Complexity of State-Variable Planning under Structural Restrictions

Peter Jonsson

May 5, 1995

ABSTRACT

Computationally tractable planning problems reported in the literature have almost exclusively been defined by syntactical restrictions. To better exploit the inherent structure in problems, it is probably necessary to study also structural restrictions on the underlying state-transition graph. Such restrictions are typically hard to test since this graph is of exponential size. We propose an intermediate approach, using a state variable model for planning and defining restrictions on the state-transition graph for each state variable in isolation. We identify such restrictions which are tractable to test and we present a planning algorithm which is correct and runs in polynomial time under these restrictions.

Moreover, we present an exhaustive map over the complexity results for planning under all combinations of four previously studied syntactical restrictions and our five new structural restrictions. This complexity map considers both the bounded and unbounded plan generation problem. Finally, we extend a provably correct, polynomial-time planner to plan for a miniature assembly line, which assembles toy cars. Although somewhat limited, this process has many similarities with real industrial processes.

The research of the author was sponsored by the Swedish Research Council for the Engineering Sciences (TFR) under grant Dnr. 93-00291.

Acknowledgements

First of all, I would like to express my thanks to my advisor, Christer Bäckström, for guidance and encouragement. I would also like to thank the two other members of my supervisory group, Erik Sandewall and Jan Małuszyński, for their support.

I would like to thank Inger Klein and Jonas Kvarnström for their contributions to this thesis. Inger Klein provided important ideas on planning algorithms for (semi)realistic industrial processes and Jonas Kvarnström found several errors and mistakes in an earlier version of this thesis.

Finally, I would like to thank my colleagues in RKLLAB and the rest of IDA for providing a stimulating research environment. Special thanks goes to the administrative and technical staff; particularly Lise-Lott Svensson and Leif Finmo.

Contents

Chapter 1

Introduction

This introductory chapter is divided into five sections. Section 1.1 presents, in very general terms, the planning problem in the artificial intelligence (AI) sense. Section 1.2 discusses the complexity aspects of planning and surveys some of the most important results in this area. Section 1.3 discusses the motivations behind the research and Section 1.4 briefly presents the results. Finally, Section 1.5 gives an overview of how this thesis is organised.

1.1 The Planning Problem

Assume that we are given some *world* (or system), such as a room together with the objects in it, an industrial process or the abstract activities going on inside a computer. Furthermore, suppose that this world can be modelled by some kind of *state*, describing what the world looks like at a certain point in time, and that there is a set of *actions* that can be performed in this world, triggering transitions between states in the world. Typically an action has some *effects*, modelling how the current state of the world would change if this action were to be executed. Similarily, actions are equipped with *preconditions*, which expresse what conditions must hold in a state for the action to be executable there. Then, the planning problem[1] is: Given an *initial* state, describing the state of the world when we will start executing the plan and a *goal* state, describing what we want the world to look like, find a sequence of actions that will change the state of the world from the initial state to the goal state. Observe that there are certain restrictions that

[1]Sometimes referred to as the *action* planning problem in order to distinguish it from path planning [Schwartz and Sharir, 1990].

this sequence of actions must satisfy. First of all, the first action must be executable in the initial state, *ie.* has its preconditions satisfied there. The successful application of this action in the initial state results in a new state. Then the second action in the sequence must be executable in this new state, resulting in yet a new state and so forth. Finally, the state resulting after executing the last action in the sequence must be the goal state. The actions are usually described using *operators*, which are action templates that can be instantiated into specific action occurrences. An action is then a specific instance of an operator.

The world states are typically modelled by formulae in some logic. This leads to a basic distinction made in planning between *first-order* and *propositional* planning formalisms. With a first-order formalism, we mean a formalism where states are described as formulae in some first-order logic. Often, we are not allowed to use any first-order formula in our world descriptions. A typical restriction is that the formulae are required to be atomic and ground, and this kind of restricted formalism is referred to as *propositional.* Instead of propositional atoms, multi-valued *state variables* are sometimes used. In such formalisms, each variable can take its values from some pre-defined, finite domain. Clearly, this is a more flexible approach than the propositional one where each atom can take only one of two values, namely true or false.

Traditionally, there has been two major approaches to representing and solving planning problems. We will refer to these as *operator-search* planning and *deductive* planning respectively. In operator-search planning, world states and operators are represented separately and an operator models how a world state is changed if the operator is executed there. As indicated by the name, operator-search planning uses some search-based method to find a solution, *ie.* a sequence of actions transforming the initial state into the goal state. Typical planning systems belonging to this class are STRIPS [Fikes and Nilsson, 1971], TWEAK [Chapman, 1987] and SNLP [McAllester and Rosenblitt, 1991].

In deductive planning, both the world and the operators are axiomatized in some suitable logic. Planning is done by attempting to prove that a state satisfying the goal can be reached from the initial state, using the available operators. If the resulting proof is constructive, the actual solution is also generated as a side-effect. Many logic formalizations of planning have been proposed in the literature, the perhaps most well-known is the *situation calculus* by McCarthy and Hayes [1969]. However, few of these formalisms have led to implemented planners or even a planning algorithm. An early exception is the QA3 system [Green, 1969] which is a resolutionbased planner for the situation calculus. A more recent example is the LLP system by Biundo *et al.* [1992] which builds on an interval-based temporal logic.

1.2 Complexity of Planning

1.2.1 Complexity Results

As a result of the growing interest in the computational aspects of planning, it is now well-known that planning is computationally hard in the general case. Propositional planning is PSPACE-complete [Bylander, 1994] in the unrestricted case. Even worse, there exists propositional planning problems with exponentially sized minimal solutions [Bäckström and Klein, 1991b], thus making the generation problem for propositional planning provably intractable. A number of restricted cases have been analysed by Bylander [1994] and Erol *et al* [1992] and their complexity range from PSPACE-complete to, in rare cases, tractable. These restricted cases are based on simple syntactic restrictions on operators, such as bounding the number of pre- and post-conditions.

Using multi-valued state-variables instead of propositional atoms cannot, of course, simplify the problem. However, it seems that state-variable formalisms can facilitate the search for restrictions leading to tractability. A line of research concerning the complexity of state-variable planning was initiated by Bäckström and Klein in [Bäckström and Klein, 1990]. This paper was the first in a row of papers [Bäckström and Klein, 1991a, Bäckström and Klein, 1991b, Bäckström, 1992b, Bäckström and Nebel, 1993] on the complexity of state-variable planning in the SAS$^+$ formalism. By starting with a simple, severely restricted problem and successively removing and replacing restrictions, several tractable subproblems could be identified.

Naturally, the complexity situation is even worse for first-order planning. The first result concerning the complexity of first-order planning was presented by Chapman [1987] who showed that first-order planning is undecidable. The proof made use of a planning problem with an infinite number of variables, which is hardly a realistic assumption. However, Erol *et. al* [1992] cleaned up this result by proving (with a less contrived proof) that first-order planning is undecidable if there are function symbols or infinitely many constants in the language. If no function symbols and only finite domains are allowed, first-order planning is decidable with a complexity ranging from EXPSPACE-complete to polynomial, depending on further restrictions.

1.2.2 Approaches to Reduce Complexity

From a complexity point of view, it is impossible to make an inherently intractable problem tractable. So, if we encounter an intractable problem, we have to sacrifice something in order to solve the problem efficiently. We can identify at least three major methods that have been proposed for attacking intractability.

- Concentrate on approximate solutions.
- Consider alternative definitions of tractability.
- Exploit restrictions in the problem.

Of course, there are different pros and cons of these three approaches. In the case of approximate solutions, it is disputable what an (reasonable) approximate solution to a planning problem is. Is it, for example, reasonable to relax the validity requirement for plans? If invalid plans are generated only infrequently, then this may be an acceptable solution. In other cases, this is completely unacceptable. Is it reasonable to relax some other quality aspect of the generated plan, such as its length? Once again, the answer must be "maybe", depending on the context. If we are working within a time-critical domain, then the answer could be no, even for plans that are moderately longer than the minimal plan. In other domains, though, we could be willing to accept even extremely lengthy plan, simply because there are no alternatives. Interesting results in this direction have been shown for the *blocks-world* domain. It has, for example, been proven that the blocks-world minimal plan generation problem is NP-complete, whereas non-minimal blocks-world planning can be carried out in polynomial time [Gupta and Nau, 1991]. A similar result holds for the SAS$^+$-US plan generation problem [Bäckström and Nebel, 1993].

The most obvious alternative definition of tractability is to settle for tractability in the average case. One problem with this approach is to define models for realistic average cases. Although attempts to define average-case planning problems has been made [Bylander, 1993], there seems to be much work to do before we have an acceptable concept of average-case planning complexity. Even if we had one, this redefinition of tractability is, of course, not sufficient for all applications. An other possibility is to consider near-polynomial classes, for instance those found in Johnson *et. al* [1988] or the class of *pseudo-polynomial* problems [Garey and Johnson, 1979].

If a problem is intractable in general, it may nevertheless have interesting subcases that are tractable. Most probably, many application problems we try to solve are more restricted than the formal problem we have used to model them. Hence, if we are fortunate, the problem may fit in some tractable subclass of the formal problem. The major drawback of this method is that it can be hard to discover inherent restrictions in the problems and develop ways of exploiting them. Since there are so many examples of this approach in the literature, we devote the entire next section to this topic.

1.2.3 Exploiting Restrictions to Reduce Complexity

In this section, we survey and comment on some approaches to reducing the complexity of planning by investigating inherent restrictions in the problems. It has to be noted that this list of methods is by no means exhaustive; several other techniques have been suggested.

Korf [1987] has analysed how different properties concerning subgoals affect the complexity of planning. He identified a number of properties such as subgoal independence and subgoal serializability and showed that the search space can be drastically reduced when such properties holds. Even though Korf managed to show impressing decreases in the size of the search space in some cases, there are drawbacks with his methods.

- The properties reduce the search needed, but they do not guarantee tractability. (Which, of course, is not surprising. We can never hope for getting away from the inherent complexity of problems.)
- It is hard to test some of the proposed properties. Bylander [1992] has shown that deciding whether an instance has serializable subgoals or not is PSPACE-complete. Probably, we are not willing to spend that much effort, especially since we are not guaranteed tractability, anyway.

Hierarchical Task Network (HTN) planning is a technique introduced by Sacerdoti [1975] in his NOAH system. A partial plan is a network consisting of two types of objects, operators and tasks. A task can be expanded into a subnetwork of new tasks and/or operators according to predefined expansion rules. The planning process starts with a single task specifying the goal, expands this task into a network according to the rules and then recursively expands the tasks in this network until it only contains operators. Since the number of expansions associated with a task can be expected to be much smaller than the whole set of operators the planner will typically need to search a smaller search space. Obviously, this technique is more domain-specific than the others since the expansions have to be predefined. It is interesting to note that this method underlies several of the most successful planners in practice, *eg.* NonLin [Tate, 1977], O-Plan [Currie and Tate, 1991] and Sipe [Wilkins, 1988]. Although seemingly very efficient in practice, not much theoretical analysis has been done for this method. Yang [1990] and Erol *et al.* [1994a, 1994b] are noticeable exceptions.

An approach resembling HTN planning is *macro operator* planning. The idea is to lump together operators into operator sequences, referred to as macro operators, and then use these macro operators as ordinary operators in the planning process. The macro operator approach has been the predominant one when combining planning with learning techniques, and the first example can be found in Fikes *et al.* [1972] as an extension of Strips. When a plan has been found for achieving some goal, it is stored in a table. Since we know the combined pre-conditions and the combined effects of this operator sequence, it is natural to view it as a macro operator. By incorporating these macro operators into the operator reportoire of the planner, the planner learns more and more macro operators which, on the long term, may decrease the search needed by the planner for new problems. Korf [1987] has presented some complexity figures for planning with macro operators.

Another approach is *hierarchical abstraction.* The idea is to first plan for the most important goals and then successively refine the plan to also achieve the less important goals. This technique was first employed in the AbStrips system [Sacerdoti, 1974]. It was shown that AbStrips was more efficient than Strips on a number of test examples, but no formal analysis was done of how and if the efficiency was affected. Other well-known systems using hierarchical abstraction are Pablo [Christensen, 1990], Abtweak [Yang and Tenenberg, 1990] and Prodigy [Knoblock, 1994]. Observe that the idea of hierarchical abstraction can be combined with both standard operator search and HTN planning, as well as with deductive planning.

Hierarchical abstraction has been analyzed by, for example, Knoblock [Knoblock, 1991b]. Knoblock tried to characterize under which restrictions an exponential reduction of the search space is possible. Using b for the branching factor and l for the length of the shortest solution, he found that the worst case size of the search space can be reduced from $O(b^l)$ to $O(l)$ under ideal conditions, leading to an exponential speed-up of the planning process. Unfortunately, these restrictions are not restrictions on the problem instance *per se*, but rather on the planning process. This makes them extremely hard to test in practice. Unfortunately, hierarchical abstraction is not always harmless, it may also lead to exponential slow-down as has been shown in Bäckström and Jonsson [1995].

1.3 Motivations

The topic of this thesis is state-variable planning that is tractable under certain restrictions on the problem structure. Below, we argue why we believe this to be a plausible approach to planning.

1.3.1 Why Tractability?

If we intend to apply results from (theoretical) computer science to real-world problems, then tractability is a fundamental issue. Obviously, if a problem is not tractable, we cannot hope for solving an arbitrary problem instance in reasonable time with any type of computer. Hence, if we want to use computers for solving planning problems, we must demand that the problems we attack are tractable. This does not seem to be the common point of view in the AI community, although there has been an increasing understanding of complexity issues over the last few years. Common arguments why tractability is not of any significant importance are:

- The intractable cases are rare, pathological cases which do not occur in practice.
- Since planning is computationally hard (even undecidable) in the general case, AI planning will never succeed anyway.

We will try to meet these arguments in turn. If the hard cases do not occur in real problems, then we have not charachterized the problem properly. In other words, the real problem is a tractable subproblem of the problem we have characterized. Levesque [1988] has stated this argument as follows:

> If the only problematic cases are the ones that do not seem to occur in practice...we can simply decide to eliminate them from consideration...But this does not eliminate our concern with extreme cases; it merely changes what cases we consider to be extreme.

One objection to this view is that eliminating the problematic cases can be an extremely non-trivial task. In fact, this is what finding tractable subcases of an intractable problem is all about.

Another answer to the first argument is that, in certain cases, we must be absolutely sure that these pathological cases do not occur in practice. Consider for example a nuclear power plant with a beginning core melt-down. If this would be one of the pathological cases for the control system of the plant, causing it to spend several years to solve the problem, we would be in deep trouble. Predicting all the infrequent anomalies a real-world system might encounter is obviously an insurmountable task. As a consequence, we will never be able to construct reliable complex systems with "pathological cases never occur" as an underlying assumption.

The second argument leads to an interesting observation. Chapman [1987, p. 350] writes about his TWEAK planner:

> The restricitions on action representation make TWEAK almost useless as a real-world planner.

However, on page 344 of the same publication, he writes:

> Any Turing machine with its input can be encoded as a planning problem in the TWEAK representation.

Consequently, TWEAK can solve any problem that a computer can solve (assuming Church's hypothesis). Combining these two quotations leads to the conclusion that real-world planning problems cannot be solved by any computer. Most likely, this is not the conclusion Chapman intended. A more reasonable conclusion is that TWEAK is too restricted in some aspects (the world-modelling power), but at the same time being too unrestricted in other aspects (the computational power).

