(T_b) \geq \text{offset}(T_a) + \text{dur}(T_a) \Rightarrow \textit{post}: \; T_a \rightarrow T_b \tag{4.39}$$

In the second case (Case (b) in Figure 4.2), for each pair $< T_a, T_b >$, given the distance between the actions a and b, if the relative earliest end of T_b, i.e. $lb(\text{dist}(a, b)) + \text{offset}(T_b) +$

$\mathtt{dur}(T_b)$, is greater than the relative start of T_a, i.e. $\mathtt{offset}(T_a)$, then it means T_b can't finish before T_a starts, i.e. $T_b \nrightarrow T_a$ must hold.

Inference 3. *For each pair of transitions T_a and T_b on each state variable and resource, where $\mathtt{act}(T_a) = a$ and $\mathtt{act}(T_b) = b$, we infer anti-precedence relation between T_a and T_b as the following:*

$$lb(\mathtt{dist}(a,b)) + \mathtt{offset}(T_b) + \mathtt{dur}(T_b) > \mathtt{offset}(T_a) \Rightarrow \; post: \; T_b \nrightarrow T_a \tag{4.40}$$

Note that each precedence constraint implies an anti-precedence constraint, but the implication does not hold in the opposite direction. This means that for a pair of transitions if Inference 2 derives a precedence constraint, then we don't need to execute Inference 3.

Each precedence relation implies a temporal constraints between a pair of transitions on a domain object, that updates the distances between the start times of the corresponding actions of the transitions. Using this updated action start time distance, on a different domain object Inference 2 and Inference 3 derive precedence and anti-precedence relations between transitions of the same pair of actions. This means that Inference 2 and Inference 3 derive precedence and anti-precedence relations between pair of transitions on a domain object from the precedence relations posted on other domain objects.

4.3.3 via Mutex Relation on Resources

Each resource transition T_r on a resource $r \in R_{reuse} \cup R_{reserve}$, needs to reserve $\mathtt{req}(T_r)$ amount of free-space on r during the interval $[\mathtt{start}(T_r), \mathtt{end}(T_r))$. Given a pair of transitions $< T_r, T'_r >$ such that $\mathtt{req}(T_r) + \mathtt{req}(T'_r) > \mathtt{capacity}(r)$, we can deduce that T_r and T'_r can not overlap on r, because there will be not enough free-space to reserve. This means that T_r and T'_r must be ordered if both of them execute on r. Each pair of transitions on a resource, where transitions have to be ordered if they execute on the resource, is called a **mutex** pair.

Definition 17. ***Mutex Pairs***
Each pair of transitions T_r, T'_r on a resource $r \in R_{reuse} \cup R_{reserve}$, is called a mutex pair if the following condition holds:

$$\mathtt{req}(T_r) + \mathtt{req}(T'_r) > \mathtt{capacity}(r)$$

Let $\mathtt{mutex}(T_r, T'_r)$ denote a *mutex* pair on a resource r, which implies that either T_r finishes execution before T'_r or starts execution after T'_r finishes. In other words, for each pair $\mathtt{mutex}(T_r, T'_r)$, if both T_r and T'_r are included in the plan, then either $T_r \rightarrow T'_r$ or $T'_r \rightarrow T_r$ must hold.

Recall that $T_r \nrightarrow T'_r$ denotes that T_r can't finish before T'_r starts on the resource r. For each mutex pair of transitions on a resource r, we propagate the following rule:

Inference 4. *For each mutex pair of transitions* $\mathtt{mutex}(T, T')$ *on a resource* r*, if* $T_r \not\twoheadrightarrow T'_r$ *holds, then we can infer* $T'_r \rightarrow T_r$*.*

$$T_r \not\twoheadrightarrow T'_r \wedge \mathtt{mutex}(T_r, T'_r) \Rightarrow \ post: \ T'_r \rightarrow T_r \tag{4.41}$$

Inference 4 and Inference 1 together can be seen as a generalization of the *Detectable Precedence* for unary (or disjunctive) resources [54]. Since each unary resource has a capacity of 1, each pair of transitions is a mutex transition pair. For a unary resource when Inference 1 derives an anti-precedence constraint between a pair of transition, Inference 4 derives a precedence constraint between the pair of transitions. For multi-capacity resources Inference 4 only derives a precedence constraint if the pair is a mutex transition pair.

Recall that for each resource $r \in R_{reuse} \cup R_{reserv}$, the solution to the transition-based constraint model creates a partial order schedule $POS(r)$, where each $POS(r)$ is a flow network with the flow $\mathtt{capacity}(r)$. Nodes in the flow network are the transitions on the resource that are included in the plan, and edges are the *support links*. The dummy start transition T_r^{start} is the source node, and the dummy end transition T_r^{end} is the sink node of the flow network. Given any two nodes, n_1 and n_2 in the flow network, a **path** between n_1 and n_2 is a sequence of directed edges that starts from n_1 and ends at n_2. In $POS(r)$ there exists one or more *paths* from the source node to each internal node and the sink node.

During the search we build this flow network in a step-by-step manner by including a pair of transitions on the resource and creating a support-link between transitions. At each search step we refine the partial network from the previous step by adding nodes and weighted edge between nodes. Figure 4.3 shows an intermediate $POS(r)$ on a resource r, where $\mathtt{capacity}(r) = 4$, nodes $\mathtt{Start}$ and $\mathtt{End}$ denote the dummy start and end transitions, and nodes T_1 to T_5 denote the transitions on r having resource requirement 2,2,3,2, and 2 respectively, and are included in the intermediate $POS(r)$. Each edge represents a support-link between two transitions. For example, the edge between T_1 and T_4 represents that $\mathtt{support}(T_1, T_4) = 1$.
Path From Source(PFS): For a transition T_r included in an intermediate $POS(r)$ a *Path From Source*, denoted as $PFS(T_r)$, is a *path* from T_r^{start} to T_r. For transition T_4 in the intermediate $POS(r)$ described in the Figure 4.3, there is a *Path From Source*,

$$PFS(T_4) = \{\mathtt{Start} \xrightarrow{2} T_1, T_1 \xrightarrow{1} T_4\}$$

Flow of a PFS: For a transition T_r, given a $PFS(T_r)$, let *flow* of the $PFS(T_r)$ be the amount of support that the flows through the path from T_r^{start} to T_r. It is the the **minimum weight** of the edges in the path. For example, $\mathrm{flow}(PFS(T_4))$ is 1.
Flow From Source (FFS): Given an intermediate $POS(r)$, for a transition T_r let $PFS(T_r)^{all}$ be the set of all $PFS(T_r)$. The total amount of support that flows from T_r^{start} to the transition

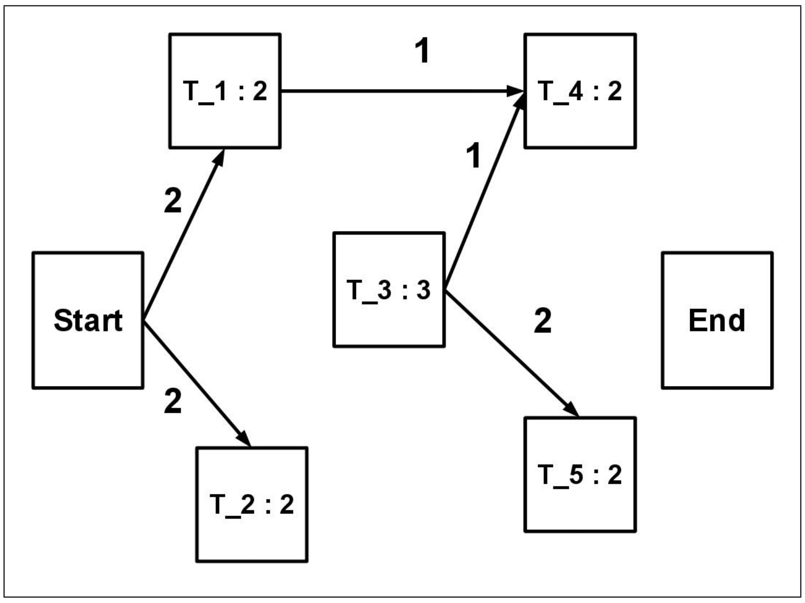


Figure 4.3: Intermediate $POS(r)$, where $\text{capacity}(r)$=4

T_r via different paths is called *Flow From Source*, denoted as

$$FFS(T_r) = \sum_{PFS(T_r) \in PFS(T_r)^{all}} \text{flow}(PFS(T_r))$$

In the implementation, at each intermediate search step, for each transition T_r, where $\text{inplan}(T_r) = true$, we calculate $FFS(T_r)$ via the following recursive formula:

$$FFS(T_r^{start}) = \text{capacity}(r) \tag{4.42}$$

$$FFS(T_r) = \sum_{<T_r', T_r> \in \text{SUP}(r)} \min\left(lb(\text{support}(T_r', T_r)), FFS(T_r')\right) \tag{4.43}$$

$FFS(T_r)$ denotes the total amount of support that flows directly from the dummy start transition T_r^{start} to T_r. We can interpret this as the amount of free-space that can be reserved only by a set of transitions P before T_r starts, where each transition in P is involved in a support-link that appears in a path in $PFS(T_r)^{all}$. For all other transitions that are not in P can only reserve $\text{capacity}(r) - FFS(T_r)$ amount of free-space before T_r starts. The maximum value of $FFS(T_r)$ is $\text{req}(T_r)$. This means that each transition in P can reserve atleast $\text{capacity}(r) - \text{req}(T_r)$ amount of free-space. Let T_r' be a transition in P. If the pair $< T_r, T_r' >$ is not a mutex pair, then $\text{req}(T_r) + \text{req}(T_r') \leq \text{capacity}(r)$. This means that T_r' always have enough free-space to reserve if it starts before T_r. But, if $< T_r, T_r' >$ is a mutex pair, i.e. $\text{req}(T_r) + \text{req}(T_r') \geq \text{capacity}(r)$, and if $\text{capacity}(r) - FFS(T_r) < \text{req}(T_r')$, then it would mean that

T'_r can't start before T_r. Since each mutex pair must be ordered, we can deduce that T'_r must start after T_r finishes.

Inference 5. *For each transition T_r on a resource r, such that* $\text{inplan}(T_r) = true$*, for each T'_r, where $< T_r, T'_r >$ is a mutex pair, and neither $T_r \rightarrow T'_r$ nor $T'_r \rightarrow T_r$ holds, we propagate the following rule:*

$$(\text{capacity}(r) - FFS(T_r)) < \text{req}(T'_r) \Rightarrow \textit{post: } T_r \rightarrow T'_r \tag{4.44}$$

For example, consider the pair of transitions $< T1, T3 >$ in the intermediate $POS(r)$ described in the Figure 4.3. $< T1, T3 >$ is a mutex pair because sum of their requirement is 5 which greater than the capacity of the resource (which is 4). We can see in the figure that $FFS(T1) = 2$, which means that $T3$ can only reserve $4 - 2 = 2$ amount of free-space if it starts before $T1$ finishes. Since requirement of $T3$ is 3, from Inference 5 we can deduce that $T3$ must starts after $T1$ finishes, i.e. $T1 \rightarrow T3$ must hold. Similarly, we can deduce using Inference 5 that $T2 \rightarrow T3$ must hold.

4.4 Temporal Inference: Lower bounding Start Times

Start times of transitions and actions are propagated at search steps by the propagation rules as described above. In a consistent search state, after propagation each transition and action have an admissible lower bound on the start times. In this section we describe how to infer better lower bounds on the start times of transitions. Since the start times of transitions and their actions are synchronized, better lower bounds on the start times of transitions imply better lower bounds on the start times of actions.

In the following subsections we describe two different inference techniques that provide admissible lower bounds on the start times of transitions. The first technique, given a transition T, calculates the earliest possible time when T's pre-condition (or demand) is going to be satisfied, and the second technique derives the lower bound by analyzing what must happen before T can start.

4.4.1 Lower Bound from Possible Supporters and Achievers

On each resource r, a transition can start its execution if its demand is satisfied by other transitions on r. Similarly, on a state variable sv a transition T_{sv} can start its execution when its pre-condition is achieved. First we describe how we estimate an admissible lower bound on the start times of resource transitions, and then for state variable transitions.

4.4.1.1 For Resource Transitions

For each resource transition a lower bound on its start time can be estimated by calculating the earliest possible time when its requirement can be fulfilled.

Possible Supporters: For a resource transition T_r, let $\text{PossSupp}(T_r)$ denote the set of possible supporters of T_r. This means that for each $T'_r \in \text{PossSupp}(T_r)$, $\text{support}(T'_r, T_r)$ is *undecided*.

Valid Supporters: Furthermore, let $\text{ValidSupp}(T_r)$ denote a subset of $\text{PossSupp}(T_r)$, such that the total possible (maximum) support from the supporters in $\text{ValidSupp}(T_r)$ is greater or equal to the remaining demand of T_r.

This means that for each subset $\text{ValidSupp}(T_r) \subseteq \text{PossSupp}(T_r)$ the following condition holds

$$\sum_{T'_r \in \text{ValidSupp}(T_r)} ub\left(\text{support}(T'_r, T_r)\right) \geq \text{RemDemand}(T_r) \tag{4.45}$$

For each $\text{ValidSupp}(T_r)$, the earliest possible time when it generates the collective support for T_r, is when the last transition in $\text{ValidSupp}(T_r)$ finishes its execution, i.e.

$$\text{eft}\left(\text{ValidSupp}(T_r)\right) = \max_{T'_r \in \text{ValidSupp}(T_r)} lb(\text{end}(T'_r))$$

Where $\text{eft}(\text{ValidSupp}(T_r))$ stands for *earliest finish time* of $\text{ValidSupp}(T_r)$. Let $VS(T_r)$ be the set of all possible valid supporter sets for T_r. The earliest possible time when T_r can be fully supported depends on the earliest finish time of the valid supporter set that has the minimum earliest finish time among all possible valid supporter sets. This means that if $\text{ValidSupp}(T_r)^{min}$ denotes such a valid supporter set, then

$$\text{ValidSupp}(T_r)^{min} = \operatorname*{arg\,min}_{\text{ValidSupp}(T_r) \in VS(T_r)} \text{eft}\left(\text{ValidSupp}(T_r)\right) \tag{4.46}$$

For a transition T_r, $\text{eft}(\text{ValidSupp}(T_r)^{min})$ is a valid lower bound on the start time of T_r, i.e.

$$\text{start}(T_r) \geq \text{eft}(\text{ValidSupp}(T_r)^{min})$$

For a resource transition T_r given the $\text{PossSupp}(T_r)$, Algorithm 1 calculates the earliest finish time of the minimum valid support set of T_r, i.e. $\text{eft}(\text{ValidSupp}(T_r)^{min})$.

Working of Algorithm 1: It first sorts the transitions in the $\text{PossSupp}(T_r)$ in the non-decreasing order of their earliest end times. Then it accumulates maximum supports sequentially from the sorted set starting from the first element. It stops when the accumulated support is greater than or equal to the remaining demand of T_r. It returns the earliest finish time of the last element it scanned as the lower bound of the start time of T_r.

Algorithm 1 $get\_\mathrm{eft}\,(\mathrm{PossSupp}(T_r))$

```
1: sort $\mathrm{PossSupp}(T_r)$ in the ascending order of earliest finish time.
2: Initialize $supp = 0$
3: for each $T_i \in \mathrm{PossSupp}(T_r)$ where $i = 1$ to $|\mathrm{PossSupp}(T_r)|$ do
4:     $supp$ += $ub\,(\mathrm{support}(T_i, T_r))$
5:     if $supp \geq \mathrm{RemDemand}(T_r)$ then
6:         return $\mathrm{eft}(T_i)$
7:     end if
8: end for
9: return $\mathrm{inf}$
```

To update the lower bound on the start time of transitions we propagate the following rule

Inference 6. *For each resource transition T_r, such that* $\mathrm{inplan}(T_r) \neq false$*,*

$$\mathrm{start}(T_r) \geq get\_\mathrm{eft}\,(\mathrm{PossSupp}(T_r)) \tag{4.47}$$

Note that if there does not exist enough support for T_r, then Algorithm 1 returns inf (infinity). It means T_r should be excluded from the plan due to lack of support.

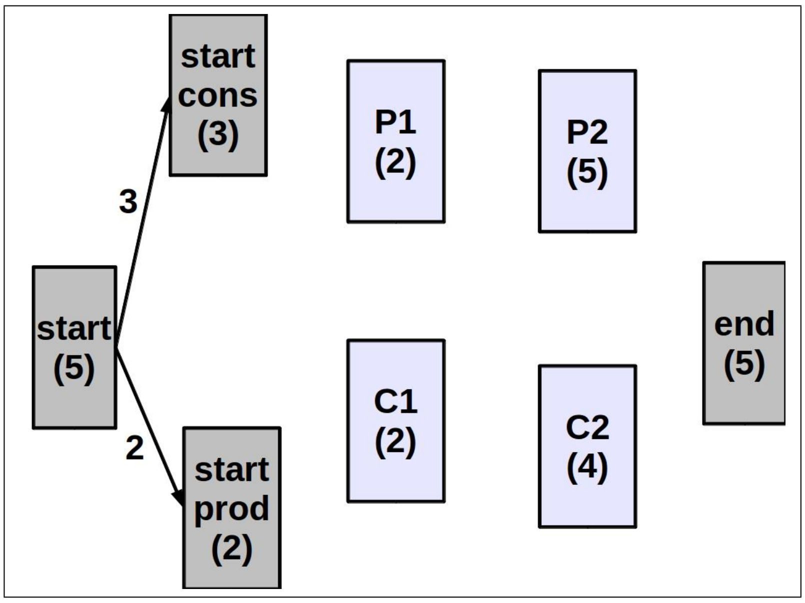


Figure 4.4: Inference on Reservoir resource

Inference 6 on Reservoir Resources: Figure 4.4 illustrates how we can infer lower bound on the start times of transitions on a reservoir resource r, that has $\mathrm{capacity}(r) = 5$, and $\mathrm{init}(r) = 2$. There are 4 resource transitions on r that could execute on r if included in the plan, where $C1$ and $C2$ are CONSUME transitions that can consume 2 and 4 units of resource respectively, and $P1$ and $P2$ are PRODUCE transitions that can produce 2 and 5 units resource respectively. All these 4 transitions have duration of 2 time units. $start$ and end represent the dummy start and end transitions on r. Since, r is a reservoir resource the dummy start CONSUME transition, $startCons$, produces 3 units of free-space and the dummy start PRODUCE transition, $startProd$, produces 2 units of resource. $start$ provides support to both dummy start CONSUME and PRODUCE transitions. Each weighted edge between transitions in the figure represents a *support-link*. Note that $start$, end, $startProd$, and $startCons$ are included in the plan, and the support links $\mathrm{support}(start, startCons) = 3$ and $\mathrm{support}(start, startProd) = 2$ are established.

For transition $C1$, $\mathrm{PossSupp}(C1) = \{startProd, P1, P2\}$. Since $startPord$ can provide enough support for $C1$ we can infer that $C1$ can start earliest at time point 0. Similarly, we can infer that $P1$ can start earliest at 0. For transition $C4$, $\mathrm{PossSupp}(C4) = \{startProd, P1, P2\}$, same as $C1$. We can see that together $startProd$ and $P1$ can provide enough support to $C4$, this means that the earliest start time of $C4$ is 2 time units, because earliest finish time of $P1$ is 2. Similarly, we can infer that the earliest start time of $P2$ is 2, because it could only start after $startCons$ and $C1$, and earliest end time of $C1$ is 2. Note that the situation in the Figure 4.4, describes a situation where none of the transitions (except for the dummy transitions) are included in the plan and no support links involving these transitions are established. Using the Inference 6 we are able to deduce that $C2$ and $P2$ can start earliest at time point 2.

Inference 6 on Reusable Resources: For a reusable resource, Inference 6 can not deduce a better lower bound on the start time of transitions on it if no decisions have been made on the support links on the resource. This is because, for each transition T_r that could execute on the reusable resource r, initially T_r^{start} will be in $\mathrm{PossSupp}(T_r)$. Since for each reusable resource r, $\mathrm{req}(T_r^{start}) = \mathrm{init}(r) = \mathrm{capacity}(r)$, for each transition T_r, T_r^{start} alone would be enough to provide support to T_r, and earliest end time of T_r^{start} is always 0.

However, during search after we made some decisions on the support links, Inference 6 could infer better lower bound for transitions on r. Figure 4.5 illustrates a situation on a reusable resource r with $\mathrm{capacity}(r) = 5$, and 4 BORROW transitions, $T1$ to $T4$, each having duration 2, where the following decisions have been made: $\mathrm{support}(start, T1) = 3$, $\mathrm{support}(start, T2) = 2$, and $\mathrm{support}(T1, T3) = 1$. Like before, $start$ and end denote the dummy start and end transition on r. Note that except for $T4$, all other transitions in the Figure 4.5 are included in the plan. Given the decisions that have already been made (each edge represents a decision, a *support-link*), we are interested to infer the lower bound on the start time of $T4$ that requires 5 units of resource. For $T4$, $\mathrm{PossSupp}(T4) = \{T1, T2, T3\}$, where $T1$ and $T2$ can provide 2 units of resource each, and $T3$ can provide 1 unit of resource. This means that using Inference 6 we can infer that $T4$ could only start at time point 4, because the maximum of the earliest end times of the transitions in the set $\{T1, T2, T3\}$ is 4 (we assume

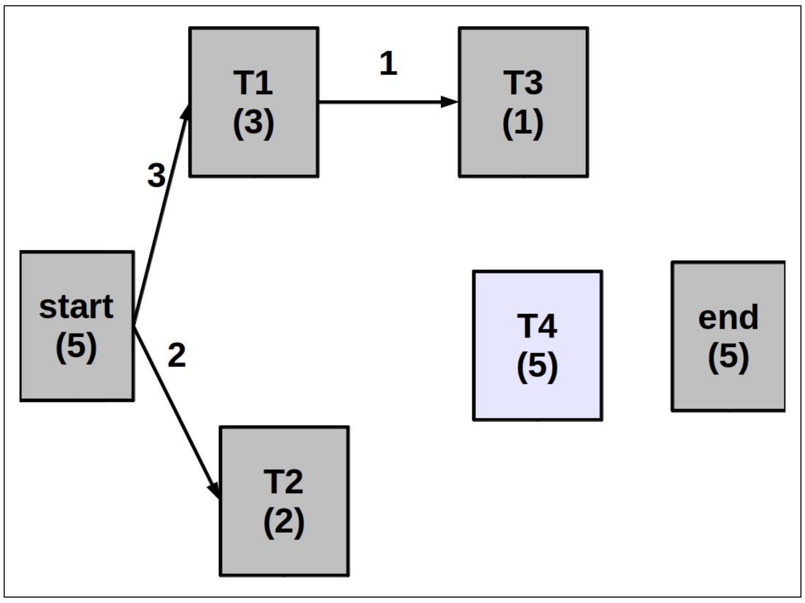


Figure 4.5: Inference on Reusable resource

that the earliest start time of $T1$ and $T2$ is 0).

Special cases of Inference 6: Inference 6 describes how to get a lower bound on the start time of a transition based on its PossSupp set. For a transition T_r, such that T_r is included in the plan, we can infer support links related to T_r using the following two inference rules.

For a resource transition T_r, such that $\text{inplan}(T_r) = \textit{true}$, if $\text{PossSupp}(T_r)$ contains only one transition T'_r, then we can deduce that T'_r must provide support to T_r. This means that for each transition T_r we apply the following inference rule:

Inference 7. *For each transition T_r, if* $\text{inplan}(T_r) = \textit{true}$ *and* $|\text{PossSupp}(T_r)| = 1$, *then*

$$\forall T'_r \in \text{PossSupp}(T_r) : \textit{set}: \text{support}(T'_r, T_r) = ub(\text{support}(T'_r, T_r)) \tag{4.48}$$

Similarly, for a transition T_r, where $\text{inplan}(T_r) = \textit{true}$, if the total possible support from all transitions in $\text{PossSupp}(T_r)$ is equal to the remaining demand of T_r, then we can deduce that all these transitions must support to T_r.

Inference 8. *For each transition T_r, if*

$$\text{inplan}(T_r) = \textit{true and} \sum_{T'_r \in \text{PossSupp}(T_r)} ub(\text{support}(T'_r, T_r) = \text{RemDemand}(T_r)$$

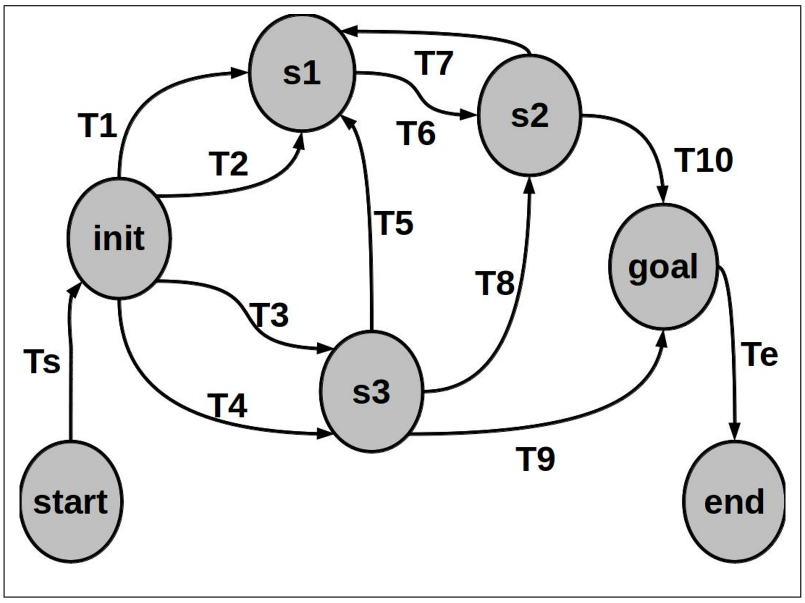


Figure 4.6: Inference on State Variable

then $\forall T' \in \text{PossSupp}(T_r)$

$$set: \ \text{support}(T'_r, T_r) = ub(\text{support}(T'_r, T_r)) \tag{4.49}$$

Note that the Algorithm 1 can be easily extend to implement Inference 7 and Inference 8.

4.4.1.2 For State Variable Transitions

For each state variable transition T_{sv} on a state variable sv, a similar principle can be applied to estimate a lower bound on the start time of T_{sv}. Let $\text{PossAchiev}(T_{sv})$ represent the set of transitions that can achieve the pre-condition of T_{sv}, this means that each $T'_{sv} \in \text{PossAchiev}(T_{sv})$, $\text{achieve}(T'_{sv}, T_{sv})$ is *undecided.* Each state variable transition T_{sv} needs only one transition from $\text{PossAchiev}(T_{sv})$. Let $T^{min}_{sv} \in \text{PossAchiev}(T_{sv})$ be a transition that has the minimum of the earliest end times of transition in $\text{PossAchiev}(T_{sv})$. The earliest time when T_{sv} can start is the earliest finish time of T^{min}_{sv}. For each non-excluded state variable transition we propagate the following rule.

Inference 9. *For each state variable transition T_{sv}, such that T_{sv} is not excluded from the plan:*

$$\text{start}(T_{sv}) \geq \min_{T'_{sv} \in \text{PossAchiev}(T_{sv})} lb(\text{end}(T'_{sv})) \tag{4.50}$$

Figure 4.6 illustrates an initial situation on a state variable sv, where *init* and *goal* are the initial and goal state of sv. *Ts* and *Te* are the dummy start and end transitions on sv.

All transitions on sv are *undecided*, except for Ts and Te that must be included in the plan. Here we assume that each transition from $T1$ to $T10$ have duration 2 time units. Given this situation, Inference 9 infers the tightest possible lower bound on the start times for transitions. For example, consider the transition $T3$, that has $\mathrm{PossAchiev}(T3) = \{Ts\}$. Inference 9 deduces that $T3$ can start earliest at time point 0. We can derive the same for $T4$. For $T9$, where $\mathrm{PossAchiev}(T9) = \{T3, T4\}$, we infer that $T9$ can only start earliest at 2. Similarly we can deduce that the earliest start time for Te is 4. This means that, from the initial situation, using the Inference 9, we can deduce that for sv goal can be achieved earliest at 4 time points.

As we have shown for resource transitions, based on PossAchiev set of a transition we can infer the achiever of the pre-condition of the transition using the following rule:

Inference 10. *For each state variable transition T_{sv}, if* $\mathrm{inplan}(T_{sv}) = true$ *and* $|\,\mathrm{PossAchiev}(T_{sv})| = 1$*, then we infer the following:* $\forall T'_{sv} \in \mathrm{PossSupp}(T_{sv})$

$$set:\ \mathrm{achieve}(T'_{sv}, T_{sv}) = 1 \tag{4.51}$$

This rule is the unit propagation rule for achieve variables.

4.4.2 Lower Bound from Active Precedence Constraints

Given a precedence constraint $T \rightarrow T'$, it implies that T' must start after T finishes, where T and T' belong to the same resource or state variable. If T is included in the plan, i.e. $\mathrm{inplan}(T) = true$, then call the precedence constraint $T \rightarrow T'$ an *active precedence constraint* for T'.

Definition 18. ***Active Precedence Relation***
For each pair of transitions $< T, T' >$, where $\mathrm{obj}(T) = \mathrm{obj}(T')$, the precedence relation $T \rightarrow T'$ is called an active precedence relation for T', if T is included in the plan, i.e. $\mathrm{inplan}(T) = true$.

For each transition T, we define a set *UnSupported Before (USB)* as follows:

Definition 19. ***UnSupported Before (USB)***
For each transition T, the set of all transitions T' such that:

- *$T' \rightarrow T$ is an active precedence relation for T*
- *if T is a resource transition, then T'_r is not yet fully supported, i.e.*

$$\mathrm{RemDemand}(T'_r) > 0$$

- *if T is a state variable transition on a state variable sv, then its pre-condition is not yet achieved, i.e.*
$$\nexists < T', T > \in \mathrm{AC}(sv) : \mathrm{achieve}(T', T) = 1$$

is called the UnSupported Before (USB) set of T, and denoted by $\mathrm{USB}(T)$*.*

For a transition T, the transitions in $\mathrm{USB}(T)$ must be scheduled on the domain object $(\mathrm{obj}(T))$, before T can start its execution.

Frontier: Let $\mathrm{frontier}(T)$ be a schedule of the transitions in $\mathrm{USB}(T)$ on $\mathrm{obj}(T)$. If T is a state variable transition, then each transition in $\mathrm{frontier}(T)$ must be totally ordered, except for the PREVAIL transitions that require same state. If T is a resource transition, then $\mathrm{frontier}(T)$ must be a *safe schedule* (as described in Definition 7, page 7) that satisfies the resource requirements of the transitions without over- or under-flowing the resource $\mathrm{obj}(T)$.

Makespan of Frontier: Let $\mathrm{makespan}(\mathrm{frontier}(T))$ denotes the *makespan* of the schedule $\mathrm{frontier}(T)$, which the maximum of the earliest end times of the transitions in the schedule, i.e.

$$\mathrm{makespan}(\mathrm{frontier}(T)) = \max_{T' \in \mathrm{frontier}(T)} lb(\mathrm{end}(T')) \tag{4.52}$$

Transition T can only start after the schedule $\mathrm{frontier}(T)$. Given a transition T, there can be multiple $\mathrm{frontier}(T)$. Let $\mathrm{frontier}(T)^{min}$ represent a $\mathrm{frontier}(T)$ that has the optimal makespan. We can deduce that T must start after $\mathrm{frontier}(T_r)^{min}$, i.e.

$$\mathrm{start}(T) \geq \mathrm{makespan}\left(\mathrm{frontier}(T)^{min}\right) \tag{4.53}$$

In this section we describe for a transition T, how we estimate an admissible bound on $\mathrm{makespan}(\mathrm{frontier}(T)^{min})$. In the following, first we describe how we do it for resource transitions and then we describe for state variable transitions.

4.4.2.1 For Resource Transitions

There are two types of resource: reusable and reservoir, and three types of resource transitions: BORROW on reusable resources, and CONSUME and PRODUCE on reservoir resources. On a reusable resource r, given a set of BORROW transitions, it is always possible to create a safe schedule if we ignore any constraints on the end times of transitions. This is the case because each transition consume some resource in the beginning and produce the same amount at the end, if we can delay transitions they will eventually get a chance to execute on the resource. This is not true for reservoir resources. For a reservoir resource, given a set of CONSUME and PRODUCE transitions, existence of a safe schedule will depend on the amount of resource

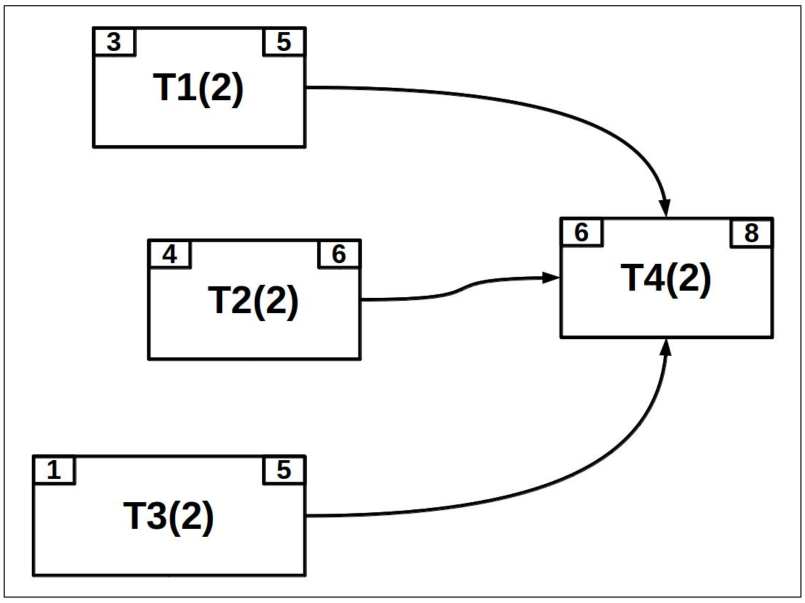


Figure 4.7: Example of $\text{USB}(T4)$

available in the beginning, and if there exists enough CONSUME transitions to support all PRODUCE transitions and enough PRODUCE transitions to support all CONSUME transitions.

Generalize Resource Requirements based on Free-Space: For a transition T_r on a resource r, irrespective of the type of T_r and r, one fact is always true: T_r reserves $\mathsf{req}(T_r)$ amount of $\mathsf{free\text{-}space}$ on r with the interval $[\mathsf{start}(T_r), \mathsf{end}(T_r))$. In any safe schedule of transitions on a resource, total reservation any overlapping set of transitions must be less than or equal to $\mathsf{capacity}(r)$. On each reservoir resource, to estimate $\mathsf{makespan}(\mathsf{frontier}(T))$ we ignore the type of transitions on the reservoir resource, instead we consider the following:

- Each reservoir resource r is a reusable resource that can provide $\mathsf{capacity}(r)$ amount of $\mathsf{free\text{-}space}$ at start
- Each transition on r consumes $\mathsf{req}(T_r)$ amount of $\mathsf{free\text{-}space}$ at the start, and produces $\mathsf{req}(T_r)$ amount of $\mathsf{free\text{-}space}$ at the end.

This means that we relax a reservoir resource as a reusable resource, by considering each pair of transitions on the reservoir resource as a **support-relevant** pair (with respect to $\mathsf{free\text{-}space}$). With this relaxation all resources will be considered as reusable resource, and we can create a safe schedule on them by ignoring any constraints on end times of the transitions.

Figure 4.7 illustrates an example where we have a resource transition $T4$ and its $\text{USB}(T4) = \{T1, T2, T3\}$, where $T1, T2$ and $T4$ have duration of 2, and $T3$ has duration of 4. Each transition has requirement of 2, and the resource has capacity of 4. The earliest start time for $T1$

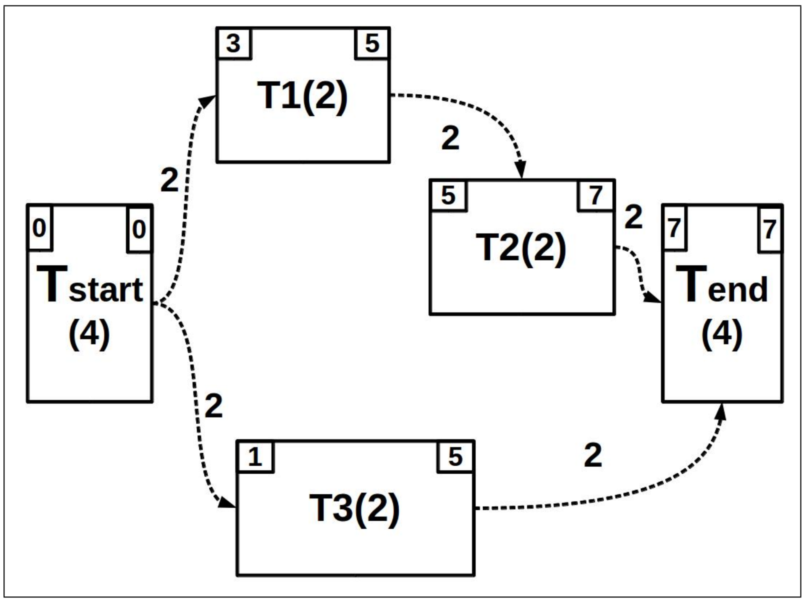


Figure 4.8: Example of $\mathsf{frontier}(T4)^{min}$

is 3, for $T2$ it is 4, for $T3$ it is 1, and for $T4$ it is 6. Each edge between a pair of transitions represents an active precedence constraint.

Given a transition T_r on a resource r, and its $\mathsf{USB}(T_r)$ we need to find a schedule, denoted by $\mathsf{frontier}(T_r)^{min}$, that solves the following scheduling problem optimally (w.r.t. makespan):

Definition 20. ***The Resource Allocation Problem***
Find a partial order schedule on a multi-capacity reusable resource r ($\mathsf{capacity}(r)$), for a set of activities (the transitions in $\mathsf{USB}(T_r)$), where each activity has a release date (earliest start time of the transition) and a fixed duration (duration of the transition).

Note that each partial order schedule on a resource r, starts with the dummy start transition T_r^{start}, ends with the dummy end transition T_r^{end}, and each transition, except T_r^{start}, must be *fully supported* via *support links* from other transitions in the partial order schedule. This means that $\mathsf{frontier}(T_r)^{min}$ is a partial order schedule on r, that solves the resource allocation problem defined in Definition 20, and has the optimal makespan. Note that makespan of a partial order schedule on a resource r is earliest start time of T_r^{end}. Figure 4.8 shows an example of a $\mathsf{frontier}(T4)^{min}$ for the resource allocation problem illustrated in the Figure 4.7. T_{start} and T_{end} represent the dummy start and end transition respectively on the resource r. The dotted weighted edges between transitions represents the support links. The makespan of $\mathsf{frontier}(T4)^{min}$ is 7 in this example.

Calculation of the Optimal Makespan Schedule

To calculate $\mathsf{frontier}(T_r)^{min}$, we use a data structure called a *frontier queue (*FQ*)*, which is a sorted list of *support nodes (SN)*, where each support node represents a transition in $\text{USB}(T_r)$. We use the notation $SN(T'_r)$ to represent a support node corresponding to the transition $T'_r \in \text{USB}(T_r)$. For each $SN(T'_r)$ we have the following attributes and values:

- $SN(T'_r).demand = \text{req}(T'_r)$
- $SN(T'_r).support = \text{req}(T'_r)$
- $SN(T'_r).dur = \text{dur}(T'_r)$
- $SN(T'_r).start = lb(\text{start}(T'_r))$
- $SN(T'_r).end = SN(T'_r).start + SN(T'_r).dur$

The FQ maintains the following two invariant properties:

1. Support nodes in FQ are ordered in non-decreasing *end* values.
2. For each SN in FQ, $SN.demand = 0$.

Algorithm 2: Algorithm 2 describes how a support node SN is added to FQ, such that FQ maintains the property that each support node in FQ is fully supported. It scans the list from the beginning and accumulates support for SN. Note that the support nodes in FQ are always ordered in non-decreasing *end* values. Each time a support has been found from an existing support node SN', Algorithm 2 updates the remaining demand of SN and remaining support of SN' (lines 3-6). Since these supports induce precedence constraints, it updates the $SN.start$ value accordingly (line 7). Note that, this represents posting a *support-link* between SN' and SN. After the demand of SN is fulfilled, Algorithm 2 adds SN into FQ (line 12), which refers to two steps: adding SN into FQ and resorting FQ.

Algorithm 2 ADD SN in FQ

```
1:  for each $SN'_i \in$ FQ, where $i = 1$ to | FQ | do
2:    if $SN.demand > 0$ then
3:      n_demand = $\max(0, SN.demand - SN'_i.support)$
4:      n_support = $\max(0, SN'_i.support - SN.demand)$
5:      $SN'_i.support$ = n_support
6:      $SN.demand$ = n_demand
7:      $SN.start = \max((SN'_i.end, SN.start)$
8:      $SN.end = SN.start + SN.dur$
9:    else
10:     break
11:   end if
12: end for
13: add $SN$ into FQ
```

Algorithm 3: For each transition T_r Algorithm 3 calculates the makespan of the schedule $\text{frontier}(T_r)^{min}$. It first adds a support node corresponding to the dummy start transition T_r^{start}, $SN(start)$, in FQ (line 1). $SN(start)$ is initialized as follows:

- $SN(start).demand = 0$
- $SN(start).support = \text{req}(T_r^{start})$
- $SN(start).dur = \text{dur}(T_r^{start})$
- $SN(start).start = 0$

Note that demand of $SN(start)$ is 0, because we assume T_r^{start} is always fully supported, and its earliest start time (release date) is 0. After adding $SN(start)$, Algorithm 3 adds the support nodes corresponding to the transitions in $\text{USB}(T_r)$ in the non-decreasing order of their earliest start time (lines 2-5). Each support node is initialized as described above. After adding all support nodes for transitions in $\text{USB}(T_r)$ to FQ, it adds a support node $SN(end)$ that corresponds to the dummy end transition T_r^{end} (line 6), which in initialized as follows:

- $SN(end).demand = \text{req}(T_r^{end})$
- $SN(end).support = 0$
- $SN(end).dur = \text{dur}(T_r^{end})$
- $SN(end).start = 0$

The support of $SN(end)$ is 0, because T_r^{end} can not provide support to any other transition. Note that the earliest start time of $SN(end)$ is 0. This is because $SN(end)$ corresponds to the dummy end transition on r for the the resource allocation problem as defined in Definition 20, not the original T_r^{end} of the planning problem. After adding the support node $SN(end)$ in FQ, Algorithm 3 returns the end value of $SN(end)$. The FQ created by Algorithm 3 is a schedule

Algorithm 3 $get\_\text{est}(T_r)$

```
1: Add $SN(start)$ in FQ.
2: Sort $\text{USB}(T_r)$ non-decreasing est
3: for each $T_i' \in \text{USB}(T_r)$, where $i = 1$ to $|\text{USB}(T_r)|$ do
4:     add $SN(T_i')$ in FQ
5: end for
6: Add $SN(end)$ in FQ.
7: return $SN(end).end$
```

$\text{frontier}(T_r)^{min}$ that has the optimal makespan, and returns the end time of the support node $SN(end)$, which is the makespan of $\text{frontier}(T_r)^{min}$.

Since Algorithm 3 calculates the optimal shcedule of the USB set, we update the lower bound on the start times of transitions via the following inference rule:

Inference 11. *For all resource transition T_r, such that* $\text{inplan}(T_r) \neq false$,

$$\text{start}(T_r) \geq get\_\text{est}(T_r) \tag{4.54}$$

Proof of Correctness

Let $seq(\text{frontier}(T_r)^{min})$ be a topological sort of the partial order schedule $\text{frontier}(T_r)^{min}$. Note that in the sequence $seq(\text{frontier}(T_r)^{min}$ the first transition is the dummy start transition and last transition is the dummy end transition on r.

Proposition 1. *If we add transitions using Algorithm 2, in the same sequence as in* $seq(\text{frontier}(T_r)^{min})$ *the end time of the last element in the resultant* FQ *is the makespan of* $\text{frontier}(T_r)^{min}$.

Proof: To show this we will show that when a support node of a transition is added in FQ, it starts at its earliest start time. We prove this by induction.

Base Case: Note the first support node to add in FQ is the support node $SN(start)$ which corresponds to the dummyT_{start} transition, that has $demand = 0$ and $start = 0$. Algorithm 2 adds the support node immediately without updating the start time. It means the support node $SN(start)$ can start at time point 0, which is its earliest start time.

Inductive Step: Let assume that the support node corresponding to the i^{th} transition in $seq(\text{frontier}(T_r)^{min})$ starts at its earliest start time. When we add the support node corresponding to the $i+1^{th}$ transition in $seq(\text{frontier}(T_r)^{min})$ into FQ, Algorithm 2 finds support for its demand from the support nodes that are already in FQ sequentially starting from the first element in FQ. Note that support nodes in FQ are fully supported and sorted in the non-decreasing order on the end times. All these support nodes are the only possible supporters of $SN(T_{i+1})$, because the corresponding transitions of these support nodes appear before T_{i+1} in $seq(\text{frontier}(T_r)^{min})$. This means that each time Algorithm 2 finds a support for the demand of $SN(T_{i+1})$, it would be earliest possible time that $SN(T_{i+1})$ can get the support. This means that the start time that Algorithm 2 determines for the support node $SN(T_{i+1})$ is the earliest possible start time for T_{i+1}.

This means that each transition that is added in FQ always start at its earliest start time. The last support added to FQ is the support node $SN(end)$, because the last transition in $seq(\text{frontier}(T_r)^{min})$ is the dummy end transition on r. Since Algorithm 2 determine the earliest start time of each support node, $SN(end)$ will start at its earliest start time, which is the makespan of $\text{frontier}(T_r)^{min}$. Since $SN(end).dur = 0$, the end time of $SN(end)$ is same as its start time.

This proves that if we add transitions using Algorithm 2, in the same sequence as in $seq(\text{frontier}(T_r)^{min})$ the end time of the last element in the resultant FQ is the makespan of $\text{frontier}(T_r)^{min}$. □

Proposition 2. *For a transition T_r, Algorithm 3 returns optimal makespan of a partial order schedule of the transitions in* $\mathrm{USB}(T_r)$ *on r.*

Proof: Suppose $seq(\mathtt{frontier}(T_r)^{min})$ is a topological sort of the partial order schedule $\mathtt{frontier}(T_r)^{min}$, where the first transition is the dummy start transition, and last transition in the dummy end transition.