1.3.2 Why State-Variables?

As has been shown by Bäckström [1992a, 1995], multi-valued, discrete state variables can be simulated by propositional atoms in traditional planning formalisms. In some cases, the methods for simulating state-variables are rather intricate and can considerably decrease the performance of the planner [Bäckström, 1994]. The other way round, simulating propositional planning with state-variables, is of course trivial (simply by using two-valued state variables).

Hence, there does not seem to be any serious disadvantages in using state-variables instead of propositional atoms. In fact, we believe that there are several good reasons for using state-variables instead of propositional atoms. First, many applications (especially in *sequential control*[2]) are more naturally described in this way and using state-variables seems to be conceptually more appealing to automatic control researchers. Secondly, we believe that the use of state-variables has considerably facilitated the search for tractable subcases of planning. To a large extent, the results in this thesis builds on the possible transitions within a state-variable. Clearly, a propositional atom (which we can view as a two-valued state-variable) would not allow for much structure of this kind to exploit. Thirdly, it will be a smaller step to generalize the state variables to structured domains such as the integers or the set of integer-valued intervals. As a first step in this direction, a state-variable formalism with interval-valued state variables have been presented in Bäckström [1988].

1.3.3 Why Structural Restrictions?

Many planning problems in manufacturing and process industry are believed to be highly structured, thus allowing for efficient planning if exploiting this structure. Recent research into assembly planning [Almgren, 1994, Márkus and Váncza, 1994] confirms this observation. However, a 'blind' domain-independent planner will most likely go on tour in an exponential search space even for tractable problems. Although heuristics may help a lot, they are often not based on a sufficiently thorough understanding of the underlying problem structure to guarantee efficiency and correctness. Further, we believe that if having a sufficiently deep understanding of the problem structure, it is better to use more formal methods than heuristics.

As we have seen, some tractability results have been shown for propositional planning. Almost all of these results are based on essentially syntactic restrictions on the set of operators. Syntactic restrictions are very appealing to study since they are typically easy to define and not very costly to test. However, to gain any deeper insight into what makes planning problems hard and easy respectively probably require that we study the structure of the problem, in particular the state-transition graph.[3] To some extent, syntac-

[2] Sequential control is a subfield of *automatic control*, dealing with combinatorial problems that often resemble planning problems.

[3] The state-transition graph is a directed graph $G = \langle V, E \rangle$ with the states of the system as nodes and $(v, w) \in E$ iff there exists a operator that transforms state v to state w.

tic restrictions allow us to do this since they undoubtedly have implications for what this graph looks like. However, their value for this purpose seems somewhat limited since many properties that are easy to express as structural restrictions would require horrendous syntactical equivalents. Putting explicit restrictions on the state-transition graph must be done with great care, however, since this graph is typically of exponential size in the size of the planning problem instance, making it extremely costly to test arbitrary properties.

Although studying structural restrictions seems to be profitable in the context of planning, not much research on structural restrictions seems to be reported in the literature. Structural restrictions in state-variable formalisms have been studied in order to decrease search complexity by Korf [1987], as was pointed out earlier. In his ALPINE system, Knoblock [Knoblock, 1991a] uses a technique that exploits inherent problem structure to automatically construct abstraction hierarchies. Mädler [1992] extends Sacerdoti's [1974] essentially syntactic state abstraction technique to *structural abstraction*, identifying bottle-neck states (*needle's eyes*) in the state-transition graph for a state-variable formalism. Smith and Peot [1993] use an *operator graph* for preprocessing planning problem instances, identifying potential threats that can be safely postponed during planning—thus, pruning the search tree. The operator graph can be viewed as an abstraction of the full state-transition graph, containing all the information relevant to analysing threats.

1.4 Contributions

This thesis is the compilation and adaption of two papers which have previously appeared in conference proceedings together with one (unpublished) research report. Below, we briefly describe the contents of these papers and, thus, roughly describe the contents of this thesis.

Checking structural properties of the state-transition graph is, as we have pointed out, probably very costly. Hence, we have taken an intermediate approach to the problem of studying the state-transition graph in Jonsson and Bäckström [1994c, 1994d]. Using a state-variable formalism we defined restrictions not on the whole state-transition graph, but on the domain-transition graph for each state variable in isolation. Since these are rather moderately-sized, in comparison to the full state-transition graph, properties on these need not be expensive to test. Although not being a substitute for restrictions on the whole state-transition graph, it turned out that many interesting and useful properties of this graph can be indirectly exploited. In particular, we identified three structural restrictions which together make planning tractable and properly generalize the tractable problems we have previously defined using syntactical restrictions [Bäckström and Nebel, 1993]. Despite being structural, these restrictions can be tested in polynomial time. We have presented a polynomial-time, sound and complete algorithm for generating minimal plans under the new structural restrictions.

After having isolated syntactical and structural restrictions that, in certain combinations, lead to tractability, it is natural to ask for the computational complexity of *all* combinations of these restrictions. For the syntactical restrictions P, U, B and S on SAS$^+$ instances, this problem was addressed by Bäckström and Nebel [1993] who provided a map over the complexity of planning for all combinations of these restrictions. We have then taken the new structural restrictions into account, providing an almost exhaustive map over the complexity of both bounded and unbounded plan generation for all combinations of the restrictions, considering also mixed structural and syntactical restrictions in Jonsson and Bäckström [1994a, 1994b]. These results have been extended in this thesis resulting in an exhaustive map over the complexity for plan generation under nine different restrictions, including those previously mentioned.

For tractable planning problems, the question of practical relevance immedieately arises. In Klein *et. al* [1995], we report how to plan for the LEGO[4] car factory [Strömberg, 1991]. This is a realistic miniature version of real industrial processes in many respects. We show that by exploiting the inherent structure of the problem, we can model it and use a polynomial time planning algorithm to solve it. More precisely, we use a simple and provably correct extension to one of our previosly presented tractable planning algorithms.

1.5 Organization of the Thesis

The remainder of this thesis is organized in the following way.

Chapter 2 defines the SAS$^+$ state variable formalism and discusses the main differences between propositional Strips and SAS$^+$. The formalism is illustrated using an example based on a simple idea of how an automatic cappucino brewer could work.

[4] LEGO is a trademark of the LEGO company.

Chapter 3 defines the restrictions that will be considered in this thesis. Four syntactical restrictions (P, U, B, S) are presented together with five structural restrictions (I, A^-, A, A^+, O).

Chapter 4 presents an algorithm for state variable planning under the restrictions I, A and O. A small workshop example is used for illustrating how the algorithm works.

Chapter 5 contains the correctness proofs and a complexity analysis for the planning algorithm in chapter 4.

Chapter 6 provides an exhaustive map of complexity results for planning under all nine restrictions from chapter 3. This map considers both optimal and non-optimal plan generation.

Chapter 7 shows how a slightly modified version of the planning algorithm can be used for planning for a semi-realistic miniature version of an industrial process. Although quite limited, we believe that this process has many features in common with real industrial processes.

Finally, some notes on terminology and notation. Regarding complexity theory we largely follows the terminology in standard references such as Baase [1988], Garey and Johnson [1979] and Johnson [1990]. It may be worth noticing that we follow the notation in [Baase, 1988] and view $O(f(n)$ as the *set* of functions growing at most as fast as $f(n)$. Hence, we may write, *eg.*, $n \in O(n^3)$ and $O(n) \subseteq O(n^2)$.

Another point worth noticing is that in complexity theory a problem is a general question, having a number of specific instances. For example, the *satisfiability* problem is to answer the general question as to whether some arbitrary set of formulae has a model. Deciding whether some specific set of formulae has a model is an *instance* of the satisfiability problem.

Finally, when illustrating plans in figures we denote ordering constraints by arrows and we typically do not show arrows representing transitive orderings only.

Chapter 2

Basic Formalism

The SAS$^+$ formalism is in principle a slight variation on the propositional STRIPS formalism [Fikes and Nilsson, 1971]. This chapter briefly recasts the main differences between the two formalisms and gives a formal definition of the SAS$^+$ formalism. The reader is referred to previous publications [Bäckström and Klein, 1991a, 1991b; Bäckström, 1992a, 1992b] for further background, motivation and examples.

2.1 World Modelling in the SAS$^+$ Formalism

There are mainly two details that differ between the SAS$^+$ formalism and the STRIPS formalism. Instead of propositional atoms we use *multi-valued state variables* and instead of using only pre- and post-conditions (or add- and delete-lists) for operators we also use a *prevail-condition*, which is a special kind of pre-condition.

Each state variable has some discrete domain of mutually exclusive *defined* values and in addition to these we also allow it to take on the *undefined* value. The undefined value is interpreted as 'don't care'—that is, the value is unknown or does not matter. A *total (world) state* assigns defined values to all state variables, while a *partial (world) state* may also assign the undefined value. We will often use the term *state* when it is clear from the context or does not matter whether the state is partial or total.

The main difference in contrast to the traditional STRIPS modelling is that the pre-condition is split into two conditions, depending on whether the variables are changed by the operator or only required to have some specific value. The 'behaviour' of an operator is modeled by its *pre-*, *post-* and *prevail-conditions*—all three being partial states. The post-condition of an operator expresses which state variables it changes and what values these variables will have after executing the operator, if executed successfully. The pre-condition specifies which values these changed variables must have before the operator is executed. The prevail-condition specifies which of the unchanged variables must have some specific value before the execution of the operator and what these values are. Both the pre- and prevail-condition of the operator must be satisfied for the operator to execute successfully.

Definition 2.1.1 An instance of the SAS$^+$ planning problem is given by a quadruple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ with components defined as follows:

- $\mathcal{V} = \{v_1, \ldots, v_m\}$ is a set of **state variables**. Each variable $v \in \mathcal{V}$ has an associated **domain** $\mathcal{D}_v$, which implicitly defines an **extended domain** $\mathcal{D}_v^+ = \mathcal{D}_v \cup \{\mathsf{u}\}$, where u denotes the **undefined value**. Further, the **total state space** $\mathcal{S} = \mathcal{D}_{v_1} \times \ldots \times \mathcal{D}_{v_m}$ and the **partial state space** $\mathcal{S}^+ = \mathcal{D}_{v_1}^+ \times \ldots \times \mathcal{D}_{v_m}^+$ are implicitly defined. Let $s[v]$ denote the value of the variable v in a state s.

- $\mathcal{O}$ is a set of **operators** of the form $\langle \mathsf{pre}, \mathsf{post}, \mathsf{prv}, \mathsf{name} \rangle$, where $\mathsf{pre}, \mathsf{post}, \mathsf{prv} \in \mathcal{S}^+$ denote the **pre-, post-** and **prevail-condition** respectively and name is a unique identifier. If $o = \langle \mathsf{pre}, \mathsf{post}, \mathsf{prv}, \mathsf{name} \rangle$ is a SAS$^+$ operator, we write $\mathsf{pre}(o)$, $\mathsf{post}(o)$, $\mathsf{prv}(o)$ and $\mathsf{name}(o)$ to denote pre, post, prv and name respectively. $\mathcal{O}$ is subject to the following two restrictions

  (R1) for all $o \in \mathcal{O}$ and $v \in \mathcal{V}$ if $\mathsf{pre}(o)[v] \neq \mathsf{u}$, then $\mathsf{pre}(o)[v] \neq \mathsf{post}(o)[v] \neq \mathsf{u}$,

  (R2) for all $o \in \mathcal{O}$ and $v \in \mathcal{V}$, $\mathsf{post}(o)[v] = \mathsf{u}$ or $\mathsf{prv}(o)[v] = \mathsf{u}$.

- $s_0 \in \mathcal{S}^+$ and $s_* \in \mathcal{S}^+$ denote the **initial state** and **goal state** respectively.

The unique identifier for each operator is not technically necessary but it will simplify some of the forthcoming proofs. Restriction R1 essentially says that a state variable can never be made undefined, once made defined by some operator. Restriction R2 says that the pre- and prevail-conditions of an operator must never define the same variable. Having the space $\mathcal{S}^+$, we need an ordering that captures the idea of one state being more well-defined than another state:

Definition 2.1.2 Let $s \sqsubseteq t$ hold iff the value s is subsumed (or satisfied) by value t, *ie.* if $s = \mathsf{u}$ or $s = t$. We extend this notion to whole states, defining

$$s \sqsubseteq t \text{ iff for all } v \in \mathcal{V},\ s[v] \sqsubseteq t[v]$$

which defines a partial order $\langle \mathcal{S}^+, \sqsubseteq \rangle$ with bottom element $\perp = \langle \mathsf{u}, \ldots, \mathsf{u} \rangle$. The operation $\sqcup$ (least upper bound) is defined in the usual way on the partial order $\langle \mathcal{S}^+, \sqsubseteq \rangle$.

We continue by defining plans.

Definition 2.1.3 Given a set of operators $\mathcal{O}$, we define the set of all operator sequences over $\mathcal{O}$ as $Seqs(\mathcal{O}) = \{\langle\rangle\} \cup \{\langle o \rangle;\omega \mid o \in \mathcal{O} \text{ and } \omega \in Seqs(\mathcal{O})\}$, where ; is the sequence concatenation operator. The members of $Seqs(\mathcal{O})$ are called **plans**.

We define some functions that operate on operator sequences.

Definition 2.1.4 Let $\omega = \langle o_1, \ldots, o_k \rangle \in Seqs(\mathcal{O})$. Then, $First(\omega) = o_1$, $Last(\omega) = o_k$ and $Rest(\omega) = \langle o_2, \ldots, o_k \rangle$. If $\omega = \langle\rangle$, then $First(\omega) = \langle\rangle$, $Last(\omega) = \langle\rangle$ and $Rest(\omega) = \langle\rangle$.

Next, we define the result of executing an operator sequence:

Definition 2.1.5 Given two states $s, t \in \mathcal{S}^+$, we define for all $v \in \mathcal{V}$,

$$(s \oplus t)[v] = \begin{cases} t[v] & \text{if } t[v] \neq \mathsf{u}, \\ s[v] & \text{otherwise.} \end{cases}$$

The function *result* gives the state resulting from executing an operator sequence and is defined recursively as

$$\begin{aligned} result(s, \langle\rangle) &= s, \\ result(s, \omega;\langle o \rangle) &= \begin{cases} t & \text{if } (\mathsf{pre}(o) \sqcup \mathsf{prv}(o)) \sqsubseteq result(s, \omega), \\ \perp & \text{otherwise}. \end{cases} \end{aligned}$$

where $t = result(s, \omega) \oplus \mathsf{post}(o)$.

This definition of the *result* function 'solves' the *frame problem* by employing the STRIPS assumption (Fikes and Nilsson [1971]), which is sufficient in this restricted formalism. Furthermore, we take the safe and cautious approach that anything can happen if an action occurs when its pre- and prevail-conditions are not satisfied. It would be somewhat more reasonable to make undefined only those state variables that are affected by the action. However, this is not important for our purposes since we will only be concerned with generating correct plans. It should be noted, though, that our definition is in contrast to those used by, for instance, Dean and Boddy [1988] and Bylander [1991]. They use the seemingly more unsafe and optimistic model that an action has no effect at all if its pre-condition is not satisified. Hence, their approach makes much stronger assumptions about the execution model than ours does. We can now define when an operator sequence solves a problem instance.