For any two transition T_i and T_{i+1} in the sequence $seq(\mathtt{frontier}(T_r)^{min})$, such that $lb(\mathtt{start}(T_i)) > lb(\mathtt{start}(T_{i+1}))$ and T_i and T_{i+1} are not the dummy start or end transition, if there exists a third transition T_k, such that $k > i + 1$ and the precedence relations $T_i \rightarrow T_k$ and $T_{i+1} \rightarrow T_k$ hold in the partial order schedule, then The lower bound of the start time of T_k will depend on the following two cases (among other things):

- **case 1**: If $T_i \rightarrow T_{i+1}$ doesn't hold in the partial order schedule, then the lower bound on the start time of T_k in the partial order schedule would be atleast:

$$\max\{lb(\mathtt{start}(T_i)) + \mathtt{dur}(T_i), lb(\mathtt{start}(T_{i+1})) + \mathtt{dur}(T_{i+1})\} \tag{4.55}$$

- **case 2**: If $T_i \rightarrow T_{i+1}$ holds in the schedule, the lower bound on the start time of T_k would be atleast:

$$\max\{lb(\mathtt{start}(T_i)) + \mathtt{dur}(T_i) + \mathtt{dur}(T_{i+1}), lb(\mathtt{start}(T_{i+1})) + \mathtt{dur}(T_{i+1})\} \tag{4.56}$$

If we swap the positions of T_i and T_{i+1}, then for case 1 above there will be no effect on the lower bound on the start time of T_k. If case 2 is true, then the lower bound on the start time of T_k would be atleast:

$$\max\{lb(\mathtt{start}(T_{i+1})) + \mathtt{dur}(T_{i+1}) + \mathtt{dur}(T_i), lb(\mathtt{start}(T_i)) + \mathtt{dur}(T_i)\} \tag{4.57}$$

Note that the lower bound from equation 4.57, can not be greater than the lower bound from equation 4.56. This is the case because both terms in equation 4.57 is smaller than the first term in equation 4.56 (recall that $lb(\mathtt{start}(T_i)) > lb(\mathtt{start}(T_{i+1}))$. Applying this repeatedly, we can find another sequence, where the dummy start transition appears in the first position and dummy end transition appears in the last position, and all transitions in between are sorted in non-decreasing order of their earliest start time, which have the same makespan as $seq(\mathtt{frontier}(T_r)^{min})$. This means that the new sequence is also a topological sort of the partial order schedule $\mathtt{frontier}(T_r)^{min}$. Algorithm 3 add transitions exactly in this sequence. In Proposition 1,we have shown that if we add support node corresponding to the transitions in this order, it return the makespan of $\mathtt{frontier}(T_r)^{min}$. □

Comparison with the Energy Precedence Constraint

This inference technique can be seen as a generalization of the *Energy Precedence Constraint* [36] on reusable resources. Given a reusable resource r, for each transition T_r, if there exists a set of transitions $before(T_r)$, where each transition $T'_r \in before(T_r)$, T'_r must execute on r, and $T'_r \rightarrow T_r$ holds, then the energy precedence constraint infers the earliest start time of T_r as the following:

$$\mathtt{start}(T_r) \geq \left(\min_{T'_r \in before(T_r)} lb(\mathtt{start}(T'_r)) \right) + \left\lfloor \frac{\sum_{T'_r \in before(T_r)} \mathtt{req}(T'_r) * \mathtt{dur}(T'_r)}{\mathtt{capacity}(r)} \right\rfloor \tag{4.58}$$

Consider the example described in the Figure 4.7 (see Section 4.4.2.1 on page 95). The energy precedence constraint calculates the earliest start time of $T4$ as follows:

$$\begin{aligned} \mathtt{start}(T4) &\geq lb(\mathtt{start}(T_1)) + \left\lfloor \frac{(2*2+2*4+2*2)}{4} \right\rfloor \\ &\geq 1 + \left\lfloor \frac{16}{4} \right\rfloor \\ &\geq 1 + 4 \\ &\geq 5 \end{aligned} \tag{4.59}$$

For this example, Inference 11 which uses Algorithm 3 to calculate the makespan of a schedule on the USB set, derives the earliest start time for $T4$ to 7, which is tighter than the value (5) deduced by the energy precedence constraint. This is because both energy precedence constraint and Inference 11, estimate the makespan of a schedule for the resource allocation problem (Definition 20), but Inference 11 does that optimally (Porposition 2). This means that we can claim the following:

Proposition 3. *Given a transition T_r and its* $\mathrm{USB}(T_r)$, *where r is a reusable resource, Inference 11 derives a lower bound on the start time of T_r that is always atleast as good as a lower bound derived by the energy precedence constraint.*

4.4.2.2 For State Variable Transitions

For each state variable sv, we can propagate better lower bounds on the start times of transitions on the state variable by considering what must happen before each transition. For a transition T_{sv}, $\mathrm{USB}(T_{sv})$ represents the set of transitions that must occur before T_{sv}, and whose pre-conditions are not achieved. Recall that all transitions on a state variable must be totally ordered, except for the PREVAIL transitions that need same state of the state variable to execute. Let $\mathrm{USB}(T_{sv})^{linear} \subseteq \mathrm{USB}(T_{sv})$ represent a subset of transitions such that there exists no two PREVAIL transitions that need same state in $\mathrm{USB}(T_{sv})^{linear}$.

For each T_{sv}, $\mathrm{USB}(T_{sv})$ may contain more than one PREVAIL transitions that need same

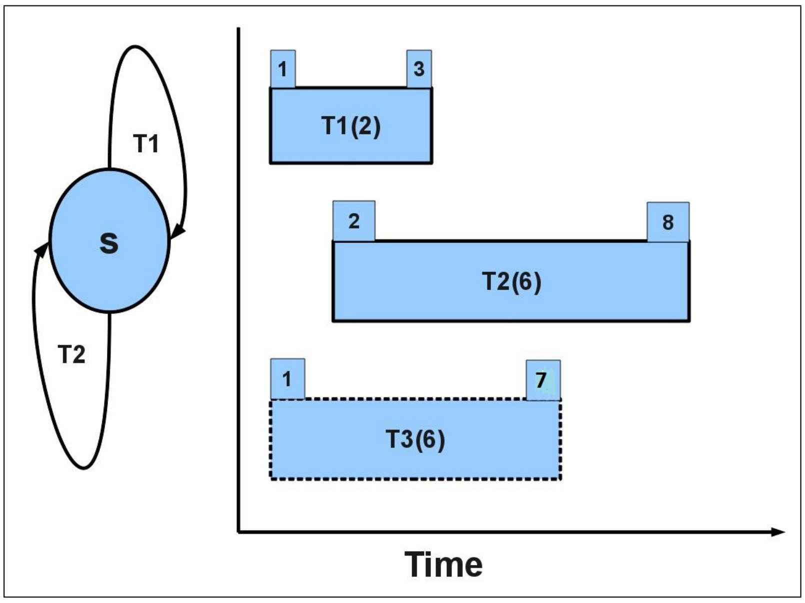


Figure 4.9: Example of PREVAIL estimation

state. Given $\text{USB}(T_{sv})$ we estimate $\text{USB}(T_{sv})^{linear}$ as follows:

1. First we make a copy of $\text{USB}(T_{sv})$ and rename it to $\text{USB}(T_{sv})^{linear}$.

2. Then for each state $s \in \text{dom}(sv)$ of the state variable sv we create a set par_s, that contains the PREVAIL transitions from the set $\text{USB}(T_{sv})^{linear}$ that need the state s.

3. We optimistically assume that all active PREVAIL transitions in each par_s will maximally overlap. For each set of PREVAIL transitions in each non-empty par_s, we create a new PREVAIL transition T_s^p, where the earliest start time of T_s^p equals the minimum of the earliest start times of the transitions in par_s, i.e.

$$\text{est}(T_s^p) = \min_{T'_{sv} \in par_s} \text{est}(T'_{sv})$$

   and the duration of T_s^p equals to the maximum of the durations among the transitions in par_s, i.e.

$$\text{dur}(T_s^p) = \max_{T'_{sv} \in par_s} \text{dur}(T'_{sv})$$

   Since T_s^p is a non-preemptive, the earliest end of T_{sv}^p is the earliest start plus the duration, i.e.

$$\text{eft}(T_s^p) = \text{est}(T_s^p) + \text{dur}(T_s^p)$$

4. For each non-empty par_s, we delete all PREVAIL transitions in it from $\text{USB}(T_{sv})^{linear}$

and add the corresponding PREVAIL transition T_s^p in $\mathrm{USB}(T_{sv})^{linear}$.

Figure 4.9 describes an example where for par_s has two PREVAIL transitions: $T1$ and $T2$, where $\mathrm{est}(T1) = 1$, $\mathrm{dur}(T1) = 2$, $\mathrm{est}(T2) = 2$, and $\mathrm{dur}(T2) = 6$. So $par_s = \{T1, T2\}$. Transition $T3$ represents the replacement PREVAIL transition for $T1$ and $T2$, where $\mathrm{est}(T3) = 1$ and $\mathrm{dur}(T3) = 6$. For a transition T_{sv}, if $T1, T2 \in \mathrm{USB}(T_r)$, then we delete $T1$ and $T2$, and add $T3$ in $\mathrm{USB}(T_{sv})^{linear}$. This means that all the transitions in $\mathrm{USB}(T_{sv})^{linear}$ must be executed sequentially before T_{sv}.

Prefix: We call a sequential execution of the transitions in $\mathrm{USB}(T_{sv})^{linear}$ a **prefix** of T_{sv}, denoted as $\mathrm{prefix}(T_{sv})$. The earliest finish time of a $\mathrm{prefix}(T_{sv})$ is the earliest finish time of last transition in $\mathrm{prefix}(T_{sv})$. Let $\mathrm{prefix}(T_{sv})^{min}$ denotes a prefix of T_{sv} that has the minimum of the earliest finish times among all possible prefix sequences for T_{sv}. T_{sv} can only start execution after $\mathrm{eft}(\mathrm{prefix}(T_{sv})^{min})$.

Algorithm 4: Algorithm 4 calculates the $\mathrm{eft}(\mathrm{prefix}(T_{sv}^{min}))$. It first creates $\mathrm{USB}(T_{sv})^{linear}$ from $\mathrm{USB}(T_{sv})$, and then sort $\mathrm{USB}(T_{sv})^{linear}$ in the non-decreasing order of the est of the transitions, and then calculate the end time of the sequence.

Algorithm 4 get_prefix_end(T_{sv})

```
1: Create $\mathrm{USB}(T_{sv})^{linear}$ from $\mathrm{USB}(T_{sv})$
2: Initialize $\mathrm{prefix}\,.end = 0$.
3: Sort $\mathrm{USB}(T_{sv})^{linear}$ in non-decreasing earliest start times
4: for each $T_i' \in \mathrm{USB}(T_{sv})^{linear}$, where $i = 1$ to $|\,\mathrm{USB}(T_{sv})^{linear}|$ do
5:     $\mathrm{prefix}\,.end = \max\left((\mathrm{prefix}\,.end + \mathrm{dur}(T_i'), \mathrm{eft}(T_i)\right)$
6: end for
7: return $\mathrm{prefix}\,.end$
```

For each state variable sv we propagate the following rule:

Inference 12. *For each state variable transition T_{sv}, such that* $\mathrm{inplan}(T_{sv}) \neq false$,

$$\mathrm{start}(T_{sv}) \geq get\_\mathrm{prefix}\_end(T_{sv}) \tag{4.60}$$

Proof of Correctness

On a state variable all transitions, except for the PREVAIL transitions that require same state, must be totally ordered. As we have discussed earlier, a state variable can be seen as a reusable resource with capacity 1, where all transitions must be sequenced. Given a transition T_r and its $\mathrm{USB}(T_r)$, where r is a reusable resource with capacity 1, Algorithm 3 returns the optimal makespan of the schedule on r of the transitions in $\mathrm{USB}(T_r)$. In this case the schedule is a sequence of transitions with optimum makespan. Algorithm 4 can be seen as a special case of Algorithm 3, where it sequences the transitions in $\mathrm{USB}(T_{sv})^{linear}$, and return the makespan.

Since it is a spacial case of Algorithm 3, it returns the optimal makespan, which is indeed a lower bound on the start time of T_{sv}.

4.4.3 Inferring Upper Bounds of End Times

We have described here how to infer lower bounds on the start times of transitions, by considering how to satisfy the pre-conditions and requirements, and what must happen before a transition. We can use similar techniques to infer upper bounds on the end times of transitions, by considering how post-conditions of transitions could satisfy pre-conditions of other transitions, and what must happen after a transition. We don't discuss these techniques here, because the basic principle remains the same, except for the PREVAIL transitions.

4.4.3.1 For PREVAIL transitions

A PREVAIL transition T^p_{sv} on a state variable sv does not achieve pre-condition of any transition, but we know that if T^p_{sv} is included in the plan there must be an EFFECT transition that must immediately follow it, because a PREVAIL transition always executes between two EFFECT transitions.

Possible Followers: Let $\text{PossFollow}(T^p_{sv})$ represent a set of EFFECT transition that can follow T^p_{sv} if T^p_{sv} is included in the plan. This means that for each transition $T_{sv} \in \text{PossFollow}(T^p_{sv})$, $< T^p_{sv}, T_{sv} > \in \text{FL}(sv)$ and $ub(\text{follow}(T^p_{sv}, T_{sv})) > 0$. Since there will be exactly one $T \in \text{PossFollow}(T^p_{sv})$ could follow T^p_{sv} in the final plan, we infer the upper bound of the end time of T^p_{sv} as follows:

Inference 13. *For each PREVAIL transition T^p_{sv},*

$$\text{end}(T^p_{sv}) \leq \max_{T_{sv} \in \text{PossFollow}(T^p_{sv})} ub(\text{start}(T_{sv})) \tag{4.61}$$

For a PREVAIL transition, T^p_{sv} if T^p_{sv} is active and there is only one EFFECT transition T_{sv} in $\text{PossFollow}(T^p_{sv})$, then we can deduce that T_{sv} must follow T^p_{sv}.

Inference 14. *For each PREVAIL transition T^p_{sv}, if* $\text{inplan}(T^p_{sv}) = true$ *and* $|\,\text{PossFollow}(T^p_{sv})| = 1$, *then we infer the following:* $\forall T'_{sv} \in \text{PossFollow}(T^p_{sv})$

$$set:\ \text{follow}(T^p_{sv}, T_{sv}) = 1 \tag{4.62}$$

4.5 Support Inference from Precedence Constraints

Given a support-relevant pair of transitions $< T_r, T'_r >$ on a resource r, in this section we describe how to infer an upper bound on $\text{support}(T_r, T'_r)$, by analyzing what must happen in between T_r and T'_r. Note that $\text{support}(T_r, T'_r) = \delta$ means that T_r provides δ amount of support to T'_r directly. For each pair $< T_r, T'_r > \in \text{SUP}(r)$, we define **must be between (MB)** set of transitions as follows.

Definition 21. ***Must be Between Set***
On each resource r, for each support-relevant pair $< T_r, T'_r > \in \text{SUP}(r)$, the Must be Between set, denoted as $MB(T_r, T'_r)$, is a set of all transitions T''_r that satisfies the following conditions:

- *T''_r is included in the plan, i.e.* $\text{inplan}(T''_r) = true$
- *T''_r is not fully supported, i.e.* $\text{RemDemand}(T''_r) > 0$
- *$T_r \rightarrow T''_r$ and $T''_r \rightarrow T'_r$ holds.*
- *$< T_r, T''_r >$ is a support-relevant pair, i.e. $< T_r, T''_r > \in \text{SUP}(r)$.*

Note that on a reservoir resource, if T_r is a PRODUCE transition, then T'_r and all transitions in $MB(T_r, T'_r)$ are CONSUME transitions. Similarly, if T_r is a CONSUME transition, then T'_r and all transitions in $MB(T_r, T'_r)$ are PRODUCE transitions.

Let $Demand(MB(T_r, T'_r)$ be the maximum of the remaining demands of the transitions in $MB(T_r, T'_r)$, i.e.

$$Demand(MB(T_r, T'_r)) = \max_{T''_r \in MB(T_r, T'_r)} \text{RemDemand}(T''_r)$$

The set of transitions in $MB(T_r, T'_r)$ must be executed between T_r and T'_r. All transitions, including T_r, that can provide support to the transitions in $MB(T_r, T'_r)$ must provide at least $Demand(MB(T_r, T'_r))$ amount of support to the set. This means that the maximum amount of support that any transition, which executes before the set $MB(T_r, T'_r)$, including T_r, can provide to T'_r is

$$\text{capacity}(r) - Demand(MB(T_r, T'_r))$$

For each resource r we execute the following inference rule:

Inference 15. *For each support-relavant pair $< T_r, T'_r >$, such that $MB(T_r, T'_r)$ is non-empty, we can estimate the maximum amount of support that T_r can provide to T'_r directly as the following:*

$$\text{support}(T_r, T'_r) \leq \text{capacity}(r) - Demand(MB(T_r, T'_r)) \tag{4.63}$$

Figure 4.10 shows a support-relevant pair $< T1, T4 >$ on a resource that has capacity 6, where $ub(\text{support}(T1, T4)) = 3$. Transitions inside the dotted line represent $MB(T1, T4)$ that contains 2 transitions: $T2$ and $T3$, where $\text{RemDemand}(T2) = 3$ and $\text{RemDemand}(T3) =$

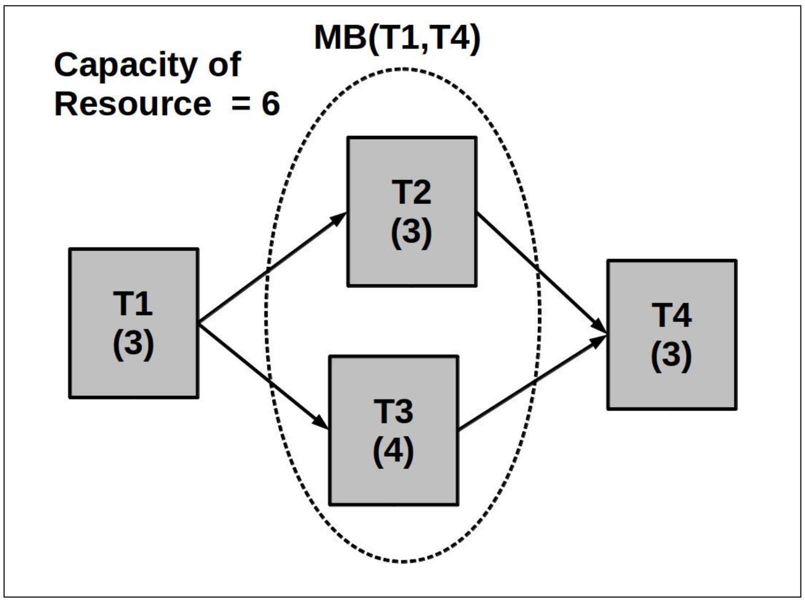


Figure 4.10: Example of Must be Between set

4. The arrows in the Figure 4.10 between the transitions represent the precedence relations between transitions. In this case $Demand(MB(T1, T4)) = 4$. Inference 15, updates the upper bound of $\mathrm{support}(T1, T4)$ as follows:

$$\begin{aligned}\mathrm{support}(T1, T4) &\leq Capacity - Demand(MB(T1, T4))\\ &\leq 6 - 4\\ &\leq 2\end{aligned}$$

This means that maximum amount of support that $T1$ can provide to $T4$ is 2.

4.6 Related Work

Our aim is to produce a flexible plan for a given planning problem by compiling the planning problem (bounded by the number of instances of actions) to a CSP, and extract a flexible plan form the solution to the CSP. Solving the compiled CSP, the transition-based constraint formulation of the planning problem, has two main aspects: how to branch on decision variables and how to propagate constraints and infer bounds on constraint variables. In this section we compare our work on solving the transition-based constraint formulation with other approaches in the planning and scheduling literature that solve planning and scheduling problems.

4.6.1 Related Branching Schemes

In the transition-based constraint formulation there are three main decision variables: achieve and follow variables for state variable transitions, and support variables for resource transitions. We first discuss the related work on branching on state variable transitions variables, and next we discuss the related work on branching on resource transitions variables.

4.6.1.1 Branching on State Variables

Our branching strategy for state variable transitions, i.e. branching on achieve and follow variables, can be seen as posting causal links as in POCL planning. This branching scheme is similar to the branching scheme used in the optimal temporal planner CPT [52]. In CPT the decision variables are the pre-conditions of actions, and it branches on possible support from actions that achieves the pre-condition. For each state variable transition we consider other state variable transitions instead of actions as possible supporters as it is in CPT. This is because we model each action's pre-condition and effects together as transitions on each state variable separately. The other difference with CPT is that by distinguishing between EFFECT and PREVAIL transitions, and having the requirement that all EFFECT transitions are sequenced on a state variable and that PREVAIL transitions are not allowed to overlap with EFFECT transitions, our branching strategy ensures that there will be no threats in the final plan. CPT adds extra constraints to eliminate threats from a plan.

4.6.1.2 Branching on Resources

In the constraint-based scheduling literature, a flexible schedule, also know as a partial-order schedule [42], is generally produced by a two step precedence constraint posting (PCP) [10] approach, which first finds a potential conflict on a resource and then posts precedence constraints between activities to resolve the resource conflict. If there is no conflict on any resource, then the partially ordered set of activities represents a valid partial-order schedule on each resource. There are two commonly used techniques to find resource conflicts: the clique-based method, and the profile-based method.

Clique-based method: This method was mainly developed for reusable resources, but it can be extended to reservoir resources as well. The main idea is to create a graph for each resource where nodes are the transitions on the resource, and two nodes are connected via a undirected edge if they overlap in time. Any clique in the graph, where the transitions in the clique together produce or consume more than the capacity of the resource, is called a *critical set*. Each critical set represents a resource conflict (also known as a **peak**). To resolve the resource conflict, we need to put enough precedence constraints between pairs of transitions (note that each precedence constraint makes the pair of transitions non-overlapping) in the peak such that the subset of overlapping transitions' total production or consumption lies between 0 and the capacity of the resource. The idea is to find a minimal clique called a *Minimal Critical Set* (MCS). The resource conflict that a MCS represents can be resolved by posting a single precedence constraint between a pair of transitions. For any given pair $< T_1, T_2 >$, there are two possible ways to resolve the conflict, either posting $T_1 \rightarrow T_2$ or $T_2 \rightarrow T_1$. These precedence relations are called the resolvers of the conflict.

Profile-based method: Profile-based methods [10, **?**, 42, 21] can be applied to both reservoir and reusable resources. In these method each transition is modeled with two possible resource events: a consume resource event at the start, and a produce resource event at the end. Requirements of consume resource events are represented as negative integers, and requirements of produce events are represented as positive integers. Let $v(e)$ denote the requirement of event e, and $t(e)$ denote the time when event e executes. Given a BORROW transition T_r, two resource events are created: a consume event c_{T_r} where $v(c_{T_r}) = -\mathtt{req}(T_r)$ and $t(c_{T_r}) = \mathtt{start}(T_r)$, and a produce event $p(T_r)$ where $v(p(T_r)) = \mathtt{req}(T_r)$ and $t(p_{T_r}) = \mathtt{end}(T_r)$, and these two events are constrained with the following temporal constraint.

$$t(c_{T_r}) + \mathtt{dur}(T_r) = t(p_{T_r})$$

For each CONSUME transition only the consume event is created, and each PRODUCE transition only the produce event is created. A precedence constraint between two resource events $e \rightarrow e'$ implies a temporal constraint $t(e') \geq t(e)$. In the rest of the discussion we assume that temporal constraints between resource events are consistent. Generally in the constraint-based scheduling literature consistency of the temporal constraints between resource events is maintained via a STN [18].

Given a resource event e on a resource r, the profile-based method calculates the resource envelopes before and after the event. A resource envelope is an interval that represents the minimum and maximum amount of resource available for use at a certain time point. Let $[L(e)^{<}_{min}, L(e)^{<}_{max}]$ be the resource envelope just before $t(e)$. In the following we describe the general idea for calculating a resource envelope given a resource event e. Let $B(e)$ be the set of resource events that must execute before e, and $U(e)$ be the set of resource events that are not ordered with e. Let $P(r)$ and $C(r)$ denote the set of all production events and consume events respectively. Note that all events in $B(e)$ must be executed before e. To estimate the maximum amount resource that could be available before e, i.e. $L(e)^{<}_{max}$, we assume that all produce events, that are not ordered after e, execute before e.

$$L(e)^{<}_{max} = \mathrm{init}(r) + \sum_{e' \in B(e)} v(e) + \sum_{e' \in P(r) \cap U(e)} v(e') \tag{4.64}$$

Similarly, to estimate the minimum amount of resource available before e, i.e. $L(e)^{<}_{min}$, we assume that all consume events, that are not ordered after e, execute before e.

$$L(e)^{<}_{min} = \text{init}(r) + \sum_{e' \in B(e)} v(e) + \sum_{e' \in C(r) \cap U(e)} v(e') \tag{4.65}$$

An event e is not safe if the following conditions does not hold:

$$0 \leq L(e)^{<}_{min} \leq L(e)^{<}_{max} \leq \text{capacity}(r)$$

This means that if $L(e)^{<}_{max} > \text{capacity}(r)$ or $L(e)^{<}_{min} < 0$, we say there is a resource conflict and e is not a safe event. Note that if $L(e)^{<}_{max} > \text{capacity}(r)$, then it means that some consumption events in $U(e)$ must be executed before e to make e safe. Similarly, if $L(e)^{<}_{min} < 0$, then some production events in $U(e)$ must be executed before e to make e safe. These possible precedence constraints are called the possible resolvers of the conflict.

Branching Scheme: A PCP-based search algorithm, that uses the two step conflict-resolving approach, can be seen as a meta-CSP approach. Each meta-CSP variable represents a resource conflict, and possible resolvers of the conflict are the possible values for the meta-CSP variable. At each step, the search selects a conflict and picks a resolver for the conflict. After selecting the resolver (which is a precedence constraint between two resource events) it propagates the temporal constraints implied by the resolver. If the temporal propagation results in a inconsistent state it chooses another resolver. If all resolvers fail to resolve the conflict, search backtracks. The problem is solved when there is no conflict left to solve.

Our Branching Scheme: As stated before, in this thesis we have taken a different resource reasoning approach to produce partial-order schedules on resources. Our approach is based on finding support for resource transitions, that are included in the plan, by posting *support links*. We branch by deciding how much support a resource transition provides to another resource transition. This idea can be seen as an adaptation of the idea of posting causal links between state variable transitions. At each branching point we decide that either a transition T provides the maximum support to T' or we lower the upper bound of the support between T and T'. As we have shown before, if all transitions that are included in the plan are fully supported, then the partial-order schedule on r created from the posted support links represents a set of valid schedules on r. Each valid schedule creates an evolution of r, where at each time point t, where $0 \leq t \leq H$, the following condition is true: $0 \leq \text{level}(r) \leq H$.