Definition 2.1.6 The ternary relation $\mathit{Valid} \subseteq \mathit{Seqs}(\mathcal{O}) \times \mathcal{S}^+ \times \mathcal{S}^+$ is defined recursively s.t. for arbitrary operator sequence $\langle o_1, \ldots, o_n \rangle \in \mathit{Seqs}(\mathcal{O})$ and arbitrary states $s, t \in \mathcal{S}^+$, $\mathit{Valid}(\langle o_1, \ldots, o_n \rangle, s, t)$ iff either

1. $n = 0$ and $t \sqsubseteq s$ or
2. $n > 0$, $\mathsf{pre}(o_1) \sqcup \mathsf{prv}(o_1) \sqsubseteq s$ and $\mathit{Valid}(\langle o_2, \ldots, o_n \rangle, (s \oplus \mathsf{post}(o_1)), t)$.

A plan $\langle o_1, \ldots, o_n \rangle \in \mathit{Seqs}(\mathcal{O})$ **solves** Π iff $\mathit{Valid}(\langle o_1, \ldots, o_n \rangle, s_0, s_*)$.

To define partially ordered plans, we must introduce the concept of *actions*, *ie.* instances of operators. Given an action a, $\mathit{type}(a)$ denotes the operator that a instantiates. Furthermore, given a set of actions $\mathcal{A}$, we define $\mathit{type}(\mathcal{A}) = \{\mathit{type}(a) \mid a \in \mathcal{A}\}$ and given an action sequence $\alpha = \langle a_1, \ldots, a_n \rangle$ of actions, $\mathit{type}(\alpha)$ denotes the operator sequence $\langle \mathit{type}(a_1), \ldots, \mathit{type}(a_n) \rangle$. The functions *First*, *Last* and *Rest* operate on action sequences in the same way as they operate on operator sequences.

Definition 2.1.7 A **partial-order plan** is a tuple $\langle \mathcal{A}, \prec \rangle$ where $\mathcal{A}$ is a set of actions and $\prec$ is a strict partial order on $\mathcal{A}$. A partial-order plan $\langle \mathcal{A}, \prec \rangle$ solves a SAS$^+$ instance Π iff $\langle \mathit{type}(a_1), \ldots, \mathit{type}(a_n) \rangle$ solves Π for each topological sort $\langle a_1, \ldots, a_n \rangle$ of $\langle \mathcal{A}, \prec \rangle$.

The set of all topological sorts of a plan Δ is denoted $CS(\Delta)$. Given an action sequence α and an action a, we often write $(\alpha; a)$ as an abbreviation for $\alpha; \langle a \rangle$. For partial orders, we need the following definitions.

Definition 2.1.8 Given a partial order $<$ on a set S we make the following definitions:

1. $<^+$ is the **transitive closure** of $<$ defined in the usual way, and

2. $<^-$ denotes the **reduction** of $<$ defined as the minimal $<_0 \subseteq <$ such that $<_0^+ = <^+$.

Definition 2.1.9 Let $\mathcal{O}$ be a set of operators over some set $\mathcal{V}$ of state variables and let $v \in \mathcal{V}$. The function *Affects* : $\mathcal{V} \times \mathcal{O} \rightarrow 2^{\mathcal{O}}$ is defined as:

$$\mathit{Affects}(v, \mathcal{O}) = \{o \in \mathcal{O} \mid \mathsf{post}(o)[v] \neq \mathsf{u}\}$$

$\mathit{Affects}(v, \mathcal{O})$ denotes the set of operators in $\mathcal{O}$ which change the value of variable v. Clearly, *Affects* can be generalized to sets of actions in the obvious way.

Definition 2.1.10 Let $\Delta = \langle \mathcal{A}, \prec \rangle$ be a partially ordered plan over some set of operators $\mathcal{O}$ (defined over some set of state variables $\mathcal{V}$) and let $v \in \mathcal{V}$. The function *Paths* : $\mathcal{V} \times (2^{\mathcal{A}} \times (2^{\mathcal{A} \times \mathcal{A}})) \rightarrow 2^{\mathit{Seqs}(\mathcal{O})}$ is defined as:

$$\mathit{Paths}(v, \langle \mathcal{A}, \prec \rangle) = CS(\langle \mathit{Affects}(v, \mathcal{A}), \prec' \rangle$$

where $\prec'$ is $\prec$ restricted to $\mathit{Affects}(v, \mathcal{A})$.

$\mathit{Paths}(v, \langle \mathcal{A}, \prec \rangle)$ denotes the set of sequences over $\mathit{Affects}(v, \mathcal{A})$ which are consistent with $\prec$. It should be noted that $\mathit{Paths}(v, \mathcal{A}, \prec)$ may contain more than one sequence[1]. For totally ordered action sequences ω, we sometimes abuse the notation and write $\mathit{Paths}(v, \omega)$ instead of $\mathit{Paths}(v, \langle \mathcal{A}, \prec \rangle)$ where $\mathcal{A}$ is the actions in ω and $\prec$ is the total ordering of $\mathcal{A}$.

Given an action sequence $\alpha = \langle a_1, \ldots, a_n \rangle$ the notation $\alpha \backslash k$ denotes the action sequence $\langle a_1, \ldots, a_{k-1} \rangle$ and α / k denotes the action sequence $\langle a_1, \ldots, a_k \rangle$, which can be illustrated as follows

$$\underbrace{\overbrace{\langle a_1, \ldots, a_{k-1}}^{\alpha \backslash k}, a_k}_{\alpha / k}, a_{k+1}, \ldots, a_n \rangle.$$

Finally, we define a restricted variant of the SAS$^+$ formalism.

Definition 2.1.11 The SAS* problem is the SAS$^+$ problem restricted to instances $\langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ satisfying

[1] For a more thorough discussion of this topic, see Bäckström [1992a].

1. $s_0 \in \mathcal{S}$

2. for every operator $o \in \mathcal{O}$ and variable $v \in \mathcal{V}$, if $\mathsf{pre}(o)[v] = \mathsf{u}$, then $\mathsf{post}(o)[v] = \mathsf{u}$.

The reasons for introducing the SAS* formalism will be made clear in chapter 6.

2.2 An Example: The Cappucino Brewer

Below we will illustrate the SAS+ formalism using an example based on a simple idea of how an automatic cappucino brewer could work. This example is chosen for illustrative purpose only and is taken from a forthcoming paper by Bäckström and Nebel [1996]. The brewer (see Figure 2.1) consists of a steam container, a coffee supply, a coffee filter and a milk steamer. In order to brew a cup of espresso there must be fresh coffee in the filter and steam in the steam container. If the milk in the milk steamer has been steamed to foam we might also top the espresso with milk foam, thus getting a cappucino (ignoring sugar and cocoa powder). We assume there is always coffee in the coffee supply and always milk or milk foam in the milk steamer. The state variables and their domains are described in Table 2.1.

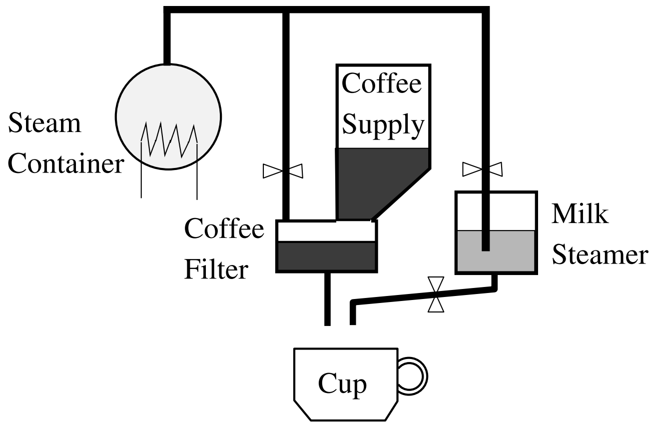


Figure 2.1: A simplified cappucino brewer

Index	Description	Domain
1	Steam Pressure	$\{High, Low\}$
2	Coffee in filter	$\{Fresh, Used, None\}$
3	Cup content	$\{Nothing, Espresso, Cappucino\}$
4	Milk steamed	$\{True, False\}$

Table 2.1: State variables for the cappucino brewer

States are written as four-tuples of state variable values. For instance,

$$\langle H, F, \mathsf{u}, F \rangle$$

denotes the state where the steam pressure is high (*ie.*, sufficiently high for operation), there is fresh coffee in the filter, we do not know what is in the cup (or if there is anything at all in it) and there is milk, not foam, in the milk steamer.

We further assume that the set of operators in Table 2.2 is available. We have assumed that the operator heat-steam heats the steam (or water) in the steam container until reaching operational pressure, whatever pressure (or water temperature) it starts from. Similarly, the operator steam-milk can be executed even if there is already foam in the milk steamer. The operator fill-filter dispenses fresh coffee into the coffee filter and can be executed only if the filter is empty. To the contrary, the operator empty-filter will empty the coffee filter even if it was previously empty. The operator brew-espresso brews espresso and dispenses it into the cup subject to the condition that the steam pressure is sufficient and there is fresh coffee in the filter. Finally, the operator top-with-foam fills the cup with milk foam.
We assume the initial state is

$$s_0 = \langle \mathsf{u}, N, N, \mathsf{u} \rangle,$$

that is, we know nothing about the steam pressure and we do not know if there is milk or foam in the milk steamer. We do know, though, that both the coffee filter and the cup are empty. To brew an espresso we set the goal

$$s_E = \langle \mathsf{u}, \mathsf{u}, E, \mathsf{u} \rangle$$

and to make a cappucino we set the goal

$$s_C = \langle \mathsf{u}, \mathsf{u}, C, \mathsf{u} \rangle.$$

Action type	pre	post	prv
heat-steam	$\langle u, u, u, u \rangle$	$\langle H, u, u, u \rangle$	$\langle u, u, u, u \rangle$
fill-filter	$\langle \mathsf{u}, N, \mathsf{u}, \mathsf{u} \rangle$	$\langle \mathsf{u}, F, \mathsf{u}, \mathsf{u} \rangle$	$\langle \mathsf{u}, \mathsf{u}, \mathsf{u}, \mathsf{u} \rangle$
empty-filter	$\langle \mathsf{u}, \mathsf{u}, \mathsf{u}, \mathsf{u} \rangle$	$\langle \mathsf{u}, N, \mathsf{u}, \mathsf{u} \rangle$	$\langle \mathsf{u}, \mathsf{u}, \mathsf{u}, \mathsf{u} \rangle$
brew-espresso	$\langle \mathsf{u}, F, N, \mathsf{u} \rangle$	$\langle \mathsf{u}, U, E, \mathsf{u} \rangle$	$\langle H, \mathsf{u}, \mathsf{u}, \mathsf{u} \rangle$
steam-milk	$\langle \mathsf{u}, \mathsf{u}, \mathsf{u}, \mathsf{u} \rangle$	$\langle \mathsf{u}, \mathsf{u}, \mathsf{u}, T \rangle$	$\langle H, \mathsf{u}, \mathsf{u}, \mathsf{u} \rangle$
top-with-foam	$\langle \mathsf{u}, \mathsf{u}, E, \mathsf{u} \rangle$	$\langle \mathsf{u}, \mathsf{u}, C, \mathsf{u} \rangle$	$\langle \mathsf{u}, \mathsf{u}, \mathsf{u}, T \rangle$

Table 2.2: Operators for the cappucino brewer

In both cases we only specify the content of the cup and leave all other state variables undefined ('don't care')—that is, both s_E and s_C are partial goal states.

There are several constraints that must be satisfied for a plan to be valid for making a cup of cappucino. For instance, the filter must be filled with fresh coffee before brewing and it must not be emptied until after brewing, if at all. The filter may even be emptied any number of times without refilling it in between. Since we do not initially know whether the steam pressure is high enough, we further have to ensure this by heating the steam before brewing espresso. We must also ensure that the coffee is brewed (released into the cup) before releasing the milk foam into the cup, otherwise the result will not be a cappucino—but more like a café au lait. Both the operator sequences

$\langle$fill-filter, heat-steam, brew-espresso, steam-milk,
top-with-foam$\rangle$

and

$\langle$heat-steam, fill-filter, empty-filter, fill-filter,
brew-espresso, empty-filter,steam-milk,
empty-filter, top-with-foam$\rangle$

are valid plans for making a cup of cappucino (according to our modelling[2]), while the following plan is not valid,

[2]Our modelling does not take temporal or dynamic aspects into account, for instance.

$\langle$fill-filter, heat-steam, brew-espresso, empty-filter,
top-with-foam$\rangle$

For further examples of problem modelling in the SAS$^+$ formalism appear in other publications [Klein, 1993, Bäckström, 1992a].

Chapter 3

Restrictions

The aim of this chapter is to formally define the restrictions to be considered in this thesis. We present five structural restrictions (I, A^-, A, A^+ and O) together with the four old syntactical restrictions (P, U, B and S). We also develop methods for checking the structural restrictions in polynomial time.

3.1 Syntactical Restrictions

Previously, four restrictions on the SAS^+ planning problem have been identified that, in certain combinations, result in tractability. An instance of the SAS^+ problem is *post-unique* (P) iff no two distinct operators can change the same state variable to the same value and it is *unary* (U) iff each operator changes exactly one state variable. The instance is *binary* (B) iff all state variable domains have exactly two values. Finally, the instance is *single-valued* (S) iff any two operators that both require the same state variable to have some specific value during their respective occurrences must require the *same* defined value. For example, single-valuedness prevents us from having two operators such that one requires a certain room to be lit during its occurrence while the other requires the same room to be dark during its occurrence. This is formally defined as follows:

Definition 3.1.1 An operator o is **unary** iff there is exactly one variable v such that $\mathsf{post}(o)[v] \neq \mathsf{u}$.

Definition 3.1.2 A SAS^+ instance $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ is

(P) Post-unique iff for all $o, o' \in \mathcal{O}$, if $\mathsf{post}(o)[v] = \mathsf{post}(o')[v] \neq \mathsf{u}$ for some $v \in \mathcal{V}$, then $o = o'$;

(U) Unary iff all $o \in \mathcal{O}$ are unary;

(B) Binary iff $|\mathcal{D}_v| = 2$ for all $v \in \mathcal{V}$,

(S) Single-valued iff there exists some state $s \in \mathcal{S}^+$ s.t. $\mathsf{prv}(o) \sqsubseteq s$ for all $o \in \mathcal{O}$.

All these are essentially syntactical restrictions on the problem instance and they are all easy to test in polynomial time [Bäckström, 1992a].

3.2 Structural Restrictions

In this section we will define five structural restrictions (I, A$^-$, A, A$^+$ and O) on the state-transition graph, or, rather, on the domain-transition graphs for the state variables. We first define some underlying concepts. Most of these are straightforward, possibly excepting the set of requestable values, which plays an important role for the planning algorithm. We will use the SAS$^+$ problem instance Σ defined in Example 3.2.1 as an example for demonstrating the various restrictions. For the definitions below, let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ be a SAS$^+$ instance.

Example 3.2.1 $\Sigma = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ where $\mathcal{V} = \{v_1, v_2\}$ and $\mathcal{D}_{v_1}, \mathcal{D}_{v_2}$ and $\mathcal{O}$ are defined in Tables 3.1 and 3.2. For our purposes the exact values of s_0 and s_* do not matter so we leave them unspecified.

In the following we assume an arbitrary SAS$^+$ instance $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$.

Definition 3.2.1 For each $v \in \mathcal{V}$ and $\mathcal{O}' \subseteq \mathcal{O}$, the set $\mathcal{P}_v^{\mathcal{O}'}$ of **prevail-requestable values** for $\mathcal{O}'$ is defined as

$$\mathcal{P}_v^{\mathcal{O}'} = \{\mathsf{prv}(o)[v] \mid o \in \mathcal{O}'\} - \{\mathsf{u}\}$$

Variable	Domain
v_1	$\{a, b, c, d\}$
v_2	$\{e, f\}$

Table 3.1: State variables of Σ

Operator	Precondition	Postcondition	Prevailcondition
o_1	$v_1 = a$	$v_1 = d$	$v_2 = e$
o_2	$v_1 = b$	$v_1 = d$	$v_2 = f$
o_3	$v_1 = c, v_2 = e$	$v_1 = d, v_2 = f$	

Table 3.2: Operators of Σ

For each $v \in \mathcal{V}$ and $\mathcal{O}' \subseteq \mathcal{O}$, the set $\mathcal{R}_v^{\mathcal{O}'}$ of **requestable values** for $\mathcal{O}'$ is defined as

$$\begin{aligned} \mathcal{R}_v^{\mathcal{O}'} &= \mathcal{P}_v^{\mathcal{O}'} \cup \\ & \quad \{\mathsf{pre}(o)[v], \mathsf{post}(o)[v] \mid o \in \mathcal{O}' \text{ and } o \text{ is non-unary}\} - \{\mathsf{u}\}. \end{aligned}$$

Example 3.2.2 For Σ, $\mathcal{P}_{v_1}^{\mathcal{O}} = \varnothing$ and $\mathcal{P}_{v_2}^{\mathcal{O}} = \{e, f\}$. Furthermore, $\mathcal{R}_{v_1}^{\mathcal{O}} = \{c, d\}$ and $\mathcal{R}_{v_2}^{\mathcal{O}} = \{e, f\}$.

Obviously, $\mathcal{P}_v^{\mathcal{O}} \subseteq \mathcal{R}_v^{\mathcal{O}} \subseteq \mathcal{D}_v$ for all $v \in \mathcal{V}$. For each state variable domain, we further define the graph of possible transitions for this domain, without taking the other domains into account, and the reachability graph for arbitrary subsets of the domain.

Definition 3.2.2 For each $v \in \mathcal{V}$, we define the corresponding **domain transition graph** G_v as a directed labelled graph $G_v = \langle \mathcal{D}_v^+, \mathcal{T}_v \rangle$ with vertex set $\mathcal{D}_v^+$ and arc set $\mathcal{T}_v$ s.t. for all $x, y \in \mathcal{D}_v^+$ and $o \in \mathcal{O}$, $\langle x, o, y \rangle \in \mathcal{T}_v$ iff $\mathsf{pre}(o)[v] \sqsubseteq x$ and $\mathsf{post}(o)[v] = y \neq \mathsf{u}$. Further, for each $X \subseteq \mathcal{D}_v^+$ we define the **reachability graph** for X as a directed graph $G_v^X = \langle X, \mathcal{T}_X \rangle$ with vertex set X and arc set $\mathcal{T}_X$ s.t. for all $x, y \in X$ s.t. $x \neq y$, $\langle x, y \rangle \in \mathcal{T}_X$ iff there is a path from x to y in G_v.

As an example, the domain-transition graphs of Σ are provided in Figure 3.1. Another way of viewing G_v^X is as the restriction to $X \subseteq \mathcal{D}_v^+$ of the transitive closure of G_v, but with unlabelled arcs and with all reflexive arcs removed. When referring to a path in a domain-transition graph below, we will typically mean the sequence of labels, *ie.* operators, along this path. We say that a path in G_v is *via* a set $X \subseteq \mathcal{D}_v$ iff each member of X is visited along the path, possibly as the initial or final vertex.

Definition 3.2.3 Let $G = \langle V, E \rangle$ be a directed graph and let G' be its associated undirected graph. The set of **components** of G is the set of largest subgraphs of G such that the associated subgraphs in G' are connected.

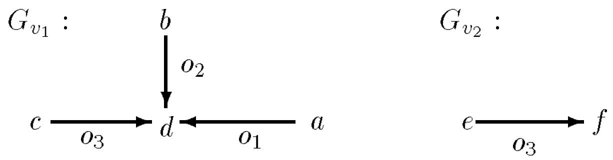


Figure 3.1: Domain-transition graphs of Σ. The undefined value is excluded.

Definition 3.2.4 An operator $o \in \mathcal{O}$ is **irreplaceable** wrt. a variable $v \in \mathcal{V}$ iff removing some arc labelled with o in G_v splits some component of G_v into two components.

Example 3.2.3 In Σ, o_3 is irreplaceable wrt. v_1 because removing it separates G_{v_1} into disjoint components. Similarly, o_3 is irreplaceable wrt. v_2.

Definition 3.2.5 Let $\mathcal{O}$ be a set of operators and let $\omega, \omega' \in Seqs(\mathcal{O})$ such that $\omega = \langle o_1, \ldots, o_k \rangle$ and $\omega' = \langle o'_1, \ldots, o'_n \rangle$. Then, $\omega \trianglelefteq \omega'$ iff there exists a subsequence $\langle o'_{i_1}, \ldots, o'_{i_k} \rangle$ of ω' such that $\mathsf{prv}(o_j) \sqsubseteq \mathsf{prv}(o'_{i_j})$ for $1 \leq j \leq k$.

Note that $\langle\rangle \trianglelefteq \omega$ for any $\omega \in Seqs(\mathcal{O})$. Furthermore, if $\omega, \omega', \psi, \psi' \in Seqs(\mathcal{O})$ and $\omega \trianglelefteq \omega'$, $\psi \trianglelefteq \psi'$, then $(\omega; \psi) \trianglelefteq (\omega'; \psi')$. In the remainder of this paper we will be primarily interested in SAS$^+$ instances satisfying the following restrictions.

Definition 3.2.6 The SAS$^+$ instance Π is:

(I) Interference-safe iff every operator $o \in \mathcal{O}$ is either unary or irreplaceable wrt. every $v \in \mathcal{V}$ it affects.

(A$^-$) Acyclic wrt. prevail-requestable values iff $G_v^{\mathcal{P}_v^{\mathcal{O}}}$ is acyclic for each $v \in \mathcal{V}$.

(A) Acyclic wrt. requestable values iff $G_v^{\mathcal{R}_v^{\mathcal{O}}}$ is acyclic for each $v \in \mathcal{V}$.

(A$^+$) Acyclic iff G_v is acyclic for each $v \in \mathcal{V}$.

(O) Prevail-order-preserving iff for all $v \in \mathcal{V}$, all $x, y \in \mathcal{D}_v^+$, all $X \subseteq \mathcal{R}_v^{\mathcal{O}}$ and all $\omega = \langle o_1, \ldots, o_m \rangle$, $\omega' = \langle o'_1, \ldots, o'_n \rangle$ in $Seqs(\mathcal{O})$, if ω is a shortest path from x to y via X and ω' is a path from x to y via X, then $\omega \trianglelefteq \omega'$.

Example 3.2.4 Consider the Σ example. By example 3.2.3, it is easy to see that Σ is interference-safe since o_3 is the only non-unary action and it is irreplaceable wrt. to all variables it affects. Since both G_{v_1} and G_{v_2} are acyclic, Σ satisfies A and as there exists one path at most between any two values in the domain-transition graphs, prevail-order-preservation also follows.

3.3 Testing Structural Restrictions

It is trivial to test the restrictions A^-, A and A^+ in polynomial time. Similarly it is easy to check whether an operator o is irreplaceable or not. For each variable that o affects, remove o from the corresponding domain-transition graphs and test if the number of components increases in each of the domain-transition graphs. This can be done in polynomial time by using some standard graph component algorithm [Baase, 1988]. Hence, checking restriction I takes polynomial time. Checking prevail-order-preservation in isolation seems hard. Fortunately, this restriction is tractable to test for instances that are acyclic wrt. requestable values, so the combination AO is tractable to test as we will show below. First, we define the notion of *weak* prevail-order-preservation.

Definition 3.3.1 A domain-transition graph G_v is **weakly prevail-order-preserving** iff for all $x, y \in \mathcal{D}_v^+$ and for all $\omega = \langle o_1, \ldots, o_m \rangle$, $\omega' = \langle o'_1, \ldots, o'_n \rangle \in \mathit{Seqs}(\mathcal{O})$, if ω is a shortest path from x to y and ω' is a path from x to y, then $\omega \trianglelefteq \omega'$.

We continue by showing that if v is a state-variable satisfying restriction A, then weak and ordinary prevail-order-preservation are equivalent.

Theorem 3.3.2 Let v be a state variable such that $G_v^{\mathcal{R}_v^{\mathcal{O}}}$ is acyclic. Then, G_v is prevail-order-preserving iff G_v is weakly prevail-order-preserving.

Proof: *only-if:* Immediate from the special case when the paths go via the empty set.
if: Let Z be an arbitrary subset of $\mathcal{R}_v^{\mathcal{O}}$. Choose x, y arbitrarily in $\mathcal{D}_v^+$ such that there exists a path from x to y via Z in G_v. Let ω be a shortest path from x to y via Z and let ψ be any path from x to y via Z. The theorem is proved by induction over $|Z|$.
Basis step: If $|Z| = 0$ then $\omega \trianglelefteq \psi$ by weak prevail-order-preservation.

Induction hypothesis: Suppose the theorem holds for all Z such that $|Z| = k$ for some $k \geq 0$.
Induction step: We prove the claim for $|Z| = k+1$. We have two cases:

1. $y = \mathsf{u}$: Since $G_v^{\mathcal{R}_v^{\mathcal{O}}}$ is acyclic and there exists a path from x to y via Z, there exists a unique value $z \in Z$ such that there exists a path from every other value in Z to z in G_v. It is obvious that every minimal path ω' from x to u via Z is such that $\mathsf{post}(Last(\omega'))[v] = z$. Hence, we can consider ω as a shortest path from x to z via $Z - \{z\}$ and ψ as a path from x to z via $Z - \{z\}$. By the induction hypothesis, $\omega \trianglelefteq \psi$.

2. $y \neq \mathsf{u}$: Since $G_v^{\mathcal{R}_v^{\mathcal{O}}}$ is acyclic there exists a unique value $z \in Z$ such that there exists a path from every other state in Z to z. Let ω_1 be a minimal prefix of ω from x to z via $Z - \{z\}$. In ψ, there exists a prefix $\psi_1 = \langle d_1, \ldots, d_i \rangle$, from x to z via $Z - \{z\}$. By the induction hypothesis, $\omega_1 \trianglelefteq \psi_1$. Furthermore, there exists a suffix ω_2 from z to y in ω such that $\langle \omega_1; \omega_2 \rangle = \omega$. Since ω is a shortest path from x to y via Z, ω_2 is a shortest path from z to y. Furthermore, there exists a suffix ψ_2 from z to y via $\varnothing$ in ψ such that $\langle \psi_1; \psi_2 \rangle = \psi$. Since G_v is weakly prevail-order-preserving, $\omega_2 \trianglelefteq \psi_2$ and hence, $\omega \trianglelefteq \psi$. □

So, under restriction A, it is sufficient to check if an instance is weakly prevail-order-preserving to conclude that it satisfies restriction O.

Definition 3.3.3 Let v be a state variable and let $x, y \in \mathcal{D}_v^+$. Then, two paths $\omega = \langle o_1; \ldots; o_n \rangle, \omega' = \langle o'_1; \ldots; o'_n \rangle \in Seqs(\mathcal{O})$ of equal length n from x to y are **prevail-consistent** iff $\mathsf{prv}(o_i) = \mathsf{prv}(o'_i)$ for all $1 \leq i \leq n$, which we denote $\omega \triangleq \omega'$.

An alternative definition is that $\omega \triangleq \omega'$ iff $\omega \trianglelefteq \omega'$ and $\omega' \trianglelefteq \omega$. Note that prevail-consistency is an equivalence relation.

Definition 3.3.4 A state variable v is **shortest-path consistent** iff for all $x, y \in \mathcal{D}_v^+$ and all shortest paths ω, ω' from x to y in G_v, ω and ω' are prevail-consistent.

Since $\langle\rangle$ is always the shortest path from a value $x \in \mathcal{D}_v^+$ to u, we can without loss of generality assume that $y \in \mathcal{D}_v$ in order to simplify the forthcoming proofs.

1 **procedure** *Check-SPC*(v)
2 **if** there exists two operators $o, o' \in \mathcal{O}$ such that the following holds:

- $\mathsf{pre}(o)[v] = \mathsf{pre}(o')[v]$ or $\mathsf{pre}(o)[v] = \mathsf{u}$
- $\mathsf{post}(o)[v] = \mathsf{post}(o')[v] \neq \mathsf{u}$
- $\mathsf{prv}(o) \neq \mathsf{prv}(o')$

then reject;
3 **for** $i \leftarrow 2$ **to** $|\mathcal{D}_v^+|$ **do**
4 **for** all $x \in \mathcal{D}_v^+, y \in \mathcal{D}_v$ s.t. there exists a shortest path from x to y in G_v of length i **do**
5 let ω be any such path
6 **for** $z \in \mathcal{D}_v - \{x, y\}$ **do**
7 $\gamma_1 \leftarrow$ a shortest path from x to z in G_v;
8 $\gamma_2 \leftarrow$ a shortest path from z to y in G_v;
9 **if** $|(\gamma_1; \gamma_2)| = |\omega|$ and not $\omega \triangleq (\gamma_1; \gamma_2)$ **then reject**;
10 **accept**;

Figure 3.2: Algorithm for testing shortest-path consistency

Lemma 3.3.5 Let v be a state variable that satisfies restriction O. Then, v is shortest-path consistent.

Proof: Arbitrarily choose $x \in \mathcal{D}_v^+$ and $y \in \mathcal{D}_v$. Let $\omega = \langle o_1, \ldots, o_n \rangle, \omega' = \langle o'_1, \ldots, o'_n \rangle \in Seqs(\mathcal{O})$ be two arbitrary shortest paths from x to y. By prevail-order-preservation, $\omega \trianglelefteq \omega'$ and $\omega' \trianglelefteq \omega$ and consequently $\omega \triangleq \omega'$. $\Box$

We now present an algorithm, *Check-SPC*, for checking shortest-path consistency. The algorithm is based on Floyd's [1962] algorithm and can be found in Figure 3.2. Note that the **for**-loop in line 3 is not technically necessary for the algorithm to function properly, it is merely inserted to make the connection between the algorithm and the correctness proof clearer.

Lemma 3.3.6 Let v be a state variable. Then, *Check-SPC* accepts iff v is shortest-path consistent.