Our branching is related to an idea called *chaining* in the context of creating partial-order schedules [42]. Chaining is a procedure that has been used to lift a resource feasible fixed time schedule to a partial order schedule. The main idea behind chaining is to consider each multi-capacity resource r with $\text{capacity}(r) = m$ as m unit-capacity resources, and each resource transition T_r on r as a set of unit-requirement transitions $unit(T_r)$, where for each transition

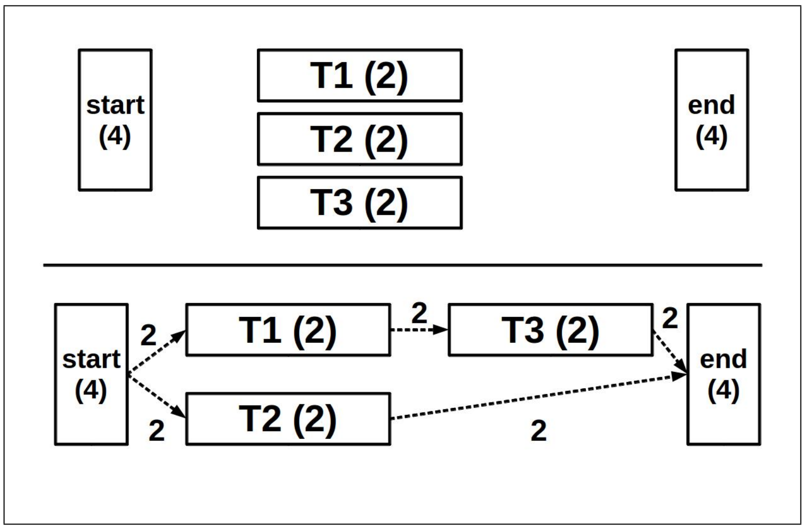


Figure 4.11: Example of Branching on Resources

$T'_r \in unit(T_r)$, $\text{req}(T') = 1$ and $|unit(T)| = req(T)$. For each transition T_r, all transitions in $unit(T_r)$ start at the same time and have as initial start time the start time from the fixed time solution. After converting the problem as above, it solves the converted problem by posting precedence constraints between the unit-requirement transitions on the unary resources.

One disadvantage of this way of branching is that we may have to make more search decisions than the conflict-resolving approach. However, the extra decisions are *easy* decisions, meaning these decisions will not lead to any dead end. This means that if we make enough decisions on support links between transitions on a resource, such that the implied precedence constraints guarantee that there is no peak and there is no possibility of a peak, then all other decisions that we need to make are the decisions that can only affect the quality of a solution, not the validity.

For example consider the scheduling problem described in Figure 4.11, where the problem is to schedule 3 BORROW transitions $T1$ to $T3$ on a reusable resource r with $\text{capacity}(r) = 4$. The top part of Figure 4.11 illustrates the initial situation, where *start* and *end* represents the dummy start and end transitions on r. We assume that each transition $T1$, $T2$ and $T3$ require 2 units of resource, and can overlap among each other. The bottom part of the Figure 4.11 shows a solution of the problem where all transitions' demand is fully supported by support links. To achieve this solution we have taken the following decisions (inclusion of support links):

$$D1 : \text{support}(start, T1) = 2$$
$$D2 : \text{support}(start, T2) = 2$$
$$D3 : \text{support}(T1, T3) = 2$$

Other support links, $\texttt{support}(T2, end) = 2$ and $\texttt{support}(T3, end) = 2$ are inferred using Inference 8 [2], because $T2$ and $T3$ are the only possible supporter of end. If we use the two step conflict-resolving approach, then we can see that the set $\{T1, T2, T3\}$ creates a conflict, because they can overlap and their total requirement is greater than the capacity of r. To resolve the conflict we need to post a single precedence constraint between these transitions. For example, if we post $T1 \rightarrow T2$, then there is no more conflict to resolve. Compared to this, our support-link based solution requires two additional decisions (decision D1 and D2) to produce a solution. However, note that the extra decisions are easy. Because once we have decided that $\texttt{support}(T1, T3) = 2$, there is no possibility to make the partial solution invalid.

The branching scheme that creates support links between resource transitions and creates causal links between state variable transitions provides a uniform decision making framework for planning related variables (selecting causal links) and scheduling (decision on support links). It also provides a way to develop inference techniques that not only consider the transitions that are included in the plan, but also consider the transitions whose inclusion status is not yet decided.

4.6.2 Related Propagation Techniques

There are mainly two classes of propagation and inference techniques in the constraint-based scheduling literature: techniques that consider absolute values of the temporal variables, and techniques that consider precedence relations between resource events. In this section we first relate our work with the first class of techniques and then we compare our work with the second class of propagation techniques.

4.6.2.1 Propagation Based on Temporal Values

Classical constraint-based scheduling algorithms use propagation methods like Time Tabling, Not-First/Not-Last, Edge-Finding etc, to infer bounds on temporal variables. These propagation methods are based on reasoning about the absolute temporal values of the transitions. The only inference rule that considers absolute temporal information to infer new precedence relations is Inference 1(see page 84). This means that classical propagation techniques can be used in conjunction with the propagation and inference techniques described in this chapter.

However, if the temporal variables have tight enough bounds, our inference rules can infer the same bounds as these classical propagation techniques. For example consider a cumulative scheduling scenario described by Vilim [53], where there are 4 transitions, A, B, C, and D that must execute on a reusable resource r, with $\texttt{capacity}(r) = 3$. Transition A has $\texttt{req}(A) = 3$ and $\texttt{dur}(A) = 1$, transition B has $\texttt{req}(B) = 1$ and $\texttt{dur}(B) = 3$, transition C has $\texttt{req}(C) = 2$

[2]Inference 8 is a special case of Inference 6, that deduces lower bound on the transition start time based on the PossSupp set, see page 92.

and $\mathrm{dur}(C) = 2$, and transition D has $\mathrm{req}(D) = 2$ and $\mathrm{dur}(D) = 3$. Transitions A and D can start their execution at time point 0, i.e. $lb(\mathrm{start}(A)) = lb(\mathrm{start}(D)) = 0$, and transitions B and C can start their execution only after time point 2, i.e. $lb(\mathrm{start}(B)) = lb(\mathrm{start}(C)) = 2$. Transition A, B, and C must finish their execution before time point 5, i.e. $ub(\mathrm{end}(A)) = ub(\mathrm{end}(B)) = ub(\mathrm{end}(C)) = 5$. Propagation 20 [3] updates the temporal variables to the following values:

$$ub(\mathrm{start}(A)) = 4 \text{ and } lb(\mathrm{end}(A)) = 1$$
$$ub(\mathrm{start}(B)) = 2 \text{ and } lb(\mathrm{end}(B)) = 5$$
$$ub(\mathrm{start}(C)) = 3 \text{ and } lb(\mathrm{end}(C)) = 4$$
$$ub(\mathrm{start}(D)) = H - 3 \text{ and } lb(\mathrm{end}(D) = 3$$

Using Inference 1 [4] we infer the following anti-precedence relations:

$$B \nrightarrow A \text{ , because } lb(\mathrm{end}(B))(5) > ub(\mathrm{start}(A))(4)$$
$$B \nrightarrow C \text{ , because } lb(\mathrm{end}(B))(5) > ub(\mathrm{start}(C))(3)$$
$$C \nrightarrow B \text{ , because } lb(\mathrm{end}(C))(4) > ub(\mathrm{start}(B))(2)$$
$$D \nrightarrow B \text{ , because } lb(\mathrm{end}(D))(3) > ub(\mathrm{start}(B))(2)$$

For the pair $< A, B >$ the total requirement is greater than the capacity of the resource, i.e. $< A, B >$ is a mutex pair. Similarly, $< A, C >$, $< A, D >$, $< C, D >$ are mutex pairs. From Inference 4 [5], we can deduce that $A \rightarrow B$, because $< A, B >$ is a mutex pair and $B \nrightarrow A$ holds. Since B is included in the plan, Propagation 15 [6]updates the latest end time of A as $ub(\mathrm{end}(A)) = 2$, and then Propagation 20 updates the latest start time of A as $ub(\mathrm{start}(A)) = 1$. Given $ub(\mathrm{start}(A)) = 1$, Inference 1 deduces the following anti-precedence relations:

$$C \nrightarrow A \text{ , because } lb(\mathrm{end}(C))(4) > ub(\mathrm{start}(A))(1)$$
$$D \nrightarrow A \text{ , because } lb(\mathrm{end}(D))(3) > ub(\mathrm{start}(A))(1)$$

Given these anti-precedence relations and the fact that both $< A, C >$ and $< A, D >$ are mutex pairs, Inference 4 deduces that $A \rightarrow C$ and $A \rightarrow D$. After posting $A \rightarrow D$, the earliest start time of D is 1 (via Propagation 15), and earliest end time of D is 4 (via Propagation 20). Given that $< D, C >$ is a mutex pair, after Inference 1 deduces that $D \nrightarrow C$ (because

[3]Propagation 20 implements the constraint $\mathrm{start}(T) + \mathrm{dur}(T) = \mathrm{end}(T)$, see page 81.
[4]Inference 1 infers anti-precedence constraints based on absolute values of temporal variables, see page 84.
[5]Inference 4 infers precedence constraints based on mutex relations, see page 86
[6]Propagation 15 implements the temporal constraint for an active precedence constraint, see page 80.

$lb(\mathrm{end}(D))(4) > ub(\mathrm{start}(C))(3))$, Inference 4 deduces that $C \rightarrow D$, which updates the earliest start time of D to 4 (via Propagation 15).

The edge-finding algorithm for cumulative scheduling described by Vilim [53], also deduces that the earliest start time of D is 4. In addition to deducing that D can only start from time point 4, we also deduce that A must execute before the set $\{B, C, D\}$. All these deductions are made even though we don't know if A or D are included in the plan or not. This means that even if A and D are optional, our propagation and inference techniques are able to deduce that A must finish execution before B, C, and D (if included in the plan), and D can only start at time point 4.

As we pointed out earlier, our propagation and inference techniques can perform deduction as described in the example above if temporal constraints (release dates and deadlines) on the transitions are tight enough. In general, however, there are situations where propagation techniques like Edge-Finding and Not-First/Not-Last find better bounds than our inference techniques.

4.6.2.2 Propagation Based on Precedence Relations

If temporal constraints are not tight enough (for example, most planning problems have a large horizon value), classical propagation techniques (mainly based on absolute temporal information) fail to deduce much. To remedy this type of situation there are other propagation techniques developed in the constraint-based scheduling literature that are based on the relative position of transitions. In this section we show that our propagation and inference techniques find better bounds than the precedence-based resource constraint propagation techniques developed by Laborie [36].

Laborie describes two main propagation techniques that infer bounds on temporal variables of transitions: the *Energy Precedence* constraint that works on reusable resources, and the *Balance* constraint that can be applied to both reusable and reservoir resources. As we have shown earlier, Inference 11 [7](see Proposition 3 on page 102) always produces bounds on the start and the end time of a transition that are as good as the bound produced by the Energy Precedence constraint or better. In the following we discuss the **applicability of the Balance constraint** with our branching scheme. This means that given our branching strategy we compare the deduction capability of the Balance constraint with our inference techniques.

Intermediate Search State: Figure 4.12 shows an intermediate search state on a reservoir resource r, where $\mathrm{capacity}(r) = 5$, and initially it was empty, i.e. $\mathrm{init}(r) = 0$. $start$ and $startConsume$ represent the dummy start transition T_r^{start} and the dummy start consume transition $T_r^{StartConsume}$. There are 4 PRODUCE transitions:$P1$ that produces 2 units of resource,

[7]Inference 11 updates the start time of a transition based on the optimal schedule for the USB set of the transition, see page 100.

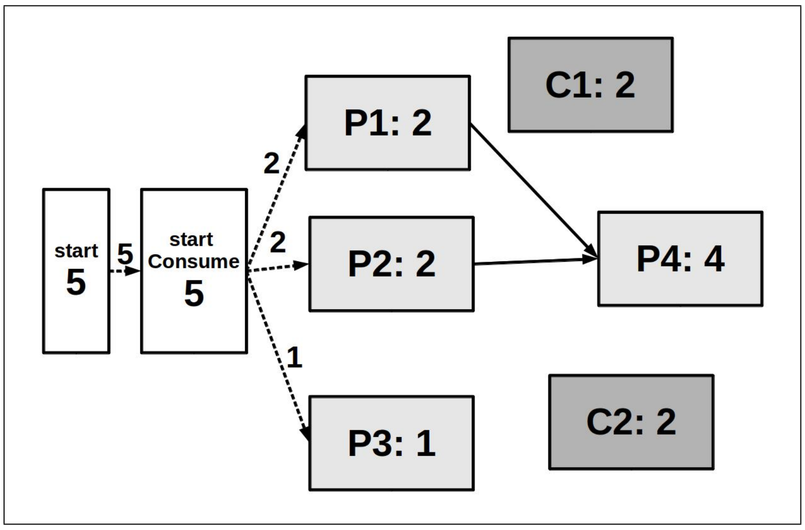


Figure 4.12: Partial search state on a reservoir resource r

$P2$ is same as $P1$, $P3$ produces 1 unit of resource, and $P4$ produces 4 units of resource. Similarly, $C1$ and $C2$ are two CONSUME transitions that consume 2 unit of resource each. All transitions have duration of 2 time units. All transitions can start at time point 0 and must finish before H. Since initial level was 0, we do not create the dummy start produce transition $T_r^{StartProduce}$. Initial propagation assigns $\text{support}(start, startConsume) = 5$ (Due to Constraint !21). In addition to this, the following decisions are made to reach the state described in the Figure 4.12:

$$\text{support}(startConsume, P1) = 2$$
$$\text{support}(startConsume, P2) = 2$$
$$\text{support}(startConsume, P3) = 1$$

Note that $< P1, P4 >$ and $< P2, P4 >$ are mutex pairs, and $FFS(P1) = 2$ and $FFS(P2) = 2$[8]. Given these information, Inference 5 [9] deduces $P1 \rightarrow P4$ and $P2 \rightarrow P4$. Since $P1$ and $P2$ are included in the plan, these precedence constraints update the earliest start time of $P4$ to 2. The solid edges between transitions in the Figure 4.12 represent the precedence relations, and the dotted edges between transitions represent the support links.

Given this intermediate search state, where the earliest start time of $P4$ is 2, we first show that the Balance constraint does not deduce any tighter bound on the start time of $P4$, while our inference techniques based on the concept of possible supporters (see page 89) deduce the

[8]Note FFS stands for Flow From Source, and for a transition T section 4.3.3 (page 86) describes how to calculate $FFS(T)$.

[9]Inference 5 infers precedence constraints based on the calculation of FFS, see page 88

lower bound on the start time of $P4$ to **4**.

The propagation technique of the balance constraint is based on the resource envelope calculation as described before. To infer the lower bound on the start time of $P4$ the Balance constraint first calculates the resource envelope just before $P4$, i.e. $[L(P4)^{<}_{min}, L(P4)^{<}_{max}]$, which can be calculated as described in equations 4.64 and 4.65. Note that in this case $B(P4)$ is the set of transitions that must finish execution before $P4$, i.e. $B(P4) = \{P1, P2\}$, and $U(P4)$ is the set of transitions that are unordered w.r.t. $P4$, i.e. $U(P4) = \{P3, C1, C2\}$. Also note that $P(r) = \{P1, P2, P3\}$ and $C(r) = \{\}$, because $P4$, $C1$ and $C2$ are not yet included in the plan. The calculations for the resource envelope as described earlier is based on resource events. We adopt it for transitions as follows:

$$L(P4)^{<}_{max} = \left(\sum_{T' \in P(r) \cap B(P4)} \mathrm{req}(T') - \sum_{T' \in C(r) \cap B(P4)} \mathrm{req}(T') \right) + \sum_{T' \in P(r) \cap U(P4)} \mathrm{req}(T') \tag{4.66}$$

$$= 4 - 0 + 0$$
$$= 4$$

$$L(P4)^{<}_{min} = \left(\sum_{T' \in P(r) \cap B(P4)} \mathrm{req}(T') - \sum_{T' \in C(r) \cap B(P4)} \mathrm{req}(T') \right) - \sum_{T' \in C(r) \cap U(P4)} \mathrm{req}(T') \tag{4.67}$$

$$= 4 - 0 - 0$$
$$= 4$$

The resource envelope before the transition $P4$ is [4,4]. The Balance constraint propagates time bound by analyzing the lower and upper bound of the envelope. If the lower bound is negative then it means that some PRODUCE transition form the set $U(P4)$ must be executed before $P4$, and if the upper bound is greater than $\mathrm{capacity}(r)$ then it means some CONSUME transition from the set $U(P4)$ must be executed before $P4$. Since in the example described in the Figure 4.12, the envelope is within 0 and $\mathrm{capacity}(r) = 5$, the Balance constraint does not deduce any tighter lower bound on the start time of $P4$.

The proposed inference techniques in this chapter for finding tighter bounds on temporal variables of a transition are divided into two types: based on the possible supporter sets (see Section 4.4.2 on page 94), and based on the set of transitions that must execute before the transition (see Section 4.4.1 on page 88). This means that in the above example our inference techniques not only consider what must happen before $P4$ but also the possible transitions that can execute before $P4$ to update its bound on the start time. In this particular example since

USB$(P4)$ (see Definition 19 on page 94) is empty, we can not deduce any tighter bound using Inference 11 [10] as described in Section 4.4.2. We can deduce a better bounds on start time for $P4$ by looking at the possible supporter sets for $P4$. Note that on a reservoir resource, only CONSUME transitions can support a PRODUCE transition, and only PRODUCE transitions can support a CONSUME transitions. In this example, the two CONSUME transitions, $C1$ and $C2$, have the same possible supporter set $\{P1, P2, P3, P4\}$. Inference 6 deduces that the earliest start time for both $C1$ and $C2$ is 2. Propagation 20 propagates the end times of both $C1$ and $C2$ to 4. Since $C1$ and $C2$ are the only possible supporters of $P4$, Inference 6 infers the earliest start time of $P4$ to be 4.

Propagation based on Resource Envelopes There are few fundamental differences between our resource inference techniques and envelope-based resource propagation techniques [26, 39]. First, our resource modeling is based on resource transitions which provides more information to the propagation techniques that infer temporal bounds and precedence relations based on different support types. In our model a PRODUCE transition can only support CONSUME transitions and vice versa. This modeling technique for resource transitions helps to infer better temporal and resource bounds. Consider the example given in Figure 4.12 and let us assume that no support decisions have been made and all transitions are included in the plan. Given this situation envelope-based propagation would detect that the given scenario is unsafe, and more ordering constraints are needed, but can not deduce any time bounds on the transition start times. Our support-based inference techniques will update the earliest start time of $C1$ and $C2$ to 2. This is because $C1$ and $C2$ can only be supported by the PRODUCE transitions.

Second, our branching choices provide more information than the usual PCP branching used in envelope-based propagation. Branching decisions on resources not only imply the precedence constraint, but also implies the fact is there can be no transition inbetween the involved transitions that will share the support amount. This extra information helps the propagation techniques to derive precedence constraints based on the support amount. Consider the above example in Figure 4.12 where it is decided that $P1 - P3$ are in the plan and they are supported by the dummy `start` transition, but $P4$, $C1$, and $C2$ are still undecided. Given these decisions we have showed that our inference techniques derive the earliest start time for $P4$ to 4. If we imagine all transitions are included in the plan, then given those three decisions envelope-based propagation would only find out the need for more precedence orders but can not find better temporal bounds.

The last difference is that envelope-based propagation techniques are based on a fixed set of resource events, while we consider transitions that are included in the plan and the transitions that are not yet excluded from the plan. For envelope-based propagation, we can deal with optional activities in two different ways. The first is to consider what is included in the plan

[10] Inference 11 updates the lower bound of the start time based on USB set of a transition, see page 100.

so far. Consider the example in Figure 4.12 again, and assume that we have not made any decisions except that all PRODUCE transitions are included in the plan. In this case envelope-propagation will find the state inconsistent, because the total production is larger than the capacity. The second way is to include all optional tasks and reason with them as if they are included in the plan. Consider an example with four PRODUCE transitions as in Figure 4.12, and suppose there were five CONSUME transitions with 2 units of resource requirements each. We know that all PRODUCE transitions are included in the plan, and no decisions have been made for the CONSUME transitions. If we assume that all these CONSUME transitions are also going to be part of the plan, then again envelope-based propagation will find the state inconsistent because total consumption is greater than the total production. Our propagation techniques handles optional activities naturally by considering how its resource requirement is going to be supported and what must happen before it.

This natural inclusion of optional activities of our representation is important for the problems we want to solve, which are in between planning and scheduling. In these problems we need to include actions (and transitions) in the plan in a step-by-step fashion. Our branching strategy keep tracks of how a transition provides support to other transition. Given this branching strategy, our inference techniques reason about when a transition's demand or precondition can be satisfied. During the reasoning it not only considers what must happen on a state variable or on a resource, it also considers what is still possible. This is why we believe our branching scheme and inference techniques are useful for solving the problems that are in between planning and scheduling.

4.7 Summary

In this chapter we have described how to solve the CSP that is generated from compilation process described in the previous chapter. We have proposed a branching scheme that branches on possible support to transitions. Each support implies a precedence constraint and inclusion of corresponding actions into the plan, which is then propagated using the propagation rules described in this chapter. We have described several inference technique that deduces temporal bounds and new precedence constraints.

The solving technique that we have described here is complete chronological backtracking search that finds a solution that satisfies all the constraints. For many practical problems finding a satisfying solution is not enough, but require optimizing some aspect of the solution. Commonly used optimization criteria are minimizing makespan, minimizing action costs etc. Although we have not discussed any particular optimization technique in this thesis, our search technique can be easily extended to support optimization criteria. One simple way to implement optimizing search method in our framework is to have bounds on the maximum end time of the dummy end action (minimizing makespan). We have showed that for solving project scheduling problem [3] this method produced competitive result.

Many real world planning problems are usually very large in terms of available actions, state variables and resources. In many cases these problems are solved using local search techniques (not complete). One such successful technique is the large neighbourhood search (LNS) technique [44]. In LNS solutions are generated by a repetitive destruction and construction process. Central to the LNS successs is the core constraint engine that checks constraint satisfaction of any given solutions. The propagation and inference techniques described in this chapter can be used for the construction step of an LNS search.

Chapter 5

Modeling Operational Constraints

In the previous chapters we have shown how to present a planning and scheduling problem using state variables, resources and actions (and transitions), and how to compile the problem into the transition-based constraint formulation, where each solution to the constraint model gives us a flexible solution plan. The transition-based constraint model ensures that each solution represents a set of valid schedules on state variables and resources under the physical constraints of these domain objects. By physical constraints we mean that on state variables all transitions, except for the PREVAIL transitions that require same state, are totally ordered, and on each resource r, at each time point $0 \leq t \leq H$, the condition $0 \leq \mathtt{level}(r,t) \leq \mathtt{capacity}(r)$ holds.

We are interested in solving problems that are in between planning and scheduling problems. These problems generally have complex temporal constraints like release dates, deadlines, time-windows, sequence dependent setup times etc. We will refer to these constraints collectively as *operational* constraints. In this chapter we will describe how we model these operational constraints in our representation, and the additional constraints that we need to add to handle these complexities in our constraint model. First we will describe how the setup times are represented. Next we will describe the operational constraints on individual states of state variables. Lastly we will show how time-window constraints are applied to state variables, resources, and actions.

5.1 Modeling Setup Times

On a domain object [1], if two transitions have to be executed consecutively, then it may be the case that the second transition can not start immediately after the first transition is finished, some delay is needed in between the end of first transition and the start of second transition. This time delay is called *setup* time or *changeover* time in the scheduling literature. Consider a coloring machine in a factory that can paint an object with one of the three different colors: red, green and blue, with the restriction that it can paint one object at a time. If there are two

[1] By domain object we mean both state variables and resources in the model

objects that need to be colored in two different colors, then before the machine can process the second object, it needs to be cleaned so there is no color contamination. The duration of the cleaning task is the Setup time between these two coloring tasks. If there are two objects that need to be painted the same color, for example blue, then the machine doesn't need any cleaning operation in between them, so the Setup time between the tasks would be 0. This is an example of *sequence dependent* setup time, because it depends on the order of the tasks. In many cases, a domain object can also have a *sequence independent* setup time, i.e a constant time delay, that must be applied between any pair of transitions on the domain object.

Although setup times are traditionally defined on reusable resources with capacity one, we generalize the concept of setup time for both state variable transitions and resource transitions. The advantage is that we can use the setup times on state variables to exclude inconsistent transition sequences by setting the time delay to infinity. In practice all setup times are triangular, but the model does not require them to be. We will describe the use of setup times on state variables later in the Case Study chapter.