Proof: *if:* We show the contrapositive. If *Check-SPC* rejects, then it has constructed an example of two shortest paths that are not prevail-consistent, either in line 2 or in line 9. Consequently, v cannot be shortest-path consistent.

only-if: We must show that if *Check-SPC* accepts, then there does not exist any $x \in \mathcal{D}_v^+, y \in \mathcal{D}_v$ and two shortest paths ω, ψ from x to y such that ω, ψ are not prevail-consistent. We show this by induction over n, the length of the shortest path from x to y. The basis step of the induction corresponds to the test in line 2 of *Check-SPC*. The induction steps corresponds to lines 3-9 for different values of i.
Basis step: Arbitrarily choose $x \in \mathcal{D}_v^+$, $y \in \mathcal{D}_v$ such that $n = 0$ or $n = 1$. Let ω and ψ be arbitrary shortest paths from x to y. If $n = 0$ then $\omega = \psi = \langle\rangle$ is a path from some $x \in \mathcal{D}_v$ to itself and $\omega \triangleq \psi$ follows immediately. If $n = 1$, then $\omega = (o)$ and $\psi = (p)$. Since the algorithm did not fail in line 2, $\mathsf{prv}(o)[v] = \mathsf{prv}(o')[v]$ and $\omega \triangleq \psi$ follows.
Induction hypothesis: Arbitrarily choose $x \in \mathcal{D}_v^+$, $y \in \mathcal{D}_v$ such that $n \leq k$, $k \geq 1$. For all pairs of shortest paths ω, ψ from x to y of length n in G_v, assume $\omega \triangleq \psi$.
Induction step: Arbitrarily choose $x \in \mathcal{D}_v^+$, $y \in \mathcal{D}_v$ such that $n = k + 1$, $k \geq 1$. Let ω and ψ be two arbitrary shortest paths from x to y in G_v. Let $\overline{\omega}$ be the shortest path from x to y that the algorithm constructs in line 5.

Since $|\omega| = k + 1$ and $k \geq 1$, there exists some $r \in \mathcal{D}_v$ such that ω is a shortest path from x to y via r, $r \neq x$ and $r \neq y$. Let γ_1 and γ_2 be the paths that the algorithm chooses in lines 7 and 8 when the value of z is r. Partition ω in two non-empty paths, ω_1, ω_2 such that $\omega = (\omega_1; \omega_2)$ and $\mathsf{post}(\mathit{Last}(\omega_1))[v] = r$. Since ω is a shortest path from x to y, ω_1 is a shortest path from x to r and ω_2 is a shortest path from r to y. By the induction hypothesis, $\gamma_1 \triangleq \omega_1$ and $\gamma_2 \triangleq \omega_2$ which implies that $(\gamma_1; \gamma_2) \triangleq (\omega_1; \omega_2)$. The algorithm did not fail in line 9, so we have that

$$\overline{\omega} \triangleq (\gamma_1; \gamma_2) \triangleq (\omega_1; \omega_2) = \omega$$

Similarily $\overline{\omega} \triangleq \psi$ so since $\triangleq$ is a transitive relation, $\omega \triangleq \psi$ and the induction step follows. $\square$

We are now able to devise an algorithm for checking weak prevail-order-preservation. The algorithm, *Check-WO*, can be found in Figure 3.3. Once again, the algorithm is based on Floyd's shortest-path algorithm and the **for**-loop in line 2 is not necessary for the algorithm to work properly, it is just inserted to clarify the correctness proof.

Lemma 3.3.7 Let v be shortest-path-consistent state variable. Then, *Check-WO* accepts iff v is weakly prevail-order-preserving.

1 **procedure** *Check-WO*(v)
2 **for** $i \leftarrow 2$ **to** $|\mathcal{D}_v^+|$ **do**
3   **for** all $x \in \mathcal{D}_v^+, y \in \mathcal{D}_v$ s.t. there exists a shortest path ω from x to y in G_v of length i **do**
4   let ω be any such path
5   **for** $z \in \mathcal{D}_v - \{x, y\}$ **do**
6          $\gamma_1 \leftarrow$ a shortest path from x to z in G_v;
7          $\gamma_2 \leftarrow$ a shortest path from z to y in G_v;
8   **if** $\omega \not\trianglelefteq (\gamma_1; \gamma_2)$ **then reject**;
9 **accept**;

Figure 3.3: Algorithm for testing weak prevail-order-preservation

Proof: *if:* We show the contrapositive. If *Check-WO* rejects, then it has constructed an example of two paths that do not satsify the requirements of weak prevail-order-preservation in line 8. Consequently, v cannot be weakly prevail-order-preserving.

only-if: We must show that if *Check-WO* accepts, then there does not exist any $x \in \mathcal{D}_v^+, y \in \mathcal{D}_v$, a shortest path ω from x to y and some other path ψ from x to y such that $\omega \not\trianglelefteq \psi$. We show this by induction over n, the length of the shortest path from x to y. The induction steps corresponds to lines 2-8 for different values of i.

Basis step: Arbitrarily choose $x \in \mathcal{D}_v^+$, $y \in \mathcal{D}_v$ such that $n = 0$. Let ω be the shortest path $\langle\rangle$ from x to y and let ψ be an arbitrary path from x to y. Obviously, $\omega \trianglelefteq \psi$.

Induction hypothesis: Arbitrarily choose $x \in \mathcal{D}_v^+$, $y \in \mathcal{D}_v$ such that $n \leq k$, $k \geq 0$. For all shortest paths ω of length $\leq n$ and all paths ψ from x to y in G_v, assume $\omega \trianglelefteq \psi$.

Induction step: Arbitrarily choose $x \in \mathcal{D}_v^+$, $y \in \mathcal{D}_v$ such that $n = k + 1$, $k \geq 0$. We show that for an arbitrary shortest path ω and some arbitrary path ψ from x to y in G_v, $\omega \trianglelefteq \psi$. Let ϖ be the shortest path from x to y that the algorithm constructs in line 4. By shortest-path consistency, $\omega \triangleq \varpi$. Clearly, $|\psi| \neq 0$ because $|\omega| \leq |\psi|$ and $|\omega| \geq 1$. If $|\psi| = 1$, then $\omega \trianglelefteq \psi$ because $\omega = \langle o \rangle$ and $\omega \triangleq \psi$ by shortest-path consistency. Otherwise there exists some $r \in \mathcal{D}_v$, $r \neq x$ and $r \neq y$, such that ψ is a path from x to y via r. Let γ_1 and γ_2 be the paths that the algorithm chooses in lines 6 and 7 when the value of z is r. Partition ψ into two non-empty paths, ψ_1, ψ_2 such that $\psi = (\psi_1; \psi_2)$ and $\mathsf{post}(Last(\omega_1))[v] = r$. We know that γ_1 is a shortest path

```
1 procedure Check-O(Π) (* Π = ⟨V, O, s_0, s_*⟩ *)
2 for v ∈ V do
3     Check-SPC(v);
4     Check-WO(v);
5 accept;
```

Figure 3.4: Algorithm for testing prevail-order-preservation

from x to r and γ_2 is a shortest path from r to y. Hence, by the induction hypothesis, $\gamma_1 \trianglelefteq \psi_1$ and $\gamma_2 \trianglelefteq \psi_2$ which implies that $(\gamma_1;\gamma_2) \trianglelefteq (\psi_1;\psi_2)$. The algorithm did not fail in line 8, so we have that

$$\overline{\omega} \trianglelefteq (\gamma_1;\gamma_2) \trianglelefteq (\psi_1;\psi_2) = \psi.$$

But we know that $\omega \triangleq \overline{\omega}$ and consequently $\omega \trianglelefteq \psi$, which concludes the induction. □

Theorem 3.3.8 Let Π be a SAS$^+$ instance satisfying restriction A. Then, *Check-O* accepts iff Π is prevail-order-preserving.

Proof: *if:* Arbitrarily select $v \in \mathcal{V}$. Since v satisfies restriction O, v is shortest-path consistent by Lemma 3.3.5. Consequently, *Check-O* cannot fail in line 3. Furthermore, v is weakly prevail-order-preserving by Theorem 3.3.2 so *Check-O* cannot fail in line 4. Hence, if v is prevail-order-preserving, *Check-O* accepts.
only-if: Arbitrarily select $v \in \mathcal{V}$. Since *Check-O* did not fail in line 3, v is shortest-path consistent by Lemma 3.3.6. Knowing that v is shortest-path consistent and that *Check-O* did not fail in line 4, we can draw the conclusion that v is weakly prevail-order-preserving by Lemma 3.3.7. Furthermore, we know that $G_v^{\mathcal{R}_v^{\mathcal{O}}}$ is acyclic. Hence, by Theorem 3.3.2, v is prevail-order-preserving. □

We proceed by analysing the worst-case time complexity of *Check-O* using straightforward, non-optimal assumptions about data structures. As we do not attempt proving an optimal bound on the complexity, we assume that states are represented as arrays of values and operators as triples of states. With this representation, we can find a shortest path between two values in a domain-transition graph G_v in $O(|\mathcal{D}_v^+||\mathcal{O}|)$ time, for example, by using Dijkstra's algorithm [Dijkstra, 1959]. (This follows from the fact that, in the worst case, one single operator can introduce $\mathcal{D}_v^+$ arcs into G_v.) Naturally, we can also decide if there exists a path between two values in a domain-transition graph in $O(|\mathcal{D}_v^+||\mathcal{O}|)$ time.

Lemma 3.3.9 *Check-SPC* has a worst-case time complexity of $O(|\mathcal{O}|^2 + |\mathcal{O}||\mathcal{D}_v^+|^5)$.

Proof: The number of operator pairs in line 2 is bounded from above by $|\mathcal{O}|^2$. The $\triangleq$ check in line 9 can clearly be performed in $O(|\omega|)$ time if we assume that comparing prevail-conditions of operators is a basic operation. Since ω is a shortest path, its length does not exceed $\mathcal{D}_v^+$, so the test takes $O(|\mathcal{D}_v^+|)$ time. Hence, lines 7-9 run in $O(|\mathcal{O}||\mathcal{D}_v^+| + |\mathcal{D}_v^+|) \subseteq O(|\mathcal{O}||\mathcal{D}_v^+|)$ time and consequently lines 6-9 runs in $O(|\mathcal{O}||\mathcal{D}_v^+|^2)$ time. Lines 4-9 can iterate at most $|\mathcal{D}_v^+|^2$ times and each turn requires the computation of a shortest path so lines 4-9 runs in

$$O(|\mathcal{D}_v^+||\mathcal{O}| + |\mathcal{O}||\mathcal{D}_v^+|^4)$$

time which equals

$$O(|\mathcal{O}||\mathcal{D}_v^+|^4).$$

Hence, the loop in lines 3 to 9 runs in

$$O(|\mathcal{O}||\mathcal{D}_v^+|^5)$$

time, so *Check-SPC* runs in $O(|\mathcal{O}|^2 + |\mathcal{O}||\mathcal{D}_v^+|^5)$ time. □

Lemma 3.3.10 *Check-WO* has a worst-case time complexity of $O(|\mathcal{O}||\mathcal{D}_v^+|^5)$.

Proof: The $\trianglelefteq$ check in line 8 can clearly be performed in $O(|\omega|)$ time if we assume that comparing prevail-conditions of operators is a basic operation. Since ω is a shortest-path, its length does not exceed $\mathcal{D}_v^+$ so the test takes $O(|\mathcal{D}_v^+|)$ time. Hence, lines 6-8 run in $O(|\mathcal{O}||\mathcal{D}_v^+| + |\mathcal{D}_v^+|) \subseteq O(|\mathcal{O}||\mathcal{D}_v^+|)$ time and consequently lines 5-8 runs in $O(|\mathcal{O}||\mathcal{D}_v^+|^2)$ time. Lines 3-8 can iterate at most $|\mathcal{D}_v^+|^2$ times and each turn requires the computation of a shortest path so lines 3-8 runs in

$$O(|\mathcal{D}_v^+|^2(|\mathcal{D}_v^+||\mathcal{O}| + |\mathcal{O}||\mathcal{D}_v^+|^2))$$

time which equals

$$O(|\mathcal{O}||\mathcal{D}_v^+|^4).$$

Hence, the loop in lines 2 to 8 runs in

$$O(|\mathcal{O}||\mathcal{D}_v^+|^5).$$

time, so *Check-WO* runs in $O(|\mathcal{O}||\mathcal{D}_v^+|^5)$ time. □

Theorem 3.3.11 *Check-O* has a worst-case time complexity of $O(|\mathcal{V}||\mathcal{O}|(|\mathcal{O}| + |\mathcal{V}| \max_{v \in \mathcal{V}} |\mathcal{D}_v^+|^5))$.

Proof: By Lemmata 3.3.9 and 3.3.10, lines 3 and 4 runs in $O(|\mathcal{O}|^2 + |\mathcal{O}||\mathcal{V}| \max_{v \in \mathcal{V}} |\mathcal{D}_v^+|^5)$ time. Hence, *Check-O* runs in $O(|\mathcal{V}||\mathcal{O}|(|\mathcal{O}| + |\mathcal{V}| \max_{v \in \mathcal{V}} |\mathcal{D}_v^+|^5))$ time. □

Chapter 4

Planning Algorithm for SAS$^+$-IAO

We begin this section by presenting the SAS$^+$-IAO planning algorithm and explaining how it works in informal terms. The formal correctness proofs will be postponed to the next section. The algorithm is illustrated with a small workshop example. This section is concluded with a proof showing that SAS$^+$-IAO is strictly more general than the SAS$^+$-PUS problem which is the maximally tractable problem under the syntactical restrictions P, U, B and S [Bäckström and Nebel, 1993].

4.1 The Algorithm

Before describing the actual planning algorithm, we make the following observations about the solutions to arbitrary SAS$^+$ instances.

Lemma 4.1.1 Let $\Theta = \langle \mathcal{B}, \prec_\Theta \rangle$ be a partial-order plan solving some SAS$^+$ instance $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$. Then for each $v \in \mathcal{V}$ and for each action sequence α which is a member of $\mathit{Paths}(v, \langle \mathcal{B}, \prec_\Theta \rangle)$, the operator sequence $\mathit{type}(\alpha)$ is a path in G_v from $s_0[v]$ to $s_*[v]$ via $\mathcal{R}_v^{\mathcal{B}}$.

Proof: It can be easily shown that $\mathit{type}(\alpha)$ is a path in G_v from $s_0[v]$ to $s_*[v]$ (A proof for this can be found in Klein [1993, Lemma C.17]) so it remains to show that this path goes via $\mathcal{R}_v^{\mathcal{B}}$. Suppose there exists some $z \in \mathcal{R}_v^{\mathcal{B}}$ with no action a in α such that $\mathsf{post}(a)[v] = z$ and $s_0[v] \neq z$. By the definition of requestable value, we have two cases:

- There exists some action $b \in \mathcal{B}$ such that $\mathsf{prv}(b)[v] = z$. Since $s_0[v] \neq z$ and no action in Θ achieves z, Θ is not a valid plan. Contradiction and the theorem follows.

- There exists some non-unary action $b \in \mathcal{B}$ such that $\mathsf{pre}(b)[v] = z$ or $\mathsf{post}(b)[v] = z$. Since $s_0[v] \neq z$ and no action in Θ achieves z, b cannot be executed so Θ is not a valid plan. Contradiction and the theorem follows. □

This is a declarative characterization of the solutions and it cannot be immediately cast in procedural terms—the main reason being that for a solution $\langle \mathcal{A}, \prec \rangle$, we cannot know the $\mathcal{R}_v^{\mathcal{A}}$ sets in advance. These sets must, hence, be computed incrementally, which can be done in polynomial time under the restrictions I and A. The algorithm *Plan* (Figures 4.1 and 4.2) serves as a plan generation algorithm under these restrictions. Henceforth, we assume $\mathcal{V} = \{v_1, \ldots, v_m\}$.