5.1.1 Extending the Representation

In this section we describe the additional modeling elements for domain objects and transitions that are needed to model setup times between transitions. Each domain object has a **Setup Matrix** that has one or more **Setup States**. Each element in the setup matrix represents the time needed to change from one setup state to another. For a domain object d, $\mathsf{Setup}(d)$ denotes the setup matrix associated with d, and $States(\mathsf{Setup}(d))$ denotes the set of setup states of the setup matrix. Given a $\mathsf{Setup}(d)$ where $s_1, s_2 \in States(\mathsf{Setup}(d))$, notation $\mathsf{Setup}(s_1, s_2)$ denotes the value of the element $s_1 s_2$ in the matrix. Table 5.1 describes an example setup matrix that represents the time delay between different coloring actions. It has three colors as setup states: Red, $Blue$, and $Green$. Since each element in a setup matrix represents time

	Red	*Blue*	*Green*
Red	0	10	15
Blue	20	0	30
Green	30	20	0

Table 5.1: Setup Matrix for colors

delays, all elements are non-negative integer values. Note that setup times between same setup states are set to 0.

For each transition T in our model we assign two setup states: $\mathsf{FromSetupState}(T)$ and $\mathsf{ToSetupState}(T)$. The $\mathsf{FromSetupState}(T)$ denotes the setup state needed when T starts execution, and the $\mathsf{ToSetupState}(T)$ denotes the setup state that results when T finishes its execution on $\text{obj}(T)$. Note that for each transition T,

$$\text{FromSetupState}(T) \in \mathit{States}(\text{Setup}(\text{obj}(T)))$$

$$\text{ToSetupState}(T) \in \mathit{States}(\text{Setup}(\text{obj}(T)))$$

Let $\text{Setup}(T, T')$ represent the time delay that is needed if T' executes right after T. Recall that a pair of transitions $< T, T' >$ can execute consecutively, if it is a *support-relevant* pair or a *achieve-relevant* pair or a *can-follow* pair [2]. The value of $\text{Setup}(T, T')$ is defined as follows:

$$\begin{aligned}\text{Setup}(T, T') &= 0 \text{ if } < T, T' > \text{ is neither a support- nor achieve-relevant nor can-follow pair}\\ &= \text{Setup}(\text{ToSetupState}(T), \text{FromSetupState}(T')) \text{ otherwise}\end{aligned}$$

In our representation we define a setup time matrix for each domain object. If setup times are not defined for a domain object, then its setup matrix has only one default setup state, and all transitions that need to execute on the domain object have same setup state. This means that for each pair of transitions $< T, T' >$ on the domain object $\text{Setup}(T, T') = 0$.

Note that our treatment of setup states for each transition is different from traditional use of the setup matrix. Traditionally, each activity has only one setup state. For example, for the coloring machine mentioned above, each coloring requirement would have only one setup state, the required color. This can be modeled in our representation by providing the same setup state for both $\text{FromSetupState}(T)$ and $\text{ToSetupState}(T)$. Having two setup state become useful to exclude inconsistent sequence of transitions on a domain object. We will discuss such examples in the next chapter.

5.1.2 Extending the Constraint Model

Since setup times are defined between two consecutive transitions executing on a domain object, we introduce the following 3 constraints into our constraint model:

Constraint 23. *For a resource r, for all support-relevant pairs $< T_r, T'_r > \in \text{SUP}(r)$ the following constraint holds*

$$\text{support}(T_r, T'_r)) > 0 \Rightarrow \text{start}(T'_r) \geq \text{end}(T_r) + \text{Setup}(T_r, T'_r) \tag{5.1}$$

Constraint 24. *For a state variable sv, for all achieve-relevant pairs $< T_{sv}, T'_{sv} > \in \text{AC}(sv)$ the following constraint holds*

$$\text{achieve}(T_{sv}, T'_{sv})) = 1 \Rightarrow \text{start}(T'_{sv}) \geq \text{end}(T_{sv}) + \text{Setup}(T_{sv}, T'_{sv}) \tag{5.2}$$

[2]Definitions of these pairs are given in Section 3.4.2 on page 51.

Constraint 25. *For a state variable sv, for all can-follow pairs $< T^p_{sv}, T'_{sv} > \in \mathrm{FL}(sv)$ the following constraint holds*

$$\mathrm{follow}(T^p_{sv}, T'_{sv})) = 1 \Rightarrow \mathrm{start}(T'_{sv}) \geq \mathrm{end}(T^p_{sv}) + \mathrm{Setup}(T^p_{sv}, T'_{sv}) \tag{5.3}$$

5.1.3 Extending Inference Rules

Using the setup time information we can infer bounds on the start and end times of transitions. Here we only show how we use setup information to get tighter lower bound on the start times. Similar technique can be used to infere tighter upper bound on the end times.

Let $minSetup(T)$ represent the minimum amount of time delay that is needed before T executes on the domain object $\mathrm{obj}(T)$. $minSetup(T)$ can be calculated as the following:

- If T is a resource transition, then

$$minSetup(T) = \min_{T' \in \mathrm{PossSupp}(T)} \mathrm{Setup}(T', T) \tag{5.4}$$

  Where $\mathrm{PossSupp}(T)$ is the possible supporter set [3] of T.

- If T is a state variable transition, then

$$minSetup(T) = \min_{T' \in \mathrm{PossAchiev}(T)} \mathrm{Setup}(T', T) \tag{5.5}$$

  where $\mathrm{PossAchiev}(T)$ is the possible achievers set [4] of T.

Given $minSetup(T)$, we can apply the following inference rules for resource transitions and state variable transitions respectively. Since $minSetup(T) \geq 0$, these new inference rules provides better lower bounds on the start times than the inference rule 6 and 9 [5].

Inference 16. *For each resource transition T_r, such that T_r is not excluded from the plan,*

$$\mathrm{start}(T_r) \geq get\_\mathrm{eft}\left(\mathrm{PossSupp}(T_r)\right) + minSetup(T_r) \tag{5.6}$$

Inference 17. *For each state variable transition T_{sv}, such that T_r is not excluded from the plan:*

$$\mathrm{start}(T_{sv}) \geq \min_{T'_{sv} \in \mathrm{PossAchiev}(T_{sv})} \mathrm{eft}(T'_{sv}) + \mathrm{Setup}(T', T) \tag{5.7}$$

[3] Possible supporter set of a transition is described in Section 4.4.1.1 (page 89)
[4] Possible Achiever set of a transition is described in Section 4.4.1.2 (page 93)
[5] These inferece rules are defined in the Section 4.4.1 (page 88)

5.2 State Variable State-Constraints

There are some temporal features or constraints that are quite common in many realistic problems, such as time windows, deadlines, release date etc. For example, in a pickup and delivery problem packages can have time windows during which they can be picked up and can be delivered. In many scheduling models these constraints are directly modeled by posting temporal constraint on related actions' start and end times.

In our representation, we model these temporal features as constraints on states of state variables. For each state variable sv, we define 4 constraints for each state $s \in \mathtt{dom}(sv)$:

- $achieve\_after(s, sv, t)$: This constraint represents that within the horizon $[0, H]$, state variable sv is allowed to change to the state s from another state only after time point t. A schedule on sv is valid if there does not exist a pair of time points $< t', t'' >$, where $t' < t'' < t$, such that the following constraint holds:

$$\mathtt{state}(sv, t') \neq s \wedge \mathtt{state}(sv, t'') = s \tag{5.8}$$

  This means that each EFFECT transition T^E_{sv} ($\neq T^{start}_{sv}$), that can change the state of sv to s, i.e. $\mathtt{post}(T^E_{sv}) = s$, must finish its execution after t.

- $achieve\_before(s, sv, t)$: This constraint represents that within the horizon $[0, H]$, state variable sv can change to the state s from another state s' only before time point t. In each valid schedule of sv there does not exist a pair of time points $< t', t'' >$, where $t < t' < t''$, such that the following constraint holds:

$$\mathtt{state}(sv, t') \neq s \wedge \mathtt{state}(sv, t'') = s \tag{5.9}$$

  This means that each EFFECT transition T^E_{sv} that can change the state of sv to s, i.e. $\mathtt{post}(T^E_{sv}) = s$, must can finish its execution before t.

- $change\_after(s, sv, t)$: This constraint represents that within the horizon $[0, H]$, state variable sv is allowed to change to another state s' from state s only after time point t. A schedule on sv is valid only if there does not exist a pair of time points $< t', t'' >$, where $t' < t'' < t$, such that the following constraint holds:

$$\mathtt{state}(sv, t') = s \wedge \mathtt{state}(sv, t'') \neq s \tag{5.10}$$

  This means that each EFFECT transition T^E_{sv} ($\neq T^{end}_{sv}$), that can change state of sv from s to another state, i.e. $\mathtt{pre}(T^E_{sv}) = s$, must start its execution after t.

- $change\_before(s, sv, t)$: This constraint represents that within the horizon $[0, H]$, state variable sv can change to a state s' from state s only before time point t. In each valid schedule of sv there does not exist a pair of time points $< t', t'' >$, where $t < t' < t''$, such that the following constraint holds:

$$\mathtt{state}(sv, t') = s \wedge \mathtt{state}(sv, t'') \neq s \tag{5.11}$$

This means that there each EFFECT transition T^E_{sv}, that can change state of sv from s to another state, i.e. $\mathtt{pre}(T^E_{sv}) = s$, must start its execution before t.

Effectively these constraints allows us to model time windows on each possible state of a state variable. For example, loc_pkg denote a state variable that models the current location of a package, that has $\mathtt{dom}(loc\_pkg) = \{pick, goal\}$, where $pick$ represents the pickup location and $goal$ represent the delivery location of the package. Let $[t^{min}_p, t^{max}_p]$ be the time window within which the package can be picked up at the pickup location, and $[t^{min}_g, t^{max}_g]$ be the time window when it must be delivered to the goal location. To model these time windows we define the following constraints on the states of the state variable loc_pkg.

- $change\_after(pick, t^{min}_p)$ and $change\_before(pick, t^{max}_p)$ to model the time windows on the pickup location.
- $achieve\_after(goal, t^{min}_g)$ and $achieve\_before(goal, t^{max}_g)$ to model the time window on the goal location.

The above state-constraints let us model absolute temporal constraints, i.e. time-windows on a state. Although time-windows on a state is a common operational constraint, there are other constraints that constrain the occurrence of the state in a relative manner. For example consider a state variable that represents the work cycle of a robot as described in Figure 5.2 It has four states, Idle, Working, Cooling, and Shutdown, where Idle is the initial state and Shutdown is the goal state. Constraints on the states are the following:

- When the robot is in the *Working* state, it must stay in that state for at least 5 min and at most 30 min.
- The maximum number of times that a robot can be cooled is 6.
- Each cooling cycle takes at least 10 min, and robot is not allowed to stay in the *Cooling* state more than 15 min.

These constraints are relative temporal constraints, meaning they don't specify any particular time points, but constrain the state variables' evolution based on time points within the horizon. To model this type of restrictions we define the following two state-constraints.

- $achieve(s, sv, min, max)$: This constraint represents that for the state variable sv, the state s must be achieved at least min number of times and at most max number of times.

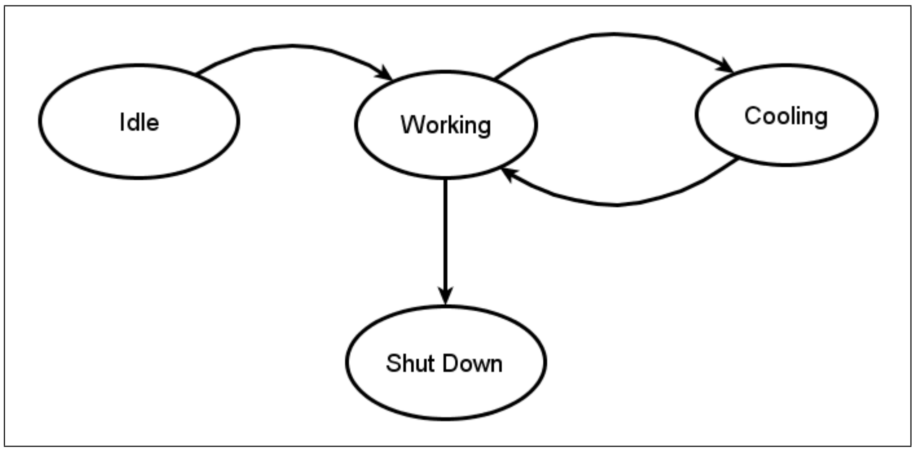


Figure 5.1: State Variable: Robot Work Cycle

We say a time point t is a s-achieving time point, if $\text{state}(sv, t) = s$, and either $t = 0$ or $\text{state}(sv, t-1) \neq s$. This constraint implies that the number of such s-achieving time points within the horizon [0,H] is bounded by the interval [min, max].

- $persist(s, sv, min, max)$: This constraint represents that for the state variable sv, the state s when achieved, must not be changed to other state until min amount of time has elapsed, and must be changed to other state after max amount time has elapsed. This means that for each pair of time points $< t, t' >$ within the horizon [0,H] where the state of sv is s, the duration $(t' - t)$ is bounded by the interval [min, max].

We can satisfy the additional requirements on our example *Robot Work Cycle* state variable, as described above, by posting the following three state-constraints: two persistent constraints on states *Working* and *Cooling*, and a achievement constraint on the state *Cooling*.

To include these constraints on states of state variables, we extend the constraint model as described below.

5.2.1 Extending the Constraint Model

As defined above, each constraint on a state s of a state variable sv effectively restricts when an EFFECT transition T^E_{sv}, related to s by pre- or post-condition, can start or finish. For each state s of a state variable sv, let $achieve(s, sv)$ denote a set of EFFECT transitions on sv, such that $\forall T \in achieve(s, sv)$, $\text{post}(T) = s$. Similarly let $change(s, sv)$ denote a set of EFFECT transitions on sv, where $\forall T \in change(s, sv)$, $\text{pre}(T) = s$.

The following temporal constraints represents the implementation of the time-window constraints on an individual state of a state variable.

Constraint 26. *For each state variable sv, for each state $s \in \mathrm{dom}(sv)$, the following temporal constraints must be satisfied.*

$$\begin{aligned}
\textit{achieve\_after}(s, sv, t) \textit{ is defined } &\Rightarrow \forall T \in \textit{achieve}(s, sv) : \\
&\quad \mathrm{end}(T) \geq t && (5.12) \\
\textit{achieve\_before}(s, sv, t) \textit{ is defined } &\Rightarrow \forall T \in \textit{achieve}(s, sv) : \\
&\quad \mathrm{end}(T) \leq t && (5.13) \\
\textit{change\_after}(s, sv, t) \textit{ is defined } &\Rightarrow \forall T \in \textit{change}(s, sv) : \\
&\quad \mathrm{start}(T) \geq t && (5.14) \\
\textit{change\_before}(s, sv, t) \textit{ is defined } &\Rightarrow \forall T \in \textit{change}(s, sv) : \\
&\quad \mathrm{start}(T) \leq t && (5.15)
\end{aligned}$$

The constraint $achieve(s, sv, min, max)$ implies a counter that counts the number of transitions that achieve the state s, and min and max represents the minimum and maximum number of such transitions. For each state variable sv, and for each state s, the following constraint implements the counter constraint for $achieve(s, sv, min, max)$

Constraint 27. *For each state variable sv, and for each state s, if $achieve(s, sv, min, max)$ is defined, then the number of s-achieving transitions included in the plan must be bounded by min and max.*

$$min \leq \sum_{\forall T \in achieve(s,sv)} \mathrm{inplan}(T) \leq max \tag{5.16}$$

The persistent constraint $persist(s, sv, min, max)$ defines a min and a max time delay between a achieve-event and a following change-event on the state s. Each $\mathrm{achieve}(T_{sv}, T'_{sv})$ variable, where $T_{sv} \in achieve(s, v)$ and $T'_{sv} \in change(s, sv)$, represents such an achieve-change event pair. In our constraint model each persistent constraint is modeled as the following temporal constraints.

Constraint 28. *For a state variable sv where $persist(s, sv, min, max)$ is defined, if an achieve variable $\mathrm{achieve}(T_{sv}, T'_{sv})$, where $T_{sv} \in achieve(s, v)$ and $T'_{sv} \in change(s, sv)$ is selected , then the difference between $\mathrm{start}(T'_{sv})$ and $\mathrm{end}(T_{sv})$ must be bounded by min and max*

$$\mathrm{achieve}(T_{sv}, T'_{sv}) = 1 \Rightarrow min \leq \mathrm{start}(T'_{sv}) - \mathrm{end}(T_{sv}) \leq max \tag{5.17}$$

5.3 Time-Windows on State Variables, Resources, Actions

In the above section we have described how to model a time-window constraint associated with a particular state of a state variable. In this section we introduce time-window constraints on the evolution of state variables and resources, and on action executions. For example consider the state variable representing status of a working robot above. If the robot is only available during a time-window then we can model that feature by specifying a time-window on the state variable. Similarly a time-window on a resource represents the temporal availability of the resource. Time-windows on actions are the most commonly occurring constraint in many real problems. For example, in a satellite domain taking a picture of a target is constrained by the time-windows describing the visibility of the observation for the satellite.

5.3.1 Extending Representation

For a domain object d we define a time-window as $time - window(d, t_{start}, t_{end})$. This constraint represents that the evolution of the domain object d is restricted within the time starting at t_{start} and ending at t_{end}. This means that the domain object d will have the same state or resource level as its initial configuration until t_{start}, and the state or resource level at the time point t_{end} will continue to persist until the end of the planning horizon H. Given a time-window the conditions on the evolution of a state variable sv are the following:

$$\forall t \in [0, t_{start}) : \mathrm{state}(sv, t) = \mathrm{init}(sv) \tag{5.18}$$

$$\forall t' \in (t_{end}, H] : \mathrm{state}(sv, t') = \mathrm{state}(sv, t_{end}) \tag{5.19}$$

Similar constraints hold for the evolution of a resource r

$$\forall t \in [0, t_{start}) : \mathrm{level}(r, t) = \mathrm{init}(r) \tag{5.20}$$

$$\forall t' \in (t_{end}, H] : \mathrm{level}(r, t') = \mathrm{level}(r, t_{end}) \tag{5.21}$$

Similar to the domain object, $time - window(a, t_{start}, t_{end})$ denotes a time-window constraint for an action a. The meaning of a time-window on an action is simple: each transition of the action must not start before t_{start} and must not end after t_{end}.

5.3.2 Extending Constraint Model

Since only transitions are responsible for changing states of state variables and levels of resources, these temporal constraints limit the execution of the transitions within the time-window. We implement the following constraint on each transition T, where T is not the dummy start or dummy end transition on the domain object d:

Constraint 29. *For each domain object d, given a time-window constraint (t_{start}, t_{end}), each transition T on d (e.i* $\mathrm{obj}(T) = d$*), where T is not the dummy start or the dummy end transition on d, must starts after t_{start} and must ends before t_{end}.*

$$\mathrm{start}(T) \geq t_{start} \textit{ and } \mathrm{end}(T) \leq t_{end} \tag{5.22}$$

For an action a, a time-window constraint simply translates to temporal constraints on the start and end variables of its transitions as mentioned before.

Constraint 30. *For each action a, given a time-window constraint (t_{start}, t_{end}), each transition T of a, can not start before t_{start} and can not end after t_{end}. This means for each T, where* $\mathrm{act}(T) = a$*, the following temporal constraints hold*

$$t_{start} \leq \mathrm{start}(T) \tag{5.23}$$

$$\mathrm{end}(T) \leq t_{end} \tag{5.24}$$

5.4 Other constraints

There are other constraints that appear in practical planning and scheduling problems frequently other than time-window and setup time constraints, such as: disjunctive time windows on goal, inclusion of action based on inclusion/exclusion of other actions etc. In this section we briefly describe how these constraints can be modeled in transition-based representation.

5.4.1 Disjunctive goal constraint

The disjunctive goal constraint describe the fact that a goal can be achieved only in one of the possible time windows. In our representation achieving goal is being on the goal state. This constraint can be modeled using state variable state time window constraints as described above. For each possible time windows, we create a copy of the goal state with the corresponding time-window constraint, and create extra copies of the actions that either use the state, achieve the state or change the state.

5.4.2 Action exclusion and implication constraints

In many practical planning application it is necessary to express action exclusion and implication constraints. An action exclusion constraint $Exclusive(a, b)$ means that at most of one of actions a and b can be in the plan. This can be modeled with a unary reservoir resource consumed by both actions. An action implication constraint $Imply(a, b)$ means that if action a is included in the plan then action b must be included. We can model action implication constraint between a pair of actions by creating a special state variable for each implication.

Figure 5.4.2 describes the state variable for the action implication constraint between action a

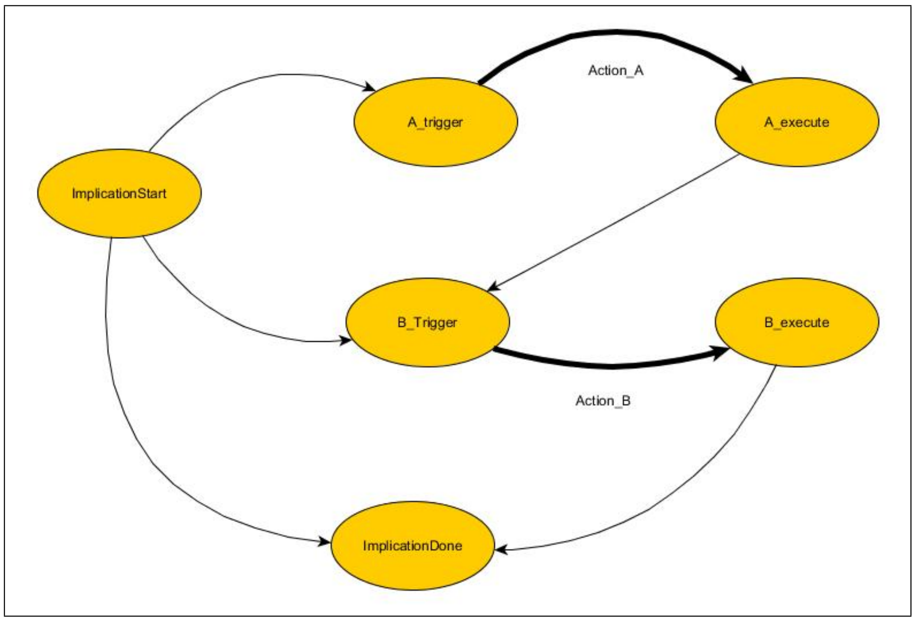


Figure 5.2: Action implication state variable

and b. The bold edges represent the transitions for real action a and b. All other edges represent transitions of dummy actions the we need to create for this special state variable. The "ImplicationStart" state represent the initial state and the "ImplicationDone" state represents the goal state for the state variable. If action a becomes active (that is included in the plan) then the path between the initial and goal state of the state variable must contain the transition of action b. Note that in normal state variables of a transition supports another transition then it implies a precedence constraint between the transitions. If the precedence constraints holds in this state variable then it would mean that $Imply(a, b)$ can only hold if $a \rightarrow b$ holds. To avoid that, for each state variable that represents an implication constraint, the precedence constraints between transitions will be ignored.

5.5 Summary

In this chapter we have described how to model sequence dependent setup times, time-windows (which generalizes the release date and deadline constraints) on domain objects and actions, and constraints (counter and time-windows) on individual states of state variables. These operational constraints are very common in problems that lie between planning and scheduling problems. As we have shown above, describing these constraints in our problem description framework is straightforward and intuitive. Compiling these constraints from the model to our transition-based constraint model is also simple, since these constraints translate to a set of simple temporal and counting constraints. We will describe modeling of a problem that includes many of these operational constraints in the next chapter.

Chapter 6

Case Study

In this chapter we will describe how to model a realistic planning and scheduling problem in our transition-based framework as described in the previous chapters. In this thesis we have proposed a framework for modeling and solving problems that are in between planning and scheduling. There are many realistic problems across different industries that fall into this category. Examples of such problems can be found in space applications, factory production, and supply chain problems in different industries.

In this chapter we choose one such problem and show how to model this problem in our framework. The purpose of this chapter is to show how realistic problems can be intuitively modeled easily in the framework, which then is compiled to a standard CSP.

6.1 Description of the complex satellite domain

Fleets of Earth-observing satellites are used for a variety of purposes: to observe weather, track movements on land and at sea, monitor climate change and volcanic eruptions, and many more. These satellites make observations and send information to the earth via ground stations. For a satellite to perform an observation a sequence of tasks need to be performed: first the satellite needs to turn one of its instrument towards the observation target, then make the observation, and then download the observed data to a ground station located on Earth. There are predefined time windows when a satellite can make an observation or download the data to a ground station based on the flight path of the satellite. Limited data storage capacity and power onboard each satellite restrict its capacity to make observations.