The heart of the algorithm is the procedure *Extend*, which operates on the global variables $X_1 \subseteq \mathcal{D}_{v_1}, \ldots, X_m \subseteq \mathcal{D}_{v_m}$, extending these monotonically. It also returns operator sequences in the global variables $\omega_1, \ldots, \omega_m$, but only their value after the last call are used by *Plan*. For each i, *Extend* first finds a shortest path ω_i in G_{v_i} from $s_0[v_i]$ to $s_*[v_i]$ via X_i. (The empty path $\langle\rangle$ is considered as the shortest path from any vertex x to u, since $\mathsf{u} \sqsubseteq x$). If no such path exists, then *Extend* fails and otherwise each X_i is set to $\mathcal{R}_{v_i}^{\mathcal{O}'}$, where $\mathcal{O}'$ is the set of all operators along the paths $\omega_1, \ldots, \omega_m$. The motivation for this is as follows: If $\mathsf{prv}(o)[v_i] = x \neq \mathsf{u}$ for some i and some operator o in some ω_j, then some action in the final plan must achieve this value, unless it holds initially. Hence, x is added to X_i to ensure that *Extend* will find a path via x in the next iteration. Similarly, each non-unary operator occurring in some ω_i must also appear in ω_j for all j such that o affects v_j.

Starting with all $X_1, \ldots, X_m$ initially empty, *Plan* calls *Extend* repeatedly until nothing more is added to these sets or *Extend* fails. Viewing *Extend* as a function $\mathit{Extend} : 2^{\mathcal{D}_{v1}} \times \ldots \times 2^{\mathcal{D}_{vm}} \rightarrow 2^{\mathcal{D}_{v1}} \times \ldots \times 2^{\mathcal{D}_{vm}}$, *ie.* ignoring the side effect on $\omega_1, \ldots, \omega_m$, this process corresponds to constructing the minimal fixed point for *Extend* in $2^{\mathcal{D}_{v1}} \times \ldots \times 2^{\mathcal{D}_{vm}}$. The paths $\omega_1, \ldots, \omega_m$ found in the last iteration contain all the operators necessary in the final solution and procedure *Instantiate* instantiates these as actions. This works such that all occurrences of a non-unary operator are merged into one unique instance while all occurrences of unary operators are made into distinct instances. It

procedure *Plan*$(\langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle)$;
$\langle X_1, \ldots, X_m \rangle \leftarrow \langle \varnothing, \ldots, \varnothing \rangle$;
 repeat
   $\langle X_1, \ldots, X_m \rangle \leftarrow \mathit{Extend}(\langle X_1, \ldots, X_m \rangle)$;
until no X_i is changed;
$\alpha_1, \ldots, \alpha_m \leftarrow \langle \rangle$;
Instantiate;
comment *Instantiate all operators s.t. all non-unary operators of the same type are instantiated to the same occurence and unary operators are always instantiated to different occurences.*
for $1 \leq i \leq m$ and $a, b \in \alpha_i$ **do**
   Order $a \prec b$ iff a immediately precedes b in α_i;
for $1 \leq i \leq m$ and $a \in \mathcal{A}$ s.t. $\mathsf{prv}(a)[v_i] \neq \mathsf{u}$ **do**
   Assume $\alpha_i = \langle a_1, \ldots, a_k \rangle$
   if $\mathsf{post}(a_l)[v_i] = \mathsf{prv}(a)[v_i]$ for some $1 \leq l \leq k$ **then**
      Order $a_l \prec a$;
      if $l < k$ **then** Order $a \prec a_{l+1}$;
   else Order $a \prec a_1$;
$\mathcal{A} \leftarrow \{a \in \alpha_i \mid 1 \leq i \leq m\}$;
if $\prec$ is irreflexive **then return** $\langle \mathcal{A}, \prec \rangle$;
else fail;

Figure 4.1: Planning algorithm

function *Extend* $(\langle X_1, \ldots, X_m \rangle)$
comment *The variables* $\omega_1, \ldots, \omega_m$ *are implicitly used as output parameters.*
for $1 \le i \le m$ **do**
  $\omega_i \leftarrow$ *Shortest-Path*$(v_i, s_0[v_i], s_*[v_i], X_i)$;
  if no such path exists **then** **fail**;
for $1 \le i \le m$ **do**
  for $1 \le j \le m$ and $o \in \omega_j$ **do**
    $X_i \leftarrow X_i \cup \{\mathsf{prv}(o)[v_i]\} - \{\mathsf{u}\}$;
for $1 \le i \le m$ **do**
  for $1 \le j \le m$ and $o \in \omega_j$ **do**
    if o not unary **then** $X_i \leftarrow X_i \cup \{\mathsf{pre}(o)[v_i], \mathsf{post}(o)[v_i]\} - \{\mathsf{u}\}$;
return $\langle X_1, \ldots, X_m \rangle$;

procedure *Shortest-Path*(v, x, y, X)
if not every value in X is reachable from x **then fail**
$\overline{X} \leftarrow \langle x \rangle$; $\omega \leftarrow \langle\rangle$;
while $X \neq \varnothing$ **do**
  if there exists a state z in X such that there exists a path in G_v from z to all other states in $X \cup \{y\}$ **then**
    $\overline{X} \leftarrow (\overline{X}; z)$;
    $X \leftarrow X - \{z\}$;
  else **fail**;
$\overline{X} \leftarrow (\overline{X}; y)$;
comment *SP is some standard algorithm for finding shortest paths in graphs, e.g. Dijkstras's algorithm*
for $i \leftarrow 1$ **to** $|\overline{X}| - 1$ **do**
  $\omega \leftarrow (\omega; SP(\overline{X}[i], \overline{X}[i+1]))$;
return ω;

procedure *Instantiate*; (*Modifies* $\alpha_1, \ldots, \alpha_m$)
for $1 \le i \le m$ **do**
  Assume $\omega_i = \langle o_1, \ldots, o_k \rangle$
  for $1 \le l \le k$ **do**
    if o_l not unary **and** there is some a of type o_l in α_j for some $j < i$ **then** $a_l \leftarrow a$;
    else Let a_l be a new instance of *type*(a_l);
  $\alpha_i \leftarrow \langle a_1, \ldots, a_k \rangle$;

Figure 4.2: Subroutines to planning algorithm

remains to compute the action ordering on the set $\mathcal{A}$ of all such operator instances (actions). For each v_i, the total order implicit in the operator sequence ω_i is kept as a total ordering on the corresponding actions. Finally, each action a s.t. $\mathsf{prv}(a)[v_i] = x \neq \mathsf{u}$ for some i must be ordered after some action a' providing this condition. There turns out to always be a unique such action, or none if $v_i = x$ initially. Similarly, a must be ordered before the first action succeeding a' that destroys its prevailcondition. Finally, if the transitive closure $\prec^+$ of $\prec$ is irreflexive, then $\langle \mathcal{A}, \prec \rangle$ is returned and otherwise *Plan* fails.

4.2 An Example: The Manufacturing Workshop

In this subsection, we will present a small, somewhat contrived example of a manufacturing workshop and show how the algorithm handles this example. We assume that there is a supply of rough workpieces and a table for putting finished products. There are also two workstations: a lathe and a drill. To simplify matters, we will consider only one single workpiece.[1] Two different shapes can be made in the lathe and one type of hole can be drilled. Furthermore, only workpieces of shape 2 fit in the drill. This gives a total of four possible combinations for the end product: rough (*ie.* not worked on), shape 1 or shape 2 without a hole and shape 2 with a hole. Note also that operator Shape2 is tougher to the cutting tool than Shape1 is—the latter allowing us to continue using the cutting tool afterwards. Finally, both the lathe and the drill require that the power is on. This is all modelled by five state variables, as shown in Table 4.1, and nine operators, as shown in Table 4.2. This example is a SAS$^+$-IAO instance, but it does not satisfy either of the P, U and S restrictions in Bäckström and Nebel [1993].[2]
Suppose we start in $s_0 = \langle S, R, M, N, N \rangle$ and set the goal $s_* = \langle T, 2, \mathsf{u}, Y, N \rangle$, that is, we want to manufacture a product of shape 2 with a drilled hole. We also know that the cutting tool for the lathe is initially in mint condition, but we do not care about its condition after finishing. Finally, the power is initially off and we are required to switch it off again before leaving the workshop.

[1] It has been suggested by Sandewall [1987] that it may be more natural to model the flow of a single workpiece and then transform the plan for this workpiece into separate cyclic plans for the workstations.

[2] Note in particular that since we do no longer require the S restriction, we can model sequences of workstations, which was not possible under the PUS restriction [Bäckström and Klein, 1991a].

Variable	Domain	Denotes
v_1	{Supply,Lathe,Drill,Table}	Position of workpiece
v_2	{Rough,1,2}	Workpiece shape
v_3	{Mint,Used}	Condition of cutting tool
v_4	{Yes,No}	Hole in workpiece
v_5	{Yes,No}	Power on

Table 4.1: State variables for the workshop example

Operator	Precondition	Postcondition	Prevailcondition
MvSL	$v_1 = S$	$v_1 = L$	
MvLT	$v_1 = L$	$v_1 = T$	
MvLD	$v_1 = L$	$v_1 = D$	$v_2 = 2$
MvDT	$v_1 = D$	$v_1 = T$	
Shape1	$v_2 = R$	$v_2 = 1$	$v_1 = L, v_3 = M, v_5 = Y$
Shape2	$v_2 = R, v_3 = M$	$v_2 = 2, v_3 = U$	$v_1 = L, v_5 = Y$
Drill	$v_4 = N$	$v_4 = Y$	$v_1 = D, v_5 = Y$
Pon	$v_5 = N$	$v_5 = Y$	
Poff	$v_5 = Y$	$v_5 = N$	

Table 4.2: Operators for the workshop example. (Domain values are denoted by their initial characters only.)

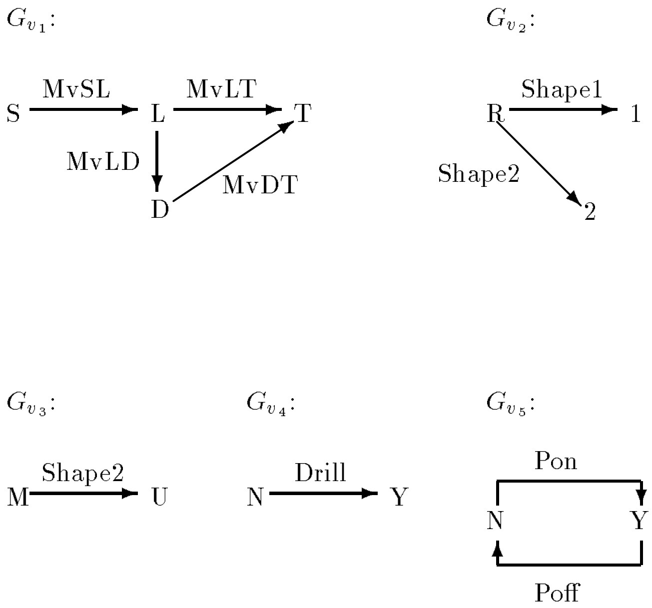


Figure 4.3: Domain-transition graphs for the workshop example.

Procedure *Plan* will make two calls to *Extend* before terminating the loop successfully, with variable values as follows:

After the first iteration:
$\omega_1 = \langle \mathrm{MvSL}, \mathrm{MvLT} \rangle, \omega_2 = \langle \mathrm{Shape2} \rangle, \omega_3 = \langle \rangle,$
$\omega_4 = \langle \mathrm{Drill} \rangle, \omega_5 = \langle \rangle$
$X_1 = \{L, D\}, X_2 = \{R, 2\}, X_3 = \{M, U\}, X_4 = \{\}, X_5 = \{Y\}$

After the second iteration:
$\omega_1 = \langle \mathrm{MvSL}, \mathrm{MvLD}, \mathrm{MvDT} \rangle, \omega_2 = \langle \mathrm{Shape2} \rangle, \omega_3 = \langle \mathrm{Shape2} \rangle,$
$\omega_4 = \langle \mathrm{Drill} \rangle, \omega_5 = \langle \mathrm{Pon}, \mathrm{Poff} \rangle$
$X_1 = \{L, D\}, X_2 = \{R, 2\}, X_3 = \{M, U\}, X_4 = \{\}, X_5 = \{Y\}$

The operators in the operator sequences $\omega_1, \ldots, \omega_5$ will be instantiated to actions, where both occurrences of Shape2 are instantiated as the same action, since Shape2 is non-unary. Since there is not more than one action of each type in this plan, we will use the name of the operators also as names of the actions. The total orders in $\omega_1, \ldots, \omega_m$ are retained in $\alpha_1, \ldots, \alpha_m$. Furthermore, Shape2 must be ordered after MvSL and before MvLD, since its prevailcondition on variable 1 equals the postcondition of MvSL for this variable. Similarly, Drill must be ordered between MvLD and MvDT because of its prevailcondition on variable 1 and both Shape2 and Drill must be ordered between Pon and Poff because of their prevailcondition on variable 5. Furthermore, MvLD must be (redundantly) ordered after Shape2 because of its prevailcondition on variable 2. The final partial-order plan is shown in Figure 4.4.

4.3 The Relation Between SAS$^+$-PUS and SAS$^+$-IAO

The maximally tractable problem under the syntactical restrictions is the SAS$^+$-PUS problem [Bäckström and Nebel, 1993]. Before proceeding with the correctness of algorithm *Plan*, we show that the SAS$^+$-IAO problem is strictly more general than the SAS$^+$-PUS problem [Bäckström, 1992b, Bäckström, 1992a]. We first show that every SAS$^+$-PUS instance is a SAS$^+$-IAO instance; then we construct a SAS$^+$-IAO instance that does not satisfy either P, U or S.

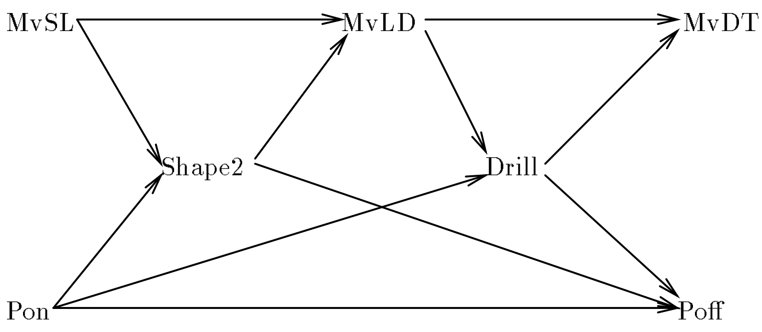


Figure 4.4: The final partial-order plan (only $\prec$, not $\prec^+$, is shown).

Lemma 4.3.1 If Π is a unary and single-valued SAS$^+$instance, then $G_v^{\mathcal{R}_v^{\mathcal{O}}}$ is acyclic for all $v \in \mathcal{V}$.

Proof: Follows immediately from the fact that $|\mathcal{R}_v^{\mathcal{O}}| \leq 1$ when an instance is both unary and single-valued. □

Lemma 4.3.2 If Π is a unary SAS$^+$instance, then Π is interference-safe.