A simplified version of this type of space-related application problem has been used to create the well known 'Satellite' planning domain, used in planning competitions. The problem that we would like to model in our formalism is a more complex version (though, still simplified from the real problem) of the satellite domain as described by David Smith via personal communication. We have chosen the satellite problem because of its familiarity in the AI planning and scheduling community, and have wide range of interesting constraints. We will show that our representation can express most of the complexities in the satellite problem.

The main task of the set of Earth observing satellites is to take pictures of a given set of targets. In general, to take a picture of a target, a satellite performs a sequence of actions: first it turns one of its instruments that is suitable for taking the picture to the target location, then it takes the picture and stores it in its on-board memory. Each picture taken by a satellite needs to be downlinked from on-board memory to a ground station. Similar to the picture-taking action sequence a satellite performs two steps to downlink the stored data: first it turns one of its antennas to the ground station's location, and then it downlinks the data.

The simple task of taking a picture by a satellite becomes complex when we consider the following factors. During the picture-taking operation, the target must be visible to the satellite instrument. For each satellite, each target is only visible within a set of time windows (when satellite is flying over or near the target). Only a select set of instruments on a satellite can be used for a given target, and depending on which instrument is used it may take a different amount of time to turn the instrument to the target, as each instrument on a satellite has its own turning rate. In general, a satellite can use more than one instrument to take pictures of different targets at the same time (provided they are all visible to the satellite at the same time), but some instruments can't be used simultaneously due to their design.

Similarly, the downlinking actions are complicated by the following facts. For each satellite, each ground station is only visible within a set of time windows. Each ground station operates in a set of predefined frequency bands, and each antenna on a satellite has its own frequency. This means that not every antenna can be used to download data to every ground station. To downlink data, the ground station must be visible and the frequency of the antenna must be compatible with the ground station's frequencies. Also, each antenna on a satellite has its own data transfer rate which defines the total duration needed to downlink a picture using that antenna. Before a downlink operation can begin, the antenna for the operation must turn to the ground station. Similar to the instruments, the duration of a turning action depends on the turning rate of the antenna.

In addition to the constraints above, there are other operational constraints that must be taken into account by the satellites while performing picture taking or downlink actions. We describe some of these constraints in the following. Both picture-taking and downlinking actions must be preceded by a turning action of either an instrument or antenna. These turning actions produce vibration, and because of that, picture-taking and turning actions can not be performed concurrently. The on-board memory of each satellite is a finite capacity solid state recorder (SSR). One of the main constraints of the SSR is that data can only be read or written on it at any point in time. This means that picture-taking and downlinking actions can't be performed simultaneously. Each action in this problem domain: turning, downlinking, or picture-taking consumes power. Each satellite has a finite capacity power source that can be only be recharged during predefined time windows when the satellite is in the view of the sun.

The satellite problem described above is in the class of problems that are in between planning and scheduling. Here, planning decision are which instruments to use for observations, and in which order, while this ordering must satisfy the resource and temporal constraints. In this chapter we will model this problem in the transition-based formulation.

6.2 Model: Satellite Domain

In this section we will describe how to model the satellite domain described in the previous section. First we will give an overview of the actions that need to be modeled. Then we will describe the state variables and the resources in the model, and last we will present the detailed model of the actions and their transitions.

In this case study we represent each user request as an observation. Each observation is a request for a particular type of picture of a target. Each type of picture can be taken by a set of instruments. Each observation in our model is associated with a target and a set of possible instruments. Only one such instrument will be used to take the picture of the target.

Each target and ground station has a physical location on the earth.

6.2.1 Overview of Actions

We will model the following actions:

1. **TurnInstrument(Inst, TargetFrom, TargetTo)**: This action turns an instrument towards the *TargetTo* location from the *TargetFrom* location. The duration of this action is the slewing time between TargetFrom and TargetTo. During a TurnInstrument action the instrument can't take any other picture or be involved in any other turning action. Since any turning action generates vibration, no other instruments on the same satellite can take any picture while this action is executing. After this action is finished executing, the instrument is pointing to the TargetTo location. By pointing we mean that the instrument is configured to point to the physical location of the *TargetTo* when it is visible to the satellite.

2. **TakePic(Inst, Target)**: This action takes a picture of the target using the instrument. The duration of this action depends on the target and the instrument. The instrument must be pointing to the target before the picture can be taken. While executing, this action stores the picture of the target in the SSR of the corresponding satellite. Note that although the turning of the instrument towards the target can happen any time, a picture-taking action can only be executed during one of the time intervals where the target is visible to the satellite.

3. **TurnAntenna(Antenna, GSFrom, GSTo)**: Similar to the TurnInstrument action, this

action turns an antenna from GSFrom to GSTo [1]. The duration of this action depends on the slewing angle between the *GSFrom* and *GSTo*. After executing this action, the antenna points to the location of *GSTo*. Similar to the TurnInstrument actions, by pointing we mean that the antenna is configured to point to the physical location of the *GSTo* when it is visible to the satellite.

4. **Downlink(Antenna, GS, Observation)**: The Downlink action downloads the observation data stored in the satellite's SSR to the ground station using the antenna. The duration of this action depends on the size of the picture and the transfer rate of the antenna. Before downlinking can begin, the antenna must be pointing towards the ground station. Similar to the TakePic action, this action can only be executed when the ground station is visible to the satellite.

5. **SwitchON(Inst)**: Each instruments can only be switched on for limited amount of time. After that it must to be switched off before it can be switched on again. This action represent switching on an instrument. We will assume that the process of switching on is not instantaneous, it takes some time before the instrument can be used to take a picture.

6. **SwitchOFF(Inst)**: Similar to the SwitchON action, this action represents the process of switching off the instrument. We will assume that it takes some time to completely switch off an instrument.

7. **Recharge(Satellite, SunTimeWindow)**: Each Satellite has a battery. Each action described above consumes a fixed amount of power from this battery. The only way to recharge this battery is via solar power. The sun is only visible to the satellite during certain time windows. We will call such intervals *SunTimeWindows*. We will assume that recharge rates are same for each Satellite-SunTimeWindow pair. Each recharge-rate is represented as p/t, where p is the amount of power and t is the unit of time.

   We create recharge actions for such time intervals, where each action recharges the battery with p amount of energy and takes t amount of time. The number of recharge actions for each Satellite-SunTimeWindow pair depends on the length of the SunTimeWindow interval. If length of the SunTimeWindow is l, then we will create (l/t) number of recharge actions.

   All these actions are constrained to be executed within the time window. For example, let's assume that from 9am to 9:30am the Sun will be visible to a satellite. If the rate of recharging is 2 units per minute, then we will create 30 recharge actions, where each recharge action produces 2 units of power and executes for a minute. Each of these

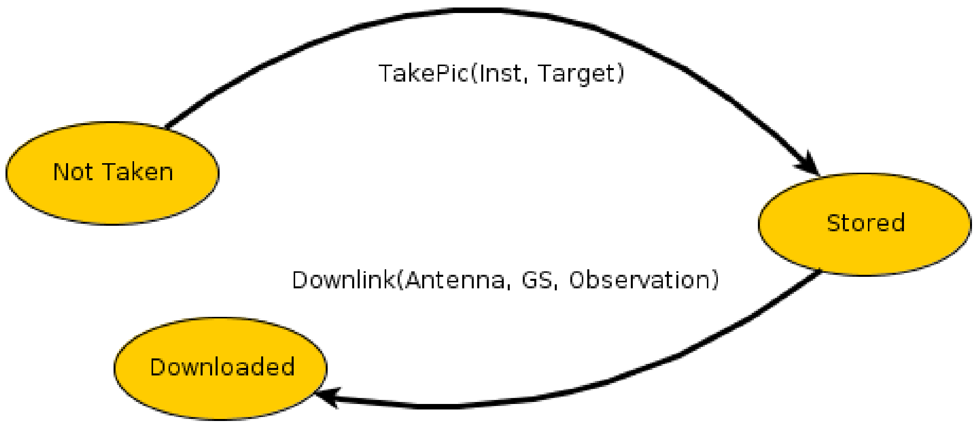
[1] GS stands for ground station.

recharge actions' earliest start time is 9am and latest finish time is 9:30am. Note that none of these recharge actions for a Satellite-SunTimeWindow pair can overlap with each other, because each recharge action represents the rate of recharging.

6.2.2 State Variables

We will model the following state variables:

1. **Observation_Status**(O_i): For each observation O_i, we create a this state variable that represents the different state that the observation can be in. Each observation state variable has 3 different states: *Not_taken*, *Stored*, and *Downloaded*. The state *Not_taken* represents the initial state of the observation where no decision has been made on which instrument will be used. The TakePic action changes the *Not_taken* state to the *Stored* state. When an observation is in the *Stored* state, this means that a picture for the observation has been taken and it is stored in the SSR of the satellite. The following figure shows the state/transition graph of this state variable.

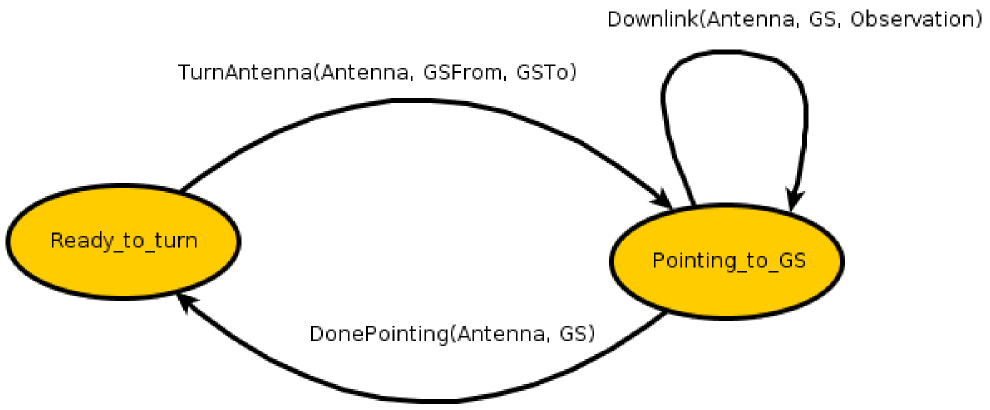


A Downlink action then changes the *Stored* state to the *Downloaded* state by downloading the picture to a ground station. For each observation, the *Downloaded* state is the final or goal state. This means that this state variable is a **goal** state variable.

Alternative Modeling As an Oversubscribed Problem:
In our problem we represent each *Observation_Status* as a goal state variable. A solution to this problem will successfully achieve all observations. We will say an observation is achieved, when it is downloaded to a ground station. In reality, for a given planning horizon, there are more observation requests than the optimal number of observations that can be achieved within the planning horizon. This means that a planner should choose if it wants to achieve an observation or not. This type of problems are called oversubscribed

planning problems. We can model a oversubscribed version of the satellite domain by not specifying any goal state for the *Observarion_Status* state variables, and make the *Stored* state a non-final state. This means that the *Observation_Status* becomes a **non-goal** state variable, and its evolution is not allowed to end at the *Stored* state. Since the only way to get to the *Downloaded* state is via *Stored* state, *Observation_Status* variable will always end its evolution on either at the *Not_taken* state (observation is not achieved) or at the *Downloaded* state (observation is achieved). Oversubscribed problems are essentially optimization problems. For this particular example the goal would be to achieve the maximum number of observations. This can be modeled in our constraint model by putting additional constraints (counting constraints) on the number of *Downlink* actions included in the plan.

2. **Antenna_Direction**(A_i): For each antenna A_i in the domain we create this state variable that describes the status of the antenna. Each antenna state variable has two states: *Ready_to_turn* and *Pointing_to_GS*. The first state describes that the antenna is free and can be turned to any direction. The second state represents that the antenna is pointing towards a ground station. Each TurnAntenna action changes this state variable's state to *Pointing_to_GS*. The *Pointing_to_GS* state is need by any Downlink action using this Antenna during its execution. Our model has a dummy zero duration action DonePointing(Antenna, GS) that causes an instantaneous change of state from *Pointing_to_GS* to *Ready_to_turn*. The following figure describes the state/transition graph of this state variable.


Downlink(Antenna, GS, Observation)
TurnAntenna(Antenna, GSFrom, GSTo)
Ready_to_turn
Pointing_to_GS
DonePointing(Antenna, GS)

For each state variable a solution represents a path from the initial state to the goal state in the state transition graph, where the path is a sequence of actions. For this state variable, each such path consists of instances of *TurnAntenna*, *Downlink*, and *DonePointing* actions. Let's consider an example of such a sequence as the following:

TurnAntenna(A,B,C) $\rightarrow$ *Downlink(A,D,O)* $\rightarrow$ *DonePointing(A,E)* $\rightarrow$ *TurnAntenna(A,D,E)* where A is the antenna, O is an observation, and B, C, D, and E are ground station locations.

If we look at the state variable in isolation, then the above sequence represents a solution (or part of it). But it doesn't represent a valid solution, because the antenna A was turned from B to C, the observation O was downlinked to D, then its state was changed to *Ready_to_turn* state from E and then it was turned from location D to E. A valid solution would be one where sequential changes in locations are consistent. For example, the following sequence represents a valid solution:

TurnAntenna(A,B,C) $\rightarrow$ *Downlink(A,C,O)* $\rightarrow$ *DonePointing(A,C)* $\rightarrow$ *TurnAntenna(A,C,E)*

An additional constraint is needed to make sure that for antenna the changes in locations that the antenna is pointing to are consistent.

To make sure that changes in the pointing direction of each antenna is valid we add a setup matrix to the state variable [2], where setup states are the ground station locations. Table 2 describes a part of the setup matrix with the setup states for the ground station locations B, C, D, and E. The time delay between the same setup states (a pair of ground

	B	*C*	*D*	*E*
B	0	inf	inf	inf
C	inf	0	inf	inf
D	inf	inf	0	inf
E	inf	inf	inf	0

Table 6.1: Setup Matrix for Antenna_Direction(A_i)

stations) is zero and for all other cases, it is set to infinity.

Actions *TurnAntenna(A, B, C)*, *Downlink(A,C,O)*, *DonePointing(A,C)*, and *TurnAntenna(A, C, E)* each has a state variable transition on the state variable *Antenna_Direction(A_i)*. Recall that in our model each transition has two setup states: a FromSetupState and a ToSetupState. The following table describes the setup states of the transitions in our example.
With these setup times, the (sub)sequence like the following:

TurnAntenna(A, B, C) $\rightarrow$ *Downlink(A,D,O)* $\rightarrow$ *DonePointing(A,E)*$\rightarrow$ *TurnAntenna(A,D,E)*

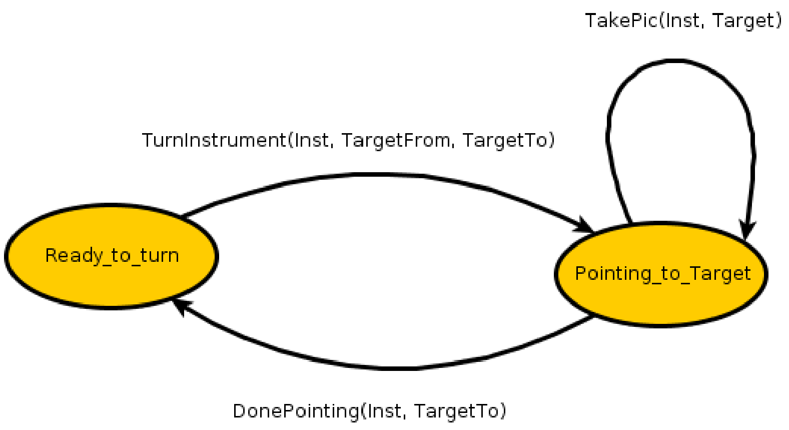
will be excluded from the solution because the actions' start and end times will violate other temporal constraints such as time windows and plan horizon.

[2]Setup matrix representation and related constraints are described in the previous chapter

Transitions	FromSetupState	ToSetupState
TurnAntenna(A, B, C)	B	C
Downlink(A,C,O)	C	C
DonePointing(A,C)	C	C
TurnAntenna(A, C, E)	C	E

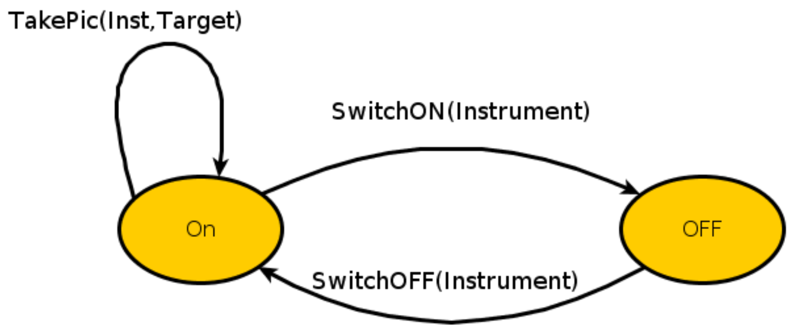
Table 6.2: Setup states of the transitions on Antenna_Direction(A_i)

3. **Instrument_Direction**(I_i): For each instrument I_i we create this state variable which is exactly same as the Antenna state variable described above, except that this state variable describes the states of an Instrument, and each instrument points towards a target. The *TurnInstrument* action changes the state of this state variable to the state *Pointing_to_target*, and TakePic action uses the state during its execution. The following picture describes state variable states and how actions can change from one state to other.


TakePic(Inst, Target)
TurnInstrument(Inst, TargetFrom, TargetTo)
Ready_to_turn
Pointing_to_Target
DonePointing(Inst, TargetTo)

   Note that the problem of possible inconsistent change in pointing location of an antenna can also happen for an instrument. To avoid it, we will use a similar setup matrix for this state variable as we used for antenna state variables. The setup states of this setup matrix are the target locations.

4. **Instrument_Status**(IS_i): Each instrument must be switched on before it can be used to take pictures. We create this state variable for each instrument with two states: *ON* and *OFF*. Each SwitchON action changes the state *OFF* to *ON*, and each SwitchOFF action changes the state *ON* to *OFF*. Each *TakePic* action needs the state *ON* during its execution. We assume at the beginning of the planning and at the end of the planning all instruments will be switched off. This means that the state *OFF* is both the initial and final state for this state variable.



Each instrument has additional constraints that state that the instrument can be switched on at a strech $MAXON(IS_i)$ units of time, and when its switched off, it must stay off at least $MINOFF(IS_i)$ units of time [3]. We model these constraints using the following two state-persistence constraints (as defined in the previous chapter):

- $persist(Instrument\_Status(IS_i), ON, 0, MAXON(IS_i))$
- $persist(Instrument\_Status(IS_i), OFF, MINOFF(IS_i), \mathrm{inf})$

Here inf represents an unbounded amount of time.

Note that in our formulation, each action can be used at most once in the plan. We will add multiple copies of *SwitchON* and *SwitchOFF* actions to represent the fact that within the planning horizon an instrument can be switched off or switched on multiple times. This modeling technique introduces some symmetry, because all these copies of *SwitchON* and *SwitchOFF* actions are equivalent. For example, if we added two copies of *SwitchON* and *SwitchOFF* actions in the model, then the following sequences are possible:

$$SwitchON_1 \rightarrow SwitchOFF_1 \rightarrow SwitchON_2 \rightarrow SwitchOFF_2 \tag{6.1}$$
$$SwitchON_2 \rightarrow SwitchOFF_2 \rightarrow SwitchON_1 \rightarrow SwitchOFF_1 \tag{6.2}$$
$$SwitchON_1 \rightarrow SwitchOFF_2 \rightarrow SwitchON_2 \rightarrow SwitchOFF_1 \tag{6.3}$$
$$SwitchON_2 \rightarrow SwitchOFF_1 \rightarrow SwitchON_1 \rightarrow SwitchOFF_2 \tag{6.4}$$

All four sequences above are valid but equivalent. This kind of symmetry may degrade the performance of the backtracking search, because it creates unnecessary choices.

We can remove this symmetry using a setup matrix with this state variable. By removing symmetry we mean making all the equivalent (sub)sequences of actions invalid except for one (the first one in our example (eq 6.1)). The setup states of this setup matrix

[3]Here $MAXON(IS_i)$ and $MINOFF(IS_i)$ are integer values

Transitions	FromSetupState	ToSetupState
SwitchON_1	$SwitchON_1$	$SwitchON_1$
SwitchON_2	$SwitchON_2$	$SwitchON_2$
SwitchOFF_1	$SwitchOFF_1$	$SwitchOFF_1$
SwitchOFF_2	$SwitchOFF_2$	$SwitchOFF_2$

Table 6.3: Setup states of the transitions on Instrument_Status(IS_i)

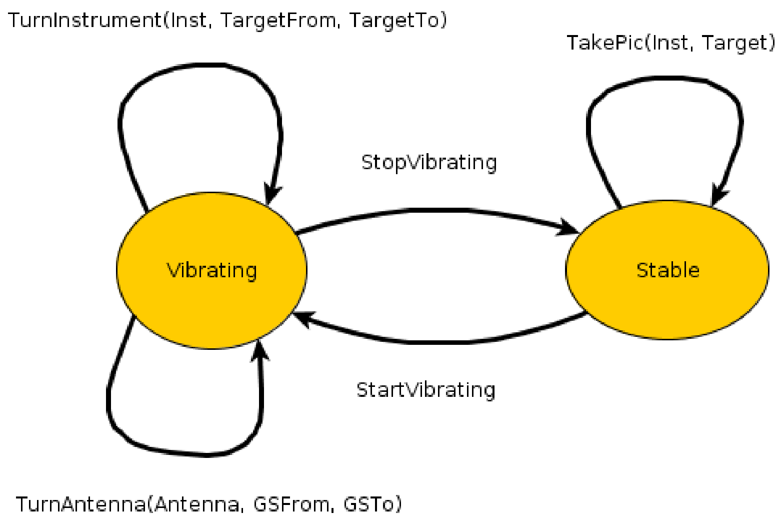
are the *SwitchON* and *SwitchOFF* action names, and each setup state pair describes the changeover time between the action pairs as described in the following table.

	$SwitchON_1$	$SwitchOFF_1$	$SwitchON_2$	$SwitchOFF_2$
$SwitchON_1$	inf	0	inf	inf
$SwitchOFF_1$	inf	inf	0	inf
$SwitchON_2$	inf	inf	inf	0
$SwitchOFF_2$	inf	inf	inf	inf

Each transition on this state variable has FromSetupState and ToSetupState defined as its action's name. The following table describes the setup states of the transitions. Using this setup matrix, all sequences, except for the first sequence described above (eq 6.1, become inconsistent. Note that the purpose of the setup matrix used here is different from the purpose it served in the *Instrument_Direction* and *Antenna_Direction* state variables. There the setup matrix is used to remove inconsistent sequences, and here it helps us to remove symmetry.

5. **Satellite_Mode(S_i)**: Recall that each turning action, either turning an antenna or an instrument, creates vibration in the satellite. Because of this vibrating effect, no picture-taking action can be executed on the satellite while any turning action is executing. This means that each satellite can be either in *stable* mode or in *vibrating* mode. When a turning (either of an antenna or an instrument) action is executing on-board, we will say that the satellite is in the vibrating mode, and in all other times we will say that the satellite is in the stable mode. Each TakePic action can only be executed when the satellite is in stable mode. This means that on a satellite any turning action can't overlap with any picture-taking action.

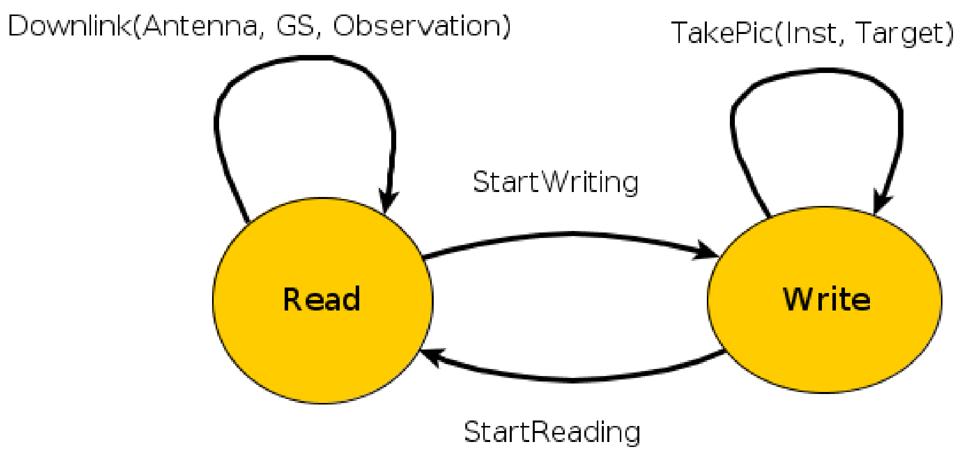
   To model this disjunctive constraint between the execution of picture-taking actions and turning actions, we create this state variable for each satellite S_i which has two states: *Stable* and *Vibrating*. We introduce two zero-duration actions: *StartVibrating* and *StopVibrating*. These actions, when performed, switch one mode to other instantaneously. Each turning action; *TurnAntenna* or *TurnInstrument*, while executing must have the satellite in the vibrating mode, and similarly during execution of any *TakePic* action the satellite must be in the stable mode, as described in the following figure.


TurnInstrument(Inst, TargetFrom, TargetTo)
TakePic(Inst, Target)
StopVibrating
Vibrating
Stable
StartVibrating
TurnAntenna(Antenna, GSFrom, GSTo)

Note that since we are going add more than one copies of the dummy actions *StartVibrating* and *StopVibrating*, we will use similar setup-matrix as in switching on and off actions to remove the symmetry.

6. **Satellite_Storage_Mode(R_i)**: Each satellite in this domain has a SSR that stores the pictures of the observations taken by instruments on-board. Each SSR supports two operations: writing on the SSR and reading from the SSR. One of the limitations of the SSR in our domain is that these two operations are mutually exclusive. This means that *TakePic* actions and *Downlink* actions can't be executed at the same time, because the *TakePic* action writes data on to the SSR and the *Downlink* action reads from the SSR.

   To model this mutually exclusive constraint on the SSR, we create this state variable for each SSR similar to the *Satellite_Mode* state variable as described above. It has two states: *Read* state, and *Write* state. Any reading operation (*Downlink* actions) can be performed while the SSR is in the *Read* state, and similarly any writing operation (*TakePic* actions) can only be performed while the SSR is in the *Write* state. To model this instantaneous change between these two states, we introduces two dummy zero-duration actions: *StartWriting* and *StartReading* as described in the following figure.



Note that since we are going add more than one copies of the dummy actions, we will use a similar setup-matrix as in *Instrument_Status* state variable to remove the symmetry.

6.2.3 Resources

1. **Satellite_Storage**(S_i): Each satellite S_i has a solid state storage (SSR) unit. Each SSR has a maximum capacity. We will assume at the start of the planning each SSR is empty. We will model this storage as a reservoir resource. Each TakePic action will store data into the SSR and each Downlink action will remove data from the SSR.

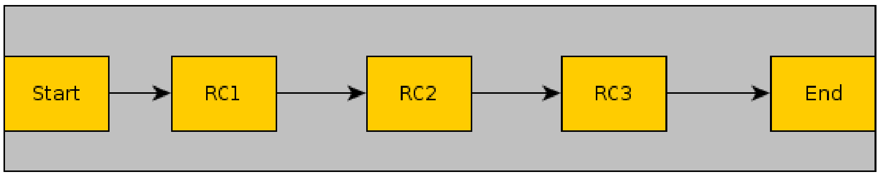
2. **Satellite_Battery**(S_i): Each satellite S_i is equipped with a battery that provides power to the satellite. Each battery has a maximum capacity. Each action except switching on and off instruments and recharging consumes power, and recharge actions produce power in the presence of the Sun. We will model each satellite's battery as a reservoir resource, and assume that at the beginning of the planning horizon all batteries are full of charge.

   Note that we ignored the amount energy consumed if an instrument is turned on over a period of time. That means, in reality if a instrument is in *ON*, then it should consume energy from the battery at a given rate. Since in our model, only transitions are allowed to consume (or produce) energy, we can't effectively model this scenario without extending the formulation.

3. **Satellite_Recharge_Access**(S_i, SW_j): For each Satellite-SunTimeWindow pair (S_i, SW_j) we create a set of recharge actions, where each action corresponds to the recharge rate of the satellite's battery. This means that recharge actions can't overlap. To model this non-overlapping constraint we create this unary resource for each Satellite and SunTimeWindow pair, and model the recharge actions to use this unary resource during their execution. This will guarantee that recharge actions will be executed in a sequence.

Note that, for any Satellite-SunTimeWindow pair, all possible of sequences of recharge action executions are equivalent. This means, if we have 3 recharge action to execute within the same SunTimeWindow: *Recharge-1*, *Recharge-2*, and *Recharge-3*, then all the execution sequences: {1,2,3}, {1,3,2}, {2,1,3}, {2,3,1}, {3,1,2} and {3,2,1} will produce same result. This symmetry in the model may cause problem for backtracking search.

We can adopt a similar technique of introducing a setup matrix to break the symmetry as used in the *Instrument_Status* state variable. The following figure describes an example where we have 3 recharge actions executed in the order *RC-1*, *RC-2*, and *RC-3*. Note that the *Start* and *End* actions are the dummy start and end action.



SunTimeWindow 1

To impose a total ordering between these three recharge actions (RC) we add the following setup matrix to this unary resource:

	Start	RC_1	RC_2	RC_3	End
Start	inf	0	inf	inf	0
RC_1	inf	inf	0	inf	0
RC_2	inf	inf	inf	0	0
RC_3	inf	inf	inf	inf	0
End	inf	inf	inf	inf	inf

Note that this setup matrix makes all the sequences except $RC_i \rightarrow RC_{i+1}$, inconsistent, and forces the RC_1 to be the first one to execute.

Time Window

Note that all the transitions on *Satellite_Recharge_Access(S_i, SW_j)* are constrained to be executed within the SW_j time-window. To model this we will add a time-window constraint on each *Satellite_Recharge_Access(S_i, SW_j)* resource as follows:

$$time - window(Satellite\_Recharge\_Access(S_i, SW_j), SW_j^{start}, SW_j^{end})$$

6.2.4 Actions

Recall that each action in our model consists of a set of transitions, where each transition has their own duration and start time that may have an offset from the start time of the action. In this section we will describe the actions and their transitions in detail. All the actions in the satellite domain have two common elements: first all transitions of each action have the same duration, and second none of the transitions of each action have any offset from start of the action. This means that for each action all transitions start at the same time, and finish at the same time. Each transition of an action has two setup states: FromSetupState and ToSetupState. We will use the setup state "default" to represent the case where the transition has no setup state.

We will describe the transitions in two different table formats: one for state variable transitions and the other for resource transitions. The following table describes the structure of the state variable transitions:

SV Name	FromState	ToState	FromSS	ToSS

The first column describes the name of the state variable that is affected by the transition, second and third column describes the state change caused by the transition. Note that for a PREVAIL transition, the columns FromState and ToState will have the same value. The last two columns describe the from and to setup states of the transition.

We will use a similar table structure to describe a resource transition. The only columns that are different from the state variable transition table are the "Requirement" and "Type" as described below.

RES Name	Req	Type	FromSS	ToSS

The "Requirement" column lists the resource requirement of the transition, and the "Type" column describes the type of the resource requirement. There are three types of resource requirements: BORROW, CONSUME and PRODUCE.

1. **TurnInstrument(Inst, TargetFrom, TargetTo)**: This action turns an instrument from a target location to another target location. We will model this action with two state variable transitions and one resource transition.

SV Name	FromState	ToState	FromSS	ToSS
Inst_Dir	Ready_To_Turn	Pointing_To_Target	TargetFrom	TargetTo
Sat_Mode	Vibrating	Vibrating	default	default

The first state variable transition is an EFFECT transition that changes the state variable *Instrument_Direction* from the state *Ready_to_Turn* to *Pointing_to_Target*. Note that this transition's FromSetupState is the location of the TargetFrom and ToSetupState state is the location of the TargetTo. The second transition is a PREVAIL transition on the state variable *Satellite_Mode* on the *Vibrating* state. This transition has no particular setup states.

The following CONSUME resource transition represents the resource consumption of this action ($B_{TurnInstrument}$) on the *Satellite_Battery* resource. This resource transition has no setup states.

RES Name	Req	Type	FromSS	ToSS
Sat_Battery	$B_{TurnInstrument}$	CON	default	default

The duration of this action (and all its transitions) depends on the slewing angle between the locations and the slewing speed of the instrument.

2. **TakePic(Inst, Target)**: This action represents taking a picture of an target using a satellite instrument. The duration of this action depends on the observation type. We will model this action with six state variable and two resource transitions.

   The following table describes the six state variable transitions.

SV Name	FromState	ToState	FromSS	ToSS
Observ_Status	Not_Taken	Stored	default	default
Inst_Dir	Pointing_To_Target	Pointing_To_Target	Target	Target
Inst_Status	ON	ON	default	default
Sat_Mode	Stable	Stable	default	default
Sat_Storage_Mode	Write	Write	default	default

   Only the first transition is an EFFECT transition that changes the state *Not_Taken* to the state *Stored* of the *Observation_Status* state variable. Other five transitions are PREVAIL transitions that represent the following facts: during execution of this action *Instrument_Direction* must have the state *Pointing_To_Target*, *Instrument_Status* must have the state *ON*, *Satellite_Mode* must have the state *Stable*, and *Satellite_Storage_Mode* must have the state *Write*.

   This action has two resource transition: a CONSUME transition that represent consumption of satellite's battery ($B_{TakePic}$) and another CONSUME transition to represent space consumption on Satellite's SSR ($D_{TakePic}$) to store the picture.

RES Name	Req	Type	FromSS	ToSS
Sat_Battery	$B_{TakePic}$	CON	default	default
Sat_Storage	$D_{TakePic}$	CON	default	default

Note that only one transition of this action has setup states defined. The PREVAIL transition on the state variable *Instrument_Direction* has both its setup states set to the target location of the action.

TimeWindows: Note that each Target is visible to an instrument only within certain time intervals. This means that for each $< Inst, Target >$ pair there is a set of time-windows $\{TW_1, TW_2, ..., TW_n\}$. To represent this constraint, we will create n (where n is size of the set of time-windows) copies of the *TakePic(Inst, Target)* action, one for each time interval in the set, and specify a time-window constraint on each copy as the following:

$$time - window(TakePic_{TW_i}(Inst, Target), TW_i^{start}, TW_i^{end})$$

3. **TurnAntenna(Antenna, GSFrom, GSTo)**: This action is similar to the *TurnInstrument* action. The duration of this action depends on the slewing angle between the locations and the slewing speed of the antenna. We will model this action with two state variable transitions and one resource transition.

   The following table describes the state variable transitions.

SV Name	FromState	ToState	FromSS	ToSS
Ante_Dir	Ready_To_Turn	Pointing_To_GS	GSFrom	GSTo
Sat_Mode	Vibrating	Vibrating	default	default

   The following CONSUME resource transition represents the resource consumption of this action on the *Satellite_Battery* resource ($B_{TurnAntenna}$).

RES Name	Req	Type	FromSS	ToSS
Sat_Battery	$B_{TurnAntenna}$	CON	default	default

   Only the EFFECT transition on *Antenna_Direction* has setup states defined; the FromSetupState is the GSFrom location and the ToSetupState state is the GSTo location.

4. **Downlink(Antenna, GS, Observation)**: This action downloads a picture from a satellite's SSR to a ground station. The duration of this action depends on the size of the observation and the transfer rate of the antenna. We will model this action with two state variable transitions and two resource transitions.

SV Name	FromState	ToState	FromSS	ToSS
Observ_Status	Stored	Downloaded	default	default
Sat_Storage_Mode	Read	Read	default	default

The EFFECT transition on the state variable *Observation_Status* will change the state from *Stored* to *Downloaded*, and the PREVAIL transition requires the *Read* state on the state variable *Satellite_Storage_Mode*.

This action has a CONSUME resource transition on the *Satellite_Battery* resource ($B_{Downlink}$) and a PRODUCE transition on the *Satellite_Storage* resource ($D_{Downlink}$).

RES Name	Req	Type	FromSetupState	ToSetupState
Sat_Battery	$B_{Downlink}$	CON	default	default
Sat_Storage	$D_{Downlink}$	PROD	default	default

Note that none of the transitions of this action has any setup state.

TimeWindows
Note that each ground station is visible to an antenna only within certain time intervals. To represent this constraint, we will create copy of the Downlink(Antenna, GS, Observation) action for each such time interval and specify the following time-window constraint, as we did for the *TakePic(Inst, Target)* actions above.

$$time - window(Downlink_{TW_i}(Antenna, GS, Observation), TW_i^{start}, TW_i^{end})$$

5. **SwitchON(Inst)**: This action represent switching on an instrument. It has a single EFFECT transition that changes the state variable *Instrument_Status*'s state from *OFF* to *ON*.

SV Name	FromState	ToState	FromSS	ToSS
Inst_Status	OFF	ON	SwitchON	SwitchON

Note that the state variable transition's $\mathsf{FromSetupState}$ and $\mathsf{ToSetupState}$ are the name of the action as we have discussed before in the overview of the *Instrument_Status* state variable. We assume that this action doesn't consume any resource, because its real power consumption is negligible for planning and scheduling purposes. We model the **SwitchOFF(Inst)** action in the same way.

6. **Recharge(Satellite, SunTimeWindow)**: This action represents an unit recharge of the Satellite's battery during a given SunTimeWindow. This action has one PRODUCE resource transition on the *Satellite_Battery* resource, and a BORROW transition on the *Satellite_Recharge_Access* resource for the given satellite and the SunTimeWindow.

RES Name	Req	Type	FromSetupState	ToSetupState
Sat_Battery	p	PROD	default	default
Sat_Rechg_Acc	1	BORR	default	default

Note that p is the recharge rate of the satellite battery. Since this action represents a unit of recharge, its production is p on the satellite battery. *Satellite_Recharge_Access* is a unary resource, all resource requirements on it will have the demand of 1.

6.3 Limitations

In this chapter we have described how to model a complex satellite problem, that has both planning and scheduling problem characteristics, using the transition-based representation. We have chosen the satellite problem because of its wide range of complexities and familiarity in the AI planning and scheduling community. We have shown how our representation can express most of the complexities in the satellite problem, however there are a few features that our representation couldn't express due to the following limitations:

1. **Continuous Resources**: In our representation resources are modeled with discrete capacities and requirements. In reality resources like the battery and the storage device of a satellite are continuous resources. We approximate the behavior of these resources by discretizing the resource requirements on them.

2. **State Resource Consumption**: There are some cases where a state variable consumes resources by being in a particular state. For example , the instruments while in the state "ON" will consume power continuously. We are unable to model that in the current representation.

3. **Requirements and Durations are fixed**: In our current representation resource requirements and durations are fixed integer quantities. This model is a restrictive model of transitions' resource requirements, because in reality many transitions' resource requirement would be dependent on their duration, where the duration of a transition would be a decision.

A continuous resource model provides a more accurate representation of the real world. However, the problem with continuous resources is that they are very hard to solve. That's why we have chosen the discrete resource model, as in most scheduling problem models. Generally, approximating a continuous resource requirement, like recharging a battery, into discrete steps is acceptable. We have shown in the problem representation chapter (Chapter 2, page 36) how we can discretize a continuous action into different levels precision. The choice of level of discretization often depends on the application in hand.

The last two items are related to each other. If we could model the the third item, then we could also extend our representation to model state resource consumption in a simple way. We will discuss this further in the next chapter (see Section 7.2.1).

6.4 Discussion

The two main aims of our proposed modeling approach is to simplify modeling of complex problems with simple constructs, and to provide a clear solution approach. We believe that a constraint-based search is best suited for solving problems that are in between planning and scheduling. That is why have chosen the compilation of the representation to a CSP. However, other solution approaches such as state-space or plan-space search could also be implemented for the representation.

We believe our representation simplifies the modeling of complex problems by providing simple and intuitive constructs like state variables, resources, actions and transitions. In the following subsection we will describe some of the distinguishing features of our representation.

6.4.1 Features of the Problem Reresentation

State Variable modeling: Our state variable model allows the domain modeler to specify different constraints like min-max achievement, min-max persistence, and time-windows on change of state (achieve/change before/after constraints) on each individual state of state variables. These constraints translate into simple temporal and counting constraints in our (compiled) constraint model as described in Chapter5. Constraints like these can also be modeled in other languages, such as PDDL2.1. In PDDL2.1 these constraints are modeled using additional state variables called Timed Initial Literals (TIL) [14], mainly to model time-window constraints , and special types of actions called “strut” and “clip” [25]. A strut action models the minimum separation constraint between two happenings[4], and clip action forces two happenings to coincide within a time interval. These special actions also introduce additional state variables. All these additional state variables need to be added on selective sets of existing action pre-conditions in the domain. We argue that our method of specifying constraints on state variables and on its states is much simpler than the modeling techniques available in PDDL2.1, because in our representation the domain modeler specifies these constraints as properties of a state variable and its state. All necessary temporal and counter constraints are handled in the implementation level (i.e. our compilation). In PDDL2.1 it is the job of the domain modeler to model these constraints via additional state variables and actions, which makes the model complicated to understand and harder to maintain in the long run.

Resource Modeling: We have simplified the resource modeling by categorizing types of resources and types of resource requirements, which provides a simpler and intuitive construct for resource modeling. These categorizations also let us develop specialized constraints and propagation techniques for different types of resources (as shown in Chapter 4). In AI planning languages resources are generally modeled in very generic way as numerical fluents. As pointed out by Boddy [8], this generic modeling of resource makes it hard for AI planners to

[4]Happenings are generally start and end points of actions in PDDL

exploit the special structures of different types of resource usages.
Action Modeling: In our action model we group the pre-conditions and effects of actions in the form of transitions. We believe this representation makes the modeling of actions simple for the modeler because it only expresses what an action does on a state variable or resource without explicitly stating the execution conditions. This action model can express complex behaviors of actions with delayed effects, different durative effects, more than one effect on the same resource or state variable etc. Although in this case study we have kept the action representations of the satellite domain simple, in Chapter 2[5] we have shown a complex version of *TurnInstrument* action (adopted from [46]). Like the complex version of *TurnInstrument* action, the intermediate effects of an durative action can also be modeled in PDDL2.1, but not directly as it is done in our representation. In PDDL2.1, the action with intermediate effects is decomposed in several durative actions, one for each intermediate happening time point. Then additional clip actions (as described above) are introduced with this set of decomposed actions to ensure all these durative actions are sequenced. We argue that our action-model is much more intuitive and simple to express than the decomposition technique needed in PDDL2.1.
Generalizing Setup Matrices: We have introduced setup matrices for both resources (as done in many scheduling models) and state variables. In our model we use setup matrices for the traditional usage, for expressing time delay between two consecutive transitions, as well as for two other reasons: excluding invalid action sequences (as shown in the *Instrument_Direction* state variable), and removing symmetry from action sequences (as shown in the *Instrument_Status* state variable). This gives the domain modeler power to express action-sequence related constraints in a simple way. On the other hand, in PDDL2.1 to represent sequence dependent setup times we need add additional strut actions between each pair of actions that needes to be sequenced, where duration of the strut action will be the setup time. This makes the domain modeler's job harder, and makes the domain description messy.

6.4.2 Modeling Abstraction for Planning/Scheduling Problems

There are other constraint modeling languages such as ZINC [16], OPL, and others that provide an abstraction layer on top of a constraint solver. These languages aim to provide modeling ease for a wide variety of constraint satisfaction problems, from Suduku to job-shop problems. Our modeling representation can be seen as a specialization of these languages, targeted at problems that fall in the category of problems that are in between planning and scheduling problems. Although problems in this category can also be modeled using languages like ZINC or OPL, we believe our representation provides a higher level abstraction that makes the job of the domain modeler easier.

[5]see page 27

6.5 Experiment

We have implemented a planner called "TransPLAN"(written in C++) that incorporates the modeling elements of the transition-based framework for planning and scheduling problems, the compilation of the model to a CSP and a solver for the compiled CSP. TransPLAN provides several APIs that a modeler can use to model problems in terms of state variables, resources, actions and transitions, and solve planning and scheduling problems.

In this section we describe the result of solving instances of Satellite scheduling problem with TransPLAN. First we will describe the parameters used for instance creation for the Satellite domain. Then we will describe the branching heuristic used for solving the CSP. We conclude the experiment section with a brief discussion about the observations.

6.5.1 Problem Instances

The problem instances used in the experiment are created by varying different parameters. Following are the description of the key parameters.

Parameters	Description
Num_Satellite	Number of available satellite in the problem
Ins_Type	Number of unique types of instruments. Each instrument type has a battery usage rate, an angular speed and a rate for processing data.
Ant_Type	Number of unique types of antennas. Each antenna type has a battery usage rate, an angular speed and a rate for downloading data.
Num_Groudstation	Number of available ground stations.
Num_Observation	Number of observations. Each observation has a randomly assigned size.
Time windows	This parameter defines the number of time windows available for satellite where sun is visible, where a ground station is visible, and where observation is visible. Although number of time windows are created for all of the above are same, each type of time-windows start time, duration , and time delay between two consecutive time-windows are assigned randomly.

We have created instances with 2-4 satellites, 2-3 ground stations, 3-4 time windows, 2-3 instrument and antenna types, and 3-9 observations.

6.5.2 Branching Heuristic

To solve any CSP effectively, a good branching heuristic is important. We have discussed our branching strategy in Section 4.1. In this experiment we have chosen a branching heuristic that simulates a forward progression search. It tries to build a path for each state variable, and a flow network for each resource, starting from the dummy start transition and ending at the dummy end transition. At each branching point we choose a pair of transitions T_{var} and T_{val}, where T_{var} and T_{val} are transitions of same domain object (either a state variable or a resource), and T_{var} can provide support to T_{val}. The following heuristic is used to choose a transition pair to branch on.

For each domain object d (either a state variable or a resource) we create two sets of transitions: $Can\_Support(d)$ and $Need\_Support(d)$. Each transition in these sets must be included in the plan. Transitions in $Can\_Support(d)$ set are able to provide support to other transition, and transitions in $Need\_Support(d)$ set needed support from other transitions.

We calculate a transitive closure set $Need\_Support(d*)$ using the $\mathrm{PossAchiev}$ (for state variables) or $\mathrm{PossSupp}$ (for resources) [6] relation of transitions in $Need\_Support(d)$. $Need\_Support(d*)$ set represents all possible paths from the dummy start transition to each transition in $Need\_Support(d)$ set on d.

For each transition T in $Can\_Support(d)$ set we create a pair $< T, T' >$ such that T' exists in the set $Need\_Support(d*)$ and T can provide support to T'. This means T' must exists in either $\mathrm{PossFollow}(T)$ if d is a state variable or in $\mathrm{PossSupp}(T)$ if d is a resource. We define end time of each pair $< T, T' >$ as the earliest end time of T' in the current state.

For each domain object d, we select a pair, $< T, T' >_d$ that has the minimum end time. Then we select the pair T_{var} and T_{val} as the pair among the domain objects that has minimum end time.

6.5.3 Result and Discussion

In the experiment we measure two main aspects of TransPLAN: search quality in terms of number of failures, and time to find a valid solution. Figure 6.1 illustrates the search quality or effectiveness of the branching heuristic (Y-axis) for each problem instance (X-axis). From the result we can observe that the number of failures are mostly zero irrespective of the size of a search problem (measured in number of actions/transitions). This means the branching heuristic is able to solve most problem instances bracktrack free. We think this is because of our propagation and inference rules that produced tighter domains of the CSP variables.

Figure 6.2 shows the solution time for each problem instance. Form the result we can observe that the solution time increases with the number of actions and transitions. This is caused by the fact that increment in number of actions and transitions increases the number of variables and constraints in the CSP, and propagation and inference rules take more time to reach a fix point. In this particular problem doamin, the number of dummy actions (such as StartVibrating/StopVibrating, StartReading/StartWriting etc.) has a significant effect on the

[6] Please see page 89, page 105, and page 93 for definition of $\mathrm{PossSupp}$, $\mathrm{PossFollow}$ and $\mathrm{PossAchiev}$ sets.