Proof: Immediate from the definition of interference-safeness. □

Lemma 4.3.3 If a SAS$^+$instance Π is post-unique, unary and single-valued, then it is prevail-order-preserving.

Proof: We show that v is weakly prevail-order-preserving and by Lemmata 4.3.1 and 3.3.2, the lemma follows. By the definition of weak prevail-order-preservation, we have to show that there does not exist any $x \in \mathcal{D}_v^+, y \in \mathcal{D}_v$, any shortest path ω from x to y and some other path ψ from x to y such that $\omega \not\trianglelefteq \psi$. We show this by induction over n, the length of the shortest path from x to y.
Basis step: Arbitrarily choose $x \in \mathcal{D}_v^+$, $y \in \mathcal{D}_v$ such that $n = 0$. Every shortest-path ω from x to y equals $\langle\rangle$ so $\omega \trianglelefteq \psi$ where ψ is an arbitrary path from x to y.
Induction hypothesis: Arbitrarily choose $x \in \mathcal{D}_v^+$, $y \in \mathcal{D}_v$ such that $n = k$, $k \geq 0$. Let ω be a shortest path and let ψ be any path from x to y. Assume $\omega \trianglelefteq \psi$.

Induction step: Arbitrarily choose $x \in \mathcal{D}_v^+$, $y \in \mathcal{D}_v$ such that $n = k+1$, $k \geq 0$. Let ω be a shortest path and let ψ be any path from x to y. We have to show that $\omega \trianglelefteq \psi$. Let $o = Last(\omega)$, $p = Last(\psi)$ and $z = \mathsf{pre}(o)[v]$. Let ω' denote ω without its last operator and let ψ' denote ψ without its last operator. If $y = \mathsf{u}$, then $\omega = \langle\rangle$ which contradicts the fact that $|\omega| = k + 1 > 0$. Hence $y \neq \mathsf{u}$ so, by post-uniqueness, $o = p$. We must consider two cases:

1. $z = \mathsf{u}$. Since ω is minimal, it consists of the operator o. We know that $o = p$ and hence $\omega \trianglelefteq \psi$.

2. $z \neq \mathsf{u}$. Then, ω' is a minimal path from x to z and ψ' is a path from x to z. Since $|\omega'| = k$, we can apply the induction hypothesis and conclude that $\omega' \trianglelefteq \psi'$. But $\omega = \langle\omega'; o\rangle$ and $\psi = \langle\psi'; p\rangle$ so $\omega \trianglelefteq \psi$ because $o = p$.

One of these cases must hold, which concludes the induction. $\Box$

Theorem 4.3.4 Every SAS$^+$-PUS instance is a SAS$^+$-IAO instance.

Proof: Immediate from Lemmata 4.3.1, 4.3.2 and 4.3.3. $\Box$

Theorem 4.3.5 There exists a SAS$^+$-IAO instance that does not satisfy either P, U or S.

Proof: Consider the previously defined SAS$^+$-IAO instance Σ. Σ is not post-unique since $\mathsf{post}(o_1)[v_1] = \mathsf{post}(o_2)[v_1] = \mathsf{post}(o_3)[v_1] = d$, it is not unary since o_3 affects both v_1 and v_2 and it is not single-valued since $\mathsf{prv}(o_1)[v_2] = e$ and $\mathsf{prv}(o_2)[v_2] = f$. $\Box$

Chapter 5

Correctness and Complexity of Planning Algorithm

This section contains all correctness and complexity results for algorithm *Plan* in the previous chapter. We begin by showing that the algorithm is sound for SAS$^+$-IA instances and continue by proving that *Plan* generates minimal solutions and is complete for SAS$^+$-IAO instances. Finally, we show that it is a polynomial algorithm.

5.1 Soundness for SAS$^+$-IA Instances

This section proves that the *Plan* algorithm is sound. That is, if the algorithm has managed to produce a partially ordered plan Δ for a SAS$^+$-IA instances $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$, then every topological sort ω of the result Δ satisfies *Valid*(ω, s_0, s_*). We begin by showing correctness of *Shortest-Path*.

Lemma 5.1.1 Let $v \in \mathcal{V}$, $x, y \in \mathcal{D}_v^+$ and let $X \subseteq \mathcal{R}_v^{\mathcal{O}}$. Then, *Shortest-Path*$(v, x, y, X)$ generates a shortest path from x to y via X in G_v.

Proof: By restriction A, there exists a unique ordering of the values in X such that we can find a path from x to y via X. Lines 2-9 of the *Shortest-Path* algorithm finds this ordering or reports that there does not exist any path from x to y via X. Lines 11-12 build the required path by finding a shortest path between the intermediate values and concatenate these in a path ω. Since there exists a unique ordering of the intermediate values and we know that we must achieve them all, ω is a shortest path from x to y via X in G_v. □

We will occasionally need the following two lemmata concerning irreplaceable operators.

Lemma 5.1.2 Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ be a SAS$^+$ instance. If $o \in \mathcal{O}$ is an irreplaceable operator, then there cannot exist any $v \in \mathcal{V}$ such that $\mathsf{pre}(o)[v] = \mathsf{u}$ and $\mathsf{post}(o)[v] = y \neq \mathsf{u}$.

Proof: Assume there exists such a variable v. As a consequence, there exists an arc labelled o from y to y in G_v. The removal of this arc cannot split any component in G_v into two components, which contradicts the definition of irreplaceability. □

Lemma 5.1.3 Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ be a SAS$^+$ instance and let $\Theta = \langle \mathcal{B}, \prec \rangle$ be a plan solving Π. If $o \in \mathcal{O}$ is an irreplaceable operator, then there cannot exist more than one action of type o in $\mathcal{B}$.

Proof: Immediate by irreplaceability and the previous lemma. □

We continue by defining *separability.*

Definition 5.1.4 Let Π be a SAS$^+$instance. A p.o. plan $\Delta = \langle \mathcal{A}, \prec \rangle$ is **separable wrt.** Π iff for each topological sort $\omega = \langle \mathcal{A}, \prec' \rangle$ of Δ and for each $v \in \mathcal{V}$, the following holds

- if there exists at least one $a \in \mathcal{A}$ such that $\mathsf{prv}(a)[v] \neq \mathsf{u}$ then for each such a there exist two paths in G_v α from $s_0[v]$ to $\mathsf{prv}(a)[v]$ and β from $\mathsf{prv}(a)[v]$ to $s_*[v]$, such that $Last(\alpha) \prec a \prec First(\beta)$ and $Paths(v, \langle \mathcal{A}, \prec' \rangle) = \{(\alpha; \beta)\}$.
- if there does not exist any $a \in \mathcal{A}$ such that $\mathsf{prv}(a)[v] \neq \mathsf{u}$ then $Paths(v, \mathcal{A}, \prec')$ is a path in G_v from $s_0[v]$ to $s_*[v]$.

We can now show that if a p.o. plan is separable, then it solves Π.

Theorem 5.1.5 A plan $\Delta = \langle \mathcal{A}, \prec \rangle$ for a SAS^+ problem instance Π solves Π if Δ is separable wrt. Π.

Proof: Let $\alpha = \langle a_1, \ldots, a_{|\mathcal{A}|} \rangle$ be an arbitrary topological sort of $\mathcal{A}$ which is consistent with $\prec$. Since Δ is a p.o. plan, at least one such α exists. We prove in two steps that $Valid(\alpha, s_0, s_*)$ holds. First we show that $Valid(\alpha, s_0, \perp)$

holds and then we show that $s_* \sqsubseteq result(s_0, \alpha)$. We show $Valid(\alpha, s_0, \perp)$ by induction over all initial sequences of α:

Basis step: The empty sequence $\langle\rangle$ trivially satisfies $Valid(\langle\rangle, s_0, \perp)$.

Induction hypothesis: Suppose $\alpha\backslash k$ satisfies $Valid(\alpha\backslash k, s_0, \perp)$ for some $k \geq 1$.

Induction step: Prove that $Valid(\alpha/k, s_0, \perp)$, *ie.*, prove that $\mathsf{pre}(a_k)[v] \sqcup \mathsf{prv}(a_k)[v] \sqsubseteq result(s_0, \alpha\backslash k)[v]$ for all $v \in \mathcal{V}$. The case where $\mathsf{pre}(a_k)[v] \sqcup \mathsf{prv}(a_k)[v] = \mathsf{u}$ is trivial so assume $\mathsf{pre}(a_k)[v] \neq \mathsf{u}$ or $\mathsf{prv}(a_k)[v] \neq \mathsf{u}$. For every $v \in \mathcal{V}$, let α_v denote the unique member of $Paths(v, \alpha)$.

1. Suppose $\mathsf{pre}(a_k)[v] \neq \mathsf{u}$. Consequently, $\mathsf{post}(a_k)[v] \neq \mathsf{u}$ so $a_k \in \alpha_v$. As Δ is separable, there exists some non-empty path $\beta = \langle a_{i_1}, \ldots, a_{i_p}\rangle$, which is a subsequence of α, from $s_0[v]$ to $s_*[v]$ in G_v such that $\alpha_v = \beta$. Hence, $k = i_r$ for some r such that $1 \leq r \leq p$. First suppose $r = 1$. Then $\mathsf{pre}(a_k)[v] = s_0[v]$, so $a_\ell \notin \alpha_v$ for $\ell < k = i_1$. Since no action in α that is ordered before a_k changes the value of v, we see that $result(s_0, \alpha\backslash k)[v] = s_0[v]$ and, hence, $\mathsf{pre}(a_k)[v] \sqsubseteq result(s_0, \alpha\backslash k)[v]$. Instead suppose $1 < r \leq p$. Then $\mathsf{pre}(a_k)[v] \sqsubseteq \mathsf{post}(a_{i_{r-1}})[v]$ since $k = i_r$ and β is a path in G_v. Obviously, $a_\ell \notin \alpha_v$ for $i_{r-1} < \ell < i_r = k$, so it follows from the induction hypothesis that $result(s_0, \alpha\backslash k)[v] = result(s_0, \alpha/i_{r-1})[v] = \mathsf{post}(a_{i_{r-1}})[v]$ and, hence, $\mathsf{pre}(a_k)[v] \sqsubseteq result(s_0, \alpha\backslash k)[v]$. In either case we have $\mathsf{pre}(a_k)[v] \sqsubseteq result(s_0, \alpha\backslash k)[v]$.

2. Suppose $\mathsf{prv}(a_k)[v] \neq \mathsf{u}$. As Δ is separable, there exist a path $\beta = \langle a_{i_1}, \ldots, a_{i_p}\rangle$ in G_v from $s_0[v]$ to $\mathsf{prv}(a_k)[v]$ and $\gamma = \langle a_{i_{p+1}}, \ldots, a_{i_q}\rangle$ from $\mathsf{prv}(a_k)[v]$ to $s_*[v]$ in Δ such that $\alpha_v = (\beta;\gamma)$ and $Last(\beta) \prec a_k \prec First(\gamma)$. Hence $a_\ell \notin \alpha_v$ for all ℓ such that $i_p < \ell < k$, so it follows from the induction hypothesis that $result(s_0, \alpha\backslash k)[v] = result(s_0, \alpha/i_p)[v] = \mathsf{post}(a_{i_p})[v] = \mathsf{prv}(a_k)[v]$.

By the previous induction, we see that $\mathsf{pre}(a_k)[v] \sqsubseteq result(s_0, \alpha\backslash k)[v]$ and $\mathsf{prv}(a_k)[v] \sqsubseteq result(s_0, \alpha\backslash k)[v]$ for $1 \leq k \leq |\mathcal{A}|$ so $Valid(\alpha, s_0, \perp)$ holds. Since α is an arbitrary topological sort of $\mathcal{A}$ which is consistent with $\prec$, Δ solves the instance $\langle \mathcal{V}, \mathcal{O}, s_0, \perp\rangle$. We continue by proving that $s_*[v] \sqsubseteq result(s_0, \alpha)[v]$ holds for all $v \in \mathcal{V}$. As Δ is separable, α_v is a path in G_v from $s_0[v]$ to $s_*[v]$. If $\alpha_v = \langle\rangle$, we see immediately that $result(s_0, \alpha)[v] = s_0[v]$. Furthermore, $s_*[v] \sqsubseteq s_0[v]$ since $\langle\rangle$ is a path from $s_0[v]$ to $s_*[v]$ in G_v. This gives that $s_*[v] \sqsubseteq result(s_0, \alpha)[v]$. On the contrary, suppose $\alpha_v = \langle a_{i_1}, \ldots, a_{i_k}\rangle$. We know that $Valid(\alpha, s_0, \perp)$ holds so we have $result(s_0, \alpha/i_k)[v] = \mathsf{post}(a_{i_k})[v]$ since a_{i_k} affects variable v. Furthermore, $a_\ell \notin \alpha_v$ for $\ell \geq i_k$ and thus $result(s_0, \alpha)[v] = result(s_0, \alpha/i_k)[v] = \mathsf{post}(a_{i_k})[v]$. Hence, $s_*[v] \sqsubseteq result(s_0, \alpha)[v]$ since α_v is a path in G_v from $s_0[v]$ to $s_*[v]$. In either case, $s_*[v] \sqsubseteq result(s_0, \alpha)[v]$. Since α and v were chosen arbitrarily, the result holds for all α and v. Hence, we have shown $Valid(\alpha, s_0, s_*)$ and consequently, Δ solves Π.

□

Henceforth, we assume that $\Delta = \langle \mathcal{A}, \prec \rangle$ is the result of applying *Plan* on some arbitrary SAS$^+$-IA problem instance Π. Below we prove that Δ is a separable plan.

Lemma 5.1.6 Δ is a plan.

Proof: Immediate from the algorithm *Plan* and the fact that it did not fail in line 18. □

Lemma 5.1.7 Δ is separable wrt. Π.

Proof: Let δ be an arbitrary topological sort of Δ and let δ_{v_i} denote the unique member of $Paths(v_i, \delta)$.

We begin by showing that δ_{v_i} is a path in G_{v_i} from $s_0[v_i]$ to $s_*[v_i]$ for every $v_i \in \mathcal{V}$. The operator sequence ω_i created in lines 3-5 is obviously a path in G_{v_i} from $s_0[v_i]$ to $s_*[v_i]$. Clearly, the only potential problem that can occur is the existence of a non-unary operator o in some ω_j, $1 \leq j \leq m$ that affects v_i. For every v_j that o affects, it occurs exactly once in ω_j since o is irreplaceable and lines 3-5 were successfully completed. Hence, *Instantiate* can correctly collect the non-unary operators and collapse them into one unique action. Thus, δ_{v_i} is a path in G_{v_i} from $s_0[v_i]$ to $s_*[v_i]$ for every $v_i \in \mathcal{V}$.

We have to show that for each $v \in \mathcal{V}$ and every $a \in \mathcal{A}$ such that $\mathsf{prv}(a)[v] \neq \mathsf{u}$, there exist two paths in G_v, α from $s_0[v]$ to $\mathsf{prv}(a)[v]$ and β from $\mathsf{prv}(a)[v]$ to $s_*[v]$ such that $Last(\alpha) \prec a \prec First(\beta)$ and $\delta_v = (\alpha; \beta)$. Choose an arbitrary action $a \in \mathcal{A}$ and a state variable $v_i \in \mathcal{V}$ such that $\mathsf{prv}(a)[v_i] \neq \mathsf{u}$. We know there is a path in G_{v_i} from $s_0[v_i]$ to $s_*[v_i]$. Furthermore, since the **repeat** loop succeeded, $\mathsf{prv}(a)[v_i]$ is achieved by some operator in ω_i or $s_0[v_i] = \mathsf{prv}(a)[v_i]$ because $\mathsf{prv}(a)[v_i]$ must have been added to X_i by some *Extend* call. Hence, we can split the instantiated version α_i of ω_i, into two paths in G_{v_i}, namely β_i from $s_0[v_i]$ to $\mathsf{prv}(a)[v_i]$ and

γ_i from $\mathsf{prv}(a)[v_i]$ to $s_*[v_i]$. This means that a will be ordered such that $Last(\beta_i) \prec a \prec First(\gamma_i)$ by line 11-16 in the algorithm. Finally, we know that $(\beta_i; \gamma_i) = \alpha_i = \delta_{v_i}$ and, consequently, Δ is separable. □

Finally, we can show that *Plan* is a sound algorithm for SAS$^+$-IA instances:

Theorem 5.1.8 The Plan algorithm is sound for SAS$^+$-IA instances.

Proof: Immediate from the fact that Δ is the result of a successful application of the algorithm on an arbitrarily chosen SAS$^+$-IA problem instance and Lemmata 5.1.6, 5.1.7 and Theorem 5.1.5 □

5.2 Minimality for SAS$^+$-IAO Instances

Showing that the result Δ of applying *Plan* to Π is minimal (in the sense of containing as few actions as possible) requires two parts: A closer study of the structure of minimal plans solving SAS$^+$-IAO instances and some reasoning about fixpoints. We begin by showing some structural properties of minimal plans and continue by investigating the properties of *Extend*, leading to Theorem 5.2.13 which states that every minimal plan solving Π must achieve at least those requestable values that Δ achieves. Minimality follows trivially in the concluding Theorem 5.2.14.

Let $\Delta = \langle \mathcal{A}, \prec \rangle$ be the result of applying *Plan* to some arbitrary SAS$^+$-IAO problem instance $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$. Let $\mathcal{I}(\mathcal{O}')$ denote the set of non-unary operators (or actions) in the operator (or action) set $\mathcal{O}'$ and define the u-*filtered union* $\hat{\cup}$ s.t.

$$X \hat{\cup} Y = (X \cup Y) - \{\mathsf{u}\}$$

for all sets of values X, Y. In order to simplify the forthcoming proofs, we also define the *kernel* of a plan:

Definition 5.2.1 Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ be a SAS$^+$problem instance and let $\Theta = \langle \mathcal{A}, \prec \rangle, \Theta' = \langle \mathcal{A}', \prec' \rangle$ be partially ordered plans over Π. The function $Ker : 2^{\mathcal{O}} \rightarrow 2^{\mathcal{D}_{v_1}} \times \ldots \times 2^{\mathcal{D}_{v_m}}$ is defined as

$$Ker(\mathcal{A}) = \langle \mathcal{R}^{\mathcal{A}}_{v_1}, \ldots, \mathcal{R}^{\mathcal{A}}_{v_m} \rangle$$

By writing $Ker(\langle \mathcal{A}, \prec \rangle)$, we mean $Ker(\mathcal{A})$. $Ker(\Theta)$ is called the **kernel of** Θ. The relation $Ker(\Theta) \leq_{Ker} Ker(\Theta')$ holds iff $Ker(\Theta)[i] \subseteq Ker(\Theta')[i]$ for $1 \leq i \leq m$.

Lemma 5.2.2 Arbitrarily choose $v \in \mathcal{V}$ and $Z, Y \subseteq \mathcal{R}_v^{\mathcal{O}}$ such that $Z \subseteq Y$. Choose $x, y \in \mathcal{D}_v^+$ and let ω be a shortest path from x to y via Z and let ψ be a shortest path from x to y via Y. Then the following holds:

1. $\mathcal{I}(\omega) \subseteq \mathcal{I}(\psi)$
2. If $Z = Y$, then $\mathcal{I}(\omega) = \mathcal{I}(\psi)$.

Proof:

1. Assume o is a non-unary operator such that $o \in \omega$ but $o \notin \psi$. Suppose $y \neq \mathsf{u}$. Then there exists a path from x to y that includes o and another path that does not include o. Clearly, this contradicts irreplaceability so $y = \mathsf{u}$. Let $p = \mathsf{post}(Last(\omega))[v]$. Since ω is a shortest path, $p \in Z$ and consequently $p \in Y$. Then there exists a path from x to p that includes o and another path from x to p that does not include o which contradicts irreplaceability. Since both cases lead to a contradiction, $\mathcal{I}(\omega) \subseteq \mathcal{I}(\psi)$.
2. If $Z = Y$ then $Z \subseteq Y$, $Y \subseteq Z$ so $\mathcal{I}(\omega) \subseteq \mathcal{I}(\psi)$, $\mathcal{I}(\psi) \subseteq \mathcal{I}(\omega)$ and $\mathcal{I}(\omega) = \mathcal{I}(\psi)$ follows. □

We begin by showing that there exists a normal form (the $\mathcal{R}$-unique form) of every plan solving Π.

Definition 5.2.3 A plan $\Theta = \langle \mathcal{B}, \prec \rangle$ is **$\mathcal{R}$-unique** iff for every $v \in \mathcal{V}$,

1. $s_0[v] \notin \mathcal{R}_v^{\mathcal{B}}$ and for every x in $\mathcal{R}_v^{\mathcal{B}}$ there exists exactly one action $b \in \mathcal{B}$ such that $\mathsf{post}(b)[v] = x$
2. $s_0[v] \in \mathcal{R}_v^{\mathcal{B}}$ and for every x in $\mathcal{R}_v^{\mathcal{B}} - \{s_0[v]\}$ there exists exactly one action $b \in \mathcal{B}$ such that $\mathsf{post}(b)[v] = x$ and there exists no action $c \in \mathcal{B}$ such that $\mathsf{post}(c)[v] = s_0[v]$.

Lemma 5.2.4 If $\Theta = \langle \mathcal{B}, \prec \rangle$ is a minimal plan solving Π then Θ is $\mathcal{R}$-unique.

Proof: Arbitrarily choose $v \in \mathcal{V}$. We have two cases to consider, depending on whether $s_0[v] \in \mathcal{R}_v^{\mathcal{B}}$ or not.

Case 1: $s_0[v] \notin \mathcal{R}_v^{\mathcal{B}}$: Let $b \in \mathcal{B}$ be an action such that $\mathsf{post}(b)[v] \in \mathcal{R}_v^{\mathcal{B}}$ for some $v \in \mathcal{V}$. We must show that b is unique. Suppose there exists two actions $b_1, b_2 \in \mathcal{B}$ such that $z = \mathsf{post}(b_1)[v] = \mathsf{post}(b_2)[v] \in \mathcal{R}_v^{\mathcal{B}}$. Assume without loss of generality that $b_1 \prec b_2$. By restriction A, there cannot exist any action $b' \in \mathcal{B}$ such that $b_1 \prec b' \prec b_2$, $\mathsf{post}(b')[v] \in \mathcal{R}_v^{\mathcal{B}}$ and $\mathsf{post}(b')[v] \neq z$. Hence, every action a affecting v, ordered after b_1 and before b_2 is unary because otherwise we would have a cycle in G_v involving at least two requestable values. Hence, if $\mathsf{post}(a)[v] \in \mathcal{R}_v^{\mathcal{B}}$ then $\mathsf{post}(a)[v] = z$. Consequently, we can remove all actions affecting v after b_1 and before (and including) b_2, reorder all actions $c \in \mathcal{B}$ that has z as pre- or prevail-condition such that $b_1 \prec^- c \prec^- d$ for all actions d affecting v such that $b_1 \prec d$. (If such actions exists.) After having done this, we have a valid plan but with a strictly smaller number of actions than Θ, thus contradicting the minimality of Θ.

Case 2: $z = s_0[v] \in \mathcal{R}_v^{\mathcal{B}}$: We begin by showing that there does not exist any action $c \in \mathcal{B}$ such that $\mathsf{post}(c)[v] = z$. Suppose there exists an action $c \in \mathcal{B}$ such that $\mathsf{post}(c)[v] = z$. By restriction A, there cannot exist any action $c' \in \mathcal{B}$ such that $c' \prec c$, $\mathsf{post}(c')[v] \in \mathcal{R}_v^{\mathcal{B}}$ and $\mathsf{post}(c')[v] \neq z$. Hence, every action a affecting v, ordered before c is unary because otherwise we would have a cycle in G_v involving at least two requestable values. Hence, if $\mathsf{post}(a)[v] \in \mathcal{R}_v^{\mathcal{B}}$ then $\mathsf{post}(a)[v] = z$. Consequently, we can remove all actions affecting v that are ordered before (and including) c, reorder all actions $a \in \mathcal{B}$ that has z as pre- or prevail-condition such that $a \prec^- d$ for all actions d affecting v. (If such actions exists.) After having done this, we have a valid plan but with a strictly smaller number of actions than Θ, thus contradicting the minimality of Θ.

To show that there exists exactly one action $b \in \mathcal{B}$ such that $\mathsf{post}(b)[v] = x$ for every x in $\mathcal{R}_v^{\mathcal{B}} - \{z\}$ is similar to the previous case. Since one of these two cases always holds, Θ is $\mathcal{R}$-unique. $\square$

Note that the previous lemma holds for arbitrary SAS$^+$-A instances. $\mathcal{R}$-uniqueness leads us to a method for decomposing minimal plans into subsequences with certain properties, which forms the basis for the results in both this and the next section.

Lemma 5.2.5 Let $\Theta = \langle \mathcal{B}, \prec_\Theta \rangle$ be an arbitrary minimal plan solving Π, let θ be an arbitrary topological sort of Θ and let θ_i denote the single member of $Paths(v_i, \theta)$ for $1 \leq i \leq m$. Given a subset R of $\mathcal{R}_{v_i}^{\mathcal{B}}$, there exists a unique subdivision $\langle \beta_0^i, \beta_1^i, \ldots, \beta_r^i \rangle$ of θ_i such that $\theta_i = (\beta_0^i; \beta_1^i; \ldots; \beta_r^i)$ and

- if $s_0[v_i] \notin R$ then $\mathsf{post}(Last(\beta_j^i))[v_i] \in R$ for all $0 \leq j \leq |R| - 1$
- if $s_0[v_i] \in R$ then $\mathsf{post}(Last(\beta_j^i))[v_i] \in R - \{s_0[v]\}$ for all $0 \leq j \leq |R| - 2$.

Proof: We know that θ_i is a path in G_{v_i} from $s_0[v_i]$ to $s_*[v_i]$ via $\mathcal{R}^{\mathcal{B}}_{v_i}$. Since $R \subseteq \mathcal{R}^{\mathcal{B}}_{v_i}$, we can divide θ_i as described. By restriction A, the values in R can be achieved in one unique order only. As we know that Δ is $\mathcal{R}$-unique, the subdivision is unique. □

The previous lemma leads to the following definition.

Definition 5.2.6 Let $\Theta = \langle \mathcal{B}, \prec_\Theta \rangle$ be an arbitrary minimal plan solving Π, let θ be an arbitrary topological sort of Θ and let θ_i denote the single member of $Paths(v_i, \theta)$ for $1 \leq i \leq m$. Then, for $R \subseteq \mathcal{R}^{\mathcal{B}}_{v_i}$, $\theta_i // R$ denotes the unique subdivision $\langle \beta^i_0, \beta^i_1, \ldots, \beta^i_r \rangle$ such that $\theta_i = (\beta^i_0; \beta^i_1; \ldots; \beta^i_r)$ and

- if $s_0[v_i] \notin R$ then $\mathsf{post}(Last(\beta^i_j))[v_i] \in R$ for all $0 \leq j \leq |R| - 1$,
- if $z = s_0[v_i] \in R$ then $\mathsf{post}(Last(\beta^i_j))[v_i] \in R - \{z\}$ for all $0 \leq j \leq |R| - 2$.

It is now possible to show an important structural property of minimal plans solving SAS$^+$-IAO instances.

Lemma 5.2.7 Let $\Theta = \langle \mathcal{B}, \prec_\Theta \rangle$ be an arbitrary minimal plan solving Π, let θ be an arbitrary topological sort of Θ and let θ_i denote the single member of $Paths(v_i, \theta)$ for $1 \leq i \leq m$. Then, θ_i is a shortest path from $s_0[v_i]$ to $s_*[v_i]$ via $\mathcal{R}^{\mathcal{B}}_{v_i}$ in G_{v_i}.

Proof: By Lemma 4.1.1, θ_i is a path from $s_0[v_i]$ to $s_*[v_i]$ via $\mathcal{R}^{\mathcal{B}}_{v_i}$ in G_{v_i}. Let $\theta_i // \mathcal{R}^{\mathcal{B}}_{v_i} = \langle \beta_0, \ldots, \beta_k \rangle$. Assume θ_i is not a shortest path and assume $s_*[v_i] \neq \mathsf{u}$. (The case when $s_0[v_i] = \mathsf{u}$ is analogous.)

Then there exists some p, $0 \leq p \leq k$ such that $\beta_p = \langle o_1, \ldots, o_q \rangle$ is not a shortest path from s to t in G_{v_i} where s is the value of v_i that β_p is applied to and t is the resulting value. Clearly, there exists a minimal path $\gamma = \langle o'_1, \ldots, o'_r \rangle$ from s to t in G_{v_i} and, by prevail-order-preservation, $\gamma \trianglelefteq \beta_p$. Consequently, we can substitute γ for β_p in θ_i and reorder the actions in $\mathcal{B}$ such that they satisfy the prevail-conditions of the actions in γ. This reordering is trivial since there exists a subsequence $\langle o_{j_1}, \ldots, o_{j_r} \rangle$ of β_i such that $\mathsf{prv}(o'_1) \sqsubseteq \mathsf{prv}(o_{j_1}), \ldots, \mathsf{prv}(o'_r) \sqsubseteq \mathsf{prv}(o_{j_r})$ by restriction O. After having done this, we have a valid plan but with a strictly smaller number of actions than Θ, thus contradicting the minimality of Θ. Hence, θ_i is a shortest path from $s_0[v_i]$ to $s_*[v_i]$ via $\mathcal{R}^{\mathcal{B}}_{v_i}$ in G_{v_i}. □

Next we prove that kernels of minimal plans are fixpoints of *Extend*.

Theorem 5.2.8 Let $\Theta = \langle \mathcal{B}, \prec \rangle$ be a minimal plan solving Π. Then $Extend(Ker(\Theta)) = Ker(\Theta)$.

Proof: Let $Ker(\Theta) = Z = \langle Z_1, \ldots, Z_m \rangle$. Choose i arbitrarily between 1 and m. Let θ be an arbitrary topological sort of Θ and let θ_i denote the single member of $Paths(v_i, \theta)$. Now apply $Extend$ to Z. The path ω_i computed in line 4 of $Extend$ is a shortest path from $s_0[v_i]$ to $s_*[v_i]$ via Z_i. By Lemma 5.2.7, θ_i is also a shortest path from $s_0[v_i]$ to $s_*[v_i]$ via Z_i. Hence, $\omega_i \triangleq \theta_i$, so