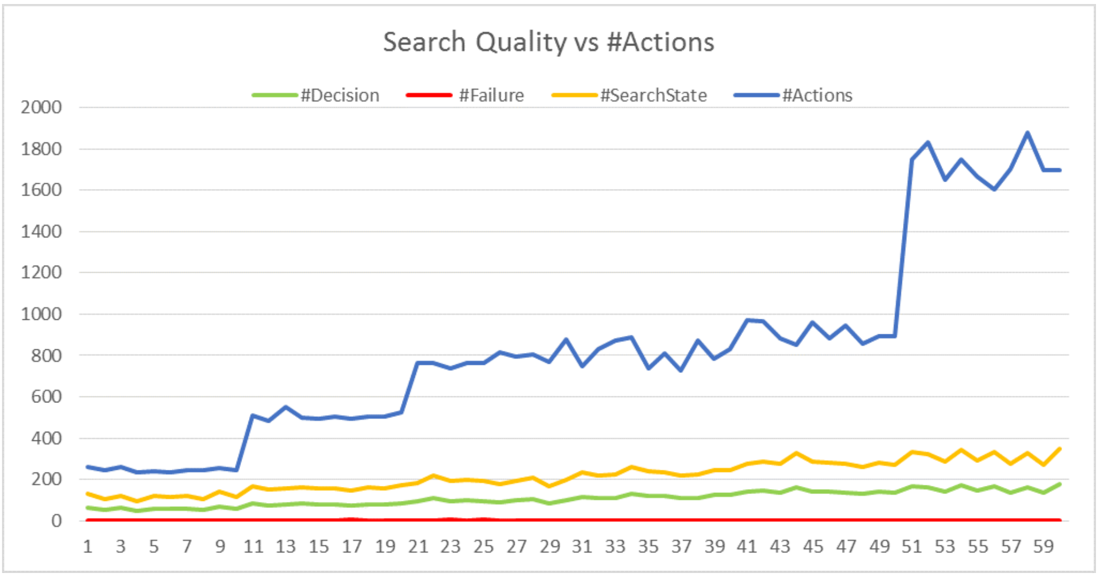


Figure 6.1: Search quality

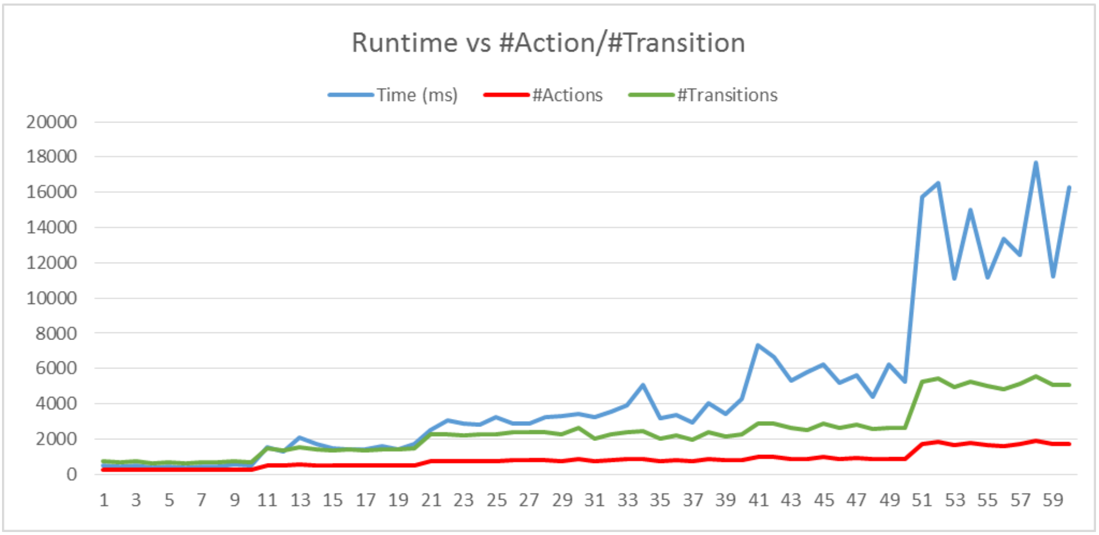


Figure 6.2: Solve time

size of the CSP. We have added each such dummy action ($2 * \#Observation$) times. Another increment in the number of actions is due to the unit recharge actions created for each possible recharge time window for each satellite. Number of such recharge actions depends on the number of time windows per satellite and the duration of time windows.

6.6 Summary

The main goal of this chapter is to demonstrate how a realistic problem, that integrates planning and scheduling aspects, can be modeled in our proposed transition based representation. We have chosen a complex version of the well known (within the AI planning community) satellite problem domain. We incorporate many complex constraints in this domain, like time windows for actions and states, complex resource usage, different modes of the SSR of satellites etc. We have shown that we can successfully model most of the complexities in a simple way.

Chapter 7

Conclusion

The aim of this thesis is to model and solve problems that are in between planning and scheduling. These problems have scheduling constraints like time-windows on actions, capacited resources, sequence-dependent setup times between activities on resources etc. In addition to these classical scheduling constraints, usually there are various complex action choices (planning-constraints) that have cascading effects. In this thesis we have proposed a solution for modeling and solving this particular class of problems via integrating planning and scheduling techniques in a uniform framework. The proposed solution has three main parts: modeling, describing the modeling tools; compilation to CSP, describing an automatic compilation of the model to a CSP; and propagation and inference techniques for solving the CSP.

We conclude the thesis in this chapter by summarizing the contributions, and discuss some future research directions that will naturally follow the current work.

7.1 Thesis Summary

In this section we will summarize the work in the main three parts of the thesis: problem representation, the constraint model, and the propagation and inference techniques for the constraint model.

7.1.1 Problem Modeling

We have extended the multi-valued state variable planning representation language with individual state constraints for state variables, explicit representation of different types of resources and resource requirements, and a compact action-transition model. The individual state-constraints for state variables enable us to express complex temporal constraints easily. Explicit representation of resources, as commonly done in scheduling problems, make it easy for users to express common resource-related constraints like capacity constraints, and it also gives a way to exploit the structure of resource requirements, as we have done in the constraint model for the resources (support links). The action-transition model makes it easy to model complex features of actions like delayed effects, effects with different durations on different domain objects etc. The semantics of each type of transition encapsulates the execution conditions at start, during, and at the end of the transition. This simplifies the task of modeling a complex action from a user point of view. Although we have only considered a discrete model of resource requirements, we have shown how we can model monotonic and non-monotonic continuous resource requirements with different granularity.

We defined a solution to a planning problem as a *flexible plan*. A flexible plan is defined as a partially ordered schedule of actions, where action start times are intervals. Partially ordered schedules are flexible, and therefore more robust to unexpected delays. A flexible plan creates a partially ordered schedule (POS) of transitions on each state variable and on each resource, where each POS represents a set of valid executions of transitions.

The main goal of our proposed modeling framework is to make the modeling task easier for a user by providing enough features to model all necessary complexities of the problem and hiding the complexity of the execution correctness within the semantics of these features. Although this means that our modeling language is somewhat more restrictive than other planning languages like PDDL or ANML, our modeling language can express most of the complexities associated with the problems that are in between planning and scheduling as we have demonstrated in the case study in Chapter 6.

7.1.2 Constraint Model

We believe that the constraint-based search framework is best suited for solving the class of problems that we are interested in. This is why we have provided an automatic compilation for our representation to a CSP. This compilation is bounded by the number of action occurrences. Our constraint model can be seen as a system of CSPs, one for each state variable and resource, that are synchronized by a simple temporal network (STN) for action start times. Each CSP for a state variable or resource is based on the concept of precedence constraints which implies temporal constraints on the action STN. Central to our constraint model is the explicit representation and maintenance of the precedence constraints between transitions on same domain object. For state variables, the main decision variables are the **causal link**, achieve variables, and for resources the main decision variables are based on **support link**, support variables. Both these decision variables imply precedence constraints when assigned. These variables unify both planning and scheduling decision making within a constraint framework.

Solving the constraint model entails posting a minimal number of precedence constraints such that goal conditions are achieved and no other constraints are violated. We have shown how a solution to our constraint model indeed gives us a valid flexible plan, which creates a valid partial order schedule on each state variable and resource.

7.1.3 Branching, Propagation and Inference

Precedence constraints between transitions play a central role in our constraint model. We have proposed a branching scheme that branches on the two main decision variables : the `achieve` variables and the `support` variables. Our branching scheme provides an alternative approach to the two step process of the constraint posting search for partial order scheduling, which first finds a resource conflict, and then posts additional precedence constraints to resolve the conflict. Based on this branching strategy we have shown how to infer new precedence relations from temporal and mutex constraints, and how to infer tighter temporal bounds from the precedence constraints.

Each supporting decision implies an explicit precedence constraint. The effects of each of these constraints are propagated via temporal and resource constraints using the propagation rules as described in Chapter 4. Based on a propagated and consistent search state, our inference rules finds two main types of constraints: additional precedence constraints, and tighter bounds on the temporal and resource support variables. Note, since the precedence constraints play a central role in our constraint model, the main aim of our inference techniques is to derive more implied precedence constraints.

We have shown that the proposed inference techniques can deduce temporal bounds and precedence relations comparable to the propagation techniques based on absolute temporal values when temporal constraints are tight (see Section 4.6.2.1 on page 112). Furthermore, we have compared our inference techniques with Laboire's work[36] on the Energy Precedence constraint (see Section 4.4.2.1 on page 102) and the Balance constraint (Section 4.6.2.2 on page 114), which are based based on relative temporal relations. All these propagation techniques tighten the bound on the temporal variables. Our inference and propagation techniques not only consider the transitions and actions that are included in the plan but can also deduce new constraints for activities that are not yet included in or excluded from the plan. We have shown that our inference techniques can also infer bounds as good as the inference techniques mentioned above, and in addition infer additional precedence constraints.

7.2 Future Work

7.2.1 State Resource Consumption

In many real life problems, state variables may consume or produce resources when in a particular state. For example, consider the satellite problem described in the previous chapter, where each instrument of a satellite continuously consumes battery while being switched on. Similarly a generator will consume fuel and produce power at a constant rate while it is running.

In the current modeling framework we don't have the capability to model this feature. In some cases, where the effect of state-resource consumptions are insignificant we can ignore them in the approximated planning model. But in many cases these state-resource consumptions have a big effect on decision making. That's why we think it is important to extend our modeling framework to include them in the future.

To be able to represent state-resource consumptions we need to know two things: how long the domain object was in the state and what is the rate of consumption. Then we can calculate the resource requirements for the state. In our current model we have assumed that the resource requirements of a transition are given as input. First we need to extend our model to represent resource requirements of a transition as function of its duration and resource requirement rate. Second, to know how long the domain object exists in a particular state, we need to extend our transition model by introducing a *GAP-Transition* between each pair of consecutive EFFECT transitions in the plan as described in the following figure. Since a plan creates a sequence of

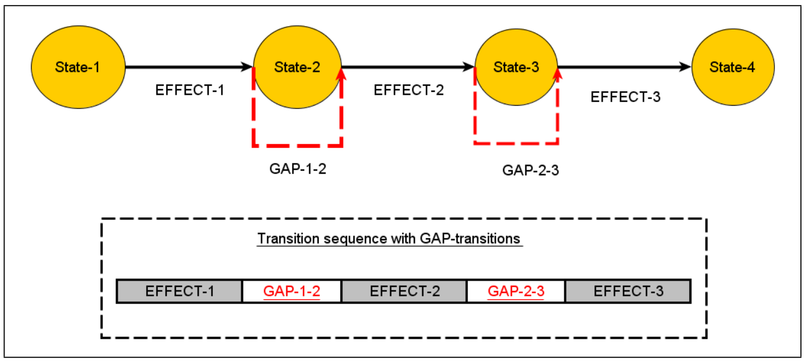


Figure 7.1: Adding GAP transition to model resource consumption

EFFECT transitions on each state variable, each GAP transition in between two consecutive EFFECT transition would represent the duration that each state persists. With this extension together with the change where we define resource requirement as function of duration and requirement-rate, we will be able to model problems with state-resource consumptions.

7.2.2 Generalized Resource Model

In our framework we can't model resources that are a mix of reusable and reservoir resources. In practice there can be resources that are both a reservoir resource and a reusable resource. For example consider water as a resource in a chemical plant, where water is produced as a by-product, used as coolant, and consumed as a solvent. If we assume that all the water is stored in a water tank, then it will not be possible to model the water tank as a resource in our modeling framework. In the future we would like to extend the resource model to have only one type of resource which can have three different resource requirements: BORROW, PRODUCE and CONSUME. This would be an easy extension, because each BORROW transition can be thought of as a combination of CONSUME (at the start) and PRODUCE (at the end

) transitions, as pointed out in Chapter 2. This means that a BORROW transition could sup-

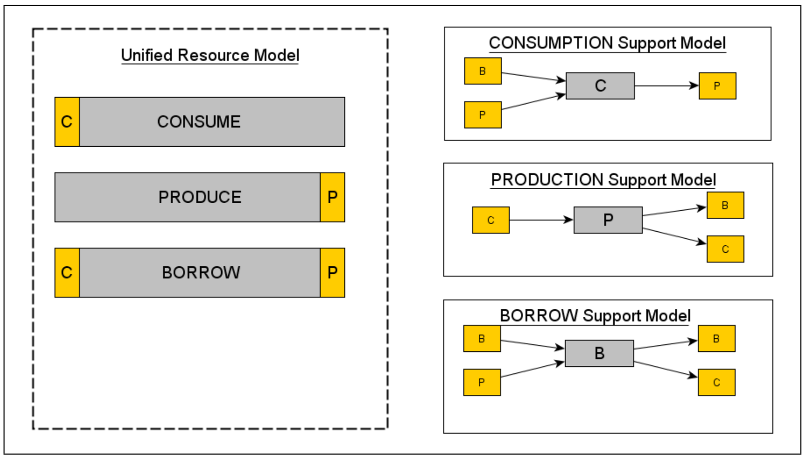


Figure 7.2: General resource model with 3 types of requirements

port CONSUME transitions and other BORROW transitions, and can be supported by other BORROW transitions and PRODUCE transitions as described in the figure 7.2.

7.2.3 Evaluation of Efficiency

In this thesis we have discussed how to model a planning problem that lies between planning and scheduling, and how to solve it via an automatic compilation to a CSP. To be able to solve problems of realistic size we need to make sure our constraint-search algorithm is efficient. There are three aspects of solving a CSP efficiently: a branching strategy that defines the search space, efficient propagation and inference techniques that help to prune the search space, and good heuristics for branching choices to guide the search.

We have proposed a new branching strategy for solving the CSP similar to assigning causal links in POP planning systems, which provides an alternative to the precedence constraint posting (PCP) approach used in constraint-based scheduling. We have compared our method for finding partial order schedules (POS) on resources with two other state of the art techniques [3]. The first technique is an envelope-based complete search technique that posts precedence constraints when resource envelopes violate the capacity constraints of the resources. The second approach is a two-step approach which creates a POS from a fixed time solution using the chaining procedure (see Section 4.6.1.2 on page 110). Our branching strategy together with Inference 4 and Inference 6 proved to be effective for solving RCPSP/max instances. We compared our results with the two state of art approaches on two quality criteria of robustness: “flexibility” and “fluidity” of the POS. Our method outperformed the envelope based PCP method, and found results as good as, and often better than to the two-step approach, but was not as efficient.

A detailed empirical evaluation of our branching and propagation techniques on problems with both planning and scheduling characteristics and comparison with the state of the art is due. Also another aspect of solving CSPs efficiently, which we have not investigated so far, is the branching heuristics. Finding an efficient heuristic for the constraint-based search algorithm that will solve all problems efficiently in this class of problems is a non-trivial problem to solve. We will consider these two topics as our future research subjects.

Bibliography

1. AIPS-98 Planning Competition Committee. PDDL - The Planning Domain Definition Language. CVC TR-98-003/DCS TR-1165. Technical report, Yale Center for Computational Vision and Control, 1998.

2. Christer Bäckström and Bernhard Nebel. Complexity results for sas+ planning. *Computational Intelligence*, 11:625–656, 1995.

3. Debdeep Banerjee and Patrik Haslum. Partial-Order Support-Link Scheduling. In Fahiem Bacchus, Carmel Domshlak, Stefan Edelkamp, and Malte Helmert, editors, *ICAPS*. AAAI, 2011.

4. Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-based Scheduling: Applying Constraint Programming to Scheduling Problems*. Kluwer Academic Publishers, 2007.

5. Tania Bedrax-weiss, Conor Mcgann, and Sailesh Ramakrishnan. Formalizing resources for planning. In *In Proceedings of the ICAPS-03 Workshop on PDDL*, 2003.

6. Sara Bernardini and David E. Smith. Developing domain-independent search control for europa2. In *ICAPS-07 Workshop on Heuristics for Domain-independent Planning: Progress, Ideas, Limitations, Challenges*, 2007.

7. L. AU Dell'Olmo P. L. Bianco Bianco, P. Dell Olmo, and M. Grazia Speranza. Heuristics for multimode scheduling problems with dedicated resources. *European Journal of Operational Research*, 107(2):260–271, June 1998.

8. M. S. Boddy. Imperfect Match: PDDL 2.1 and Real Applications. *Journal of AI Research*, 20:133–137, 2003.

9. Peter Brucker, Andreas Drexl, Rolf Mohring, Klaus Neumann, and Erwin Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3–41, January 1999.

10. A. Cesta, A. Oddi, and Stephen Smith. A constraint-based method for project scheduling with time windows. Technical Report CMU-RI-TR-00-34, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, February 2000.

11. Amedeo Cesta and Angelo Oddi. A formal domain description language for a temporal planner. In Marco Gori and Giovanni Soda, editors, *Topics in Artificial Intelligence*, volume 992 of *Lecture Notes in Computer Science*, pages 255–260. Springer Berlin , Heidelberg, 1995.

12. Amedeo Cesta and Cristiano Stella. A time and resource problem for planning architectures. In Sam Steel and Rachid Alami, editors, *ECP*, volume 1348 of *Lecture Notes in Computer Science*, pages 117–129. Springer, 1997.

13. Amanda Jane Coles, Andrew Coles, Maria Fox, and Derek Long. Colin: Planning with continuous linear numeric change. *J. Artif. Intell. Res. (JAIR)*, 44:1–96, 2012.

14. S. Cresswell and A. Coddington. Planning with timed literals and deadlines. In J. Porteous, editor, *Proceedings of the 22nd Workshop of the UK Planning and Scheduling Special Interest Group*, pages 22–35. University of Strathclyde, 9-10 December 2003. ISSN 1368-5708.

15. Smith David E., Frank Jeremy, and Cushing William. The anml language. In *ICAPS Workshop on Knowledge Engineering for Planning and Scheduling*, 2008.

16. M. Garcia de la Banda, K. Marriott, R. Rafeh, and M. Wallace. The modelling language zinc. In *12th International Conference on Principles and Practice of Constraint Programming*, LNCS, pages 700–705. Springer, 2006.

17. Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

18. Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. In *KR*, pages 83–93, 1989.

19. Minh Binh Do and Subbarao Kambhampati. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artificial Intelligence.*, 132(2):151–182, 2001.

20. F. Dvorak and R. Bartak. Ai Planning with Time and Resource Constraints. In *Znalosti*, pages 72–83, 2010.

21. Amin El-Kholy and Barry Richards. Temporal and resource reasoning in planning: the parcplan approach. In W. Wahlster, editor, *Proccedings of ECAI 96*. John Wiley {& Sons, 1996.

22. Michael D. Ernst, Todd D. Millstein, and Daniel S. Weld. Automatic sat-compilation of planning problems. In *IJCAI*, pages 1169–1177, 1997.

23. Markus P.J. Formhrez. Constraint-based scheduling. In *Invited tutorial at the American Control conference (ACC'01)*, June 2001.

24. Maria Fox and Derek Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:2003, 2003.

25. Maria Fox, Derek Long, and Keith Halsey. An investigation into the expressive power of pddl2.1. In Ramon López de Mántaras and Lorenza Saitta, editors, *ECAI*, pages 328–342. IOS Press, 2004.

26. Jeremy Frank. Bounding the resource availability of partially ordered events with constant resource impact. In M. Wallace, editor, *Proccedings of CP 2004*, pages 242–259, 2004.

27. S. Fratini, F. Pecora, and A. Cesta. Unifying Planning and Scheduling as Timelines in a Component-Based Perspective. *Archives of Control Sciences*, 18(2):231–271, 2008.

28. Malik Ghallab and Hervé Laruelle. Representation and Control in IxTeT, a Temporal Planner. In *AIPS*, pages 61–67, 1994.

29. K. Halsey, D. Long, and M. Fox. CRIKEY- a Planner Looking at the Integration of Scheduling and Planning. In *Proceedings of the Workshop on Integration Scheduling Into Planning at ICAPS'03*, pages 46–52, June 2004.

30. Jörg Hoffmann, Carla P. Gomes, Bart Selman, and Henry A. Kautz. Sat encodings of state-space reachability problems in numeric domains. In Manuela M. Veloso, editor, *IJCAI*, pages 1918–1923, 2007.

31. Jorg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

32. Ari K. Jónsson, Paul H. Morris, Nicola Muscettola, Kanna Rajan, and Benjamin D. Smith. Planning in interplanetary space: Theory and practice. In *AIPS*, pages 177–186, 2000.

33. Subbarao Kambhampati. Model-lite planning for the web age masses: the challenges of planning with incomplete and evolving domain models. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 2*, pages 1601–1604. AAAI Press, 2007.

34. Henry Kautz and Joachim P. Walser. Integer optimization models of ai planning problems. *Knowledge Engineering Review*, 15(1):101–117, 2000.

35. Henry A. Kautz and Bart Selman. Planning as satisfiability. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI'92)*, pages 359–363, 1992.

36. Philippe Laborie. Algorithms for propagating resource constraints in ai planning and scheduling: Existing approaches and new results. *Artif. Intell.*, 143(2):151–188, 2003.

37. David Mcallester and David Rosenblitt. Systematic nonlinear planning. In *In Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 634–639, 1991.

38. Nicola Muscettola. HSTS: Integrating planning and scheduling. Technical Report CMU-RI-TR-93-05, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, March 1993.

39. Nicola Muscettola. Computing the envelope for stepwise-constant resource allocations. In P. Van Hentenryck, editor, *Proccedings of CP 2002*, pages 139–154. Springer-Verlag, 2002.

40. Claude Le Pape. Edge-finding constraint propagation algorithms for disjunctive and cumulative. In *Proceedings of 15th Workshop of the U.K. Planning Special Interest Group*, 1996.

41. J. Scott Penberthy and Daniel S. Weld. Temporal planning with continuous change. In *AAAI*, pages 1010–1015, 1994.

42. Nicola Policella, Amedeo Cesta, Angelo Oddi, and Stephen F. Smith. From precedence constraint posting to partial order schedules. a csp approach to robust scheduling. *AI Communications*, 20(3):163–180, 2007.

43. Cédric Pralet and Gérard Verfaillie. Using Constraint Networks on Timelines to Model and Solve Planning and Scheduling Problems. In Jussi Rintanen, Bernhard Nebel, J. Christopher Beck, and Eric A. Hansen, editors, *ICAPS*, pages 272–279. AAAI, 2008.

44. Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming*, CP '98, pages 417–431, London, UK, UK, 1998. Springer-Verlag.

45. Ji-Ae Shin and Ernest Davis. Processes and continuous change in a sat-based planner. *Artif. Intell.*, 166(1-2):194–253, 2005.

46. David E. Smith. The case for durative actions: A commentary on pddl2.1. *Journal of AI Research*, 20:149–154, 2003.

47. David E. Smith, Jeremy Frank, and Ari K. Jonsson. Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15(1):47–83, 2000.

48. Stephen Smith and C. Cheng. Slack-based heuristics for constraint satisfaction scheduling. In *Proceedings 11th National Conference on Artificial Intelligence*, July 1993.

49. Stephen F. Smith and Marcel Becker. An ontology for constructing scheduling systems. In *Working Notes from 1997 AAAI Spring Symposium on Ontological Engineering*, 1997.

50. Philippe Torres and Pierre Lopez. On not-first/not-last conditions in disjunctive scheduling. *European Journal of Operational Research*, 127(2):332–343, December 2000.

51. Peter van Beek and Xinguang Chen. Cplan: A constraint programming approach to planning. In *AAAI '99/IAAI '99: Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, pages 585–590, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.

52. Vincent Vidal and Hector Geffner. Branching and pruning: An optimal temporal pocl planner based on constraint programming. *Artificial Intelligence*, 170(3):298–335, 2006.

53. Petr Vilím. Edge finding filtering algorithm for discrete cumulative resources in $O(kn \log n)$. In *CP'09: Proceedings of the 15th international conference on Principles and practice of constraint programming*, pages 802–816, Berlin, Heidelberg, 2009. Springer-Verlag.

54. Petr Vilím, Roman Barták, and Ondřej Čepek. Unary resource constraint with optional activities. In *Principles and Pracitce of Constraint Programming - CP 2004, Toronto, Canada*, 2004.

55. Daniel S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.

56. Cushing William and Smith David E. The perils of discrete resource models. In *ICAPS Workshop onInternational Planning Competition: Past, Present and Future*, 2007.

57. Steven A. Wolfman and Daniel S. Weld. The LPSAT engine and its application to resource planning. In Thomas Dean, editor, *IJCAI*, pages 310–317. Morgan Kaufmann, 1999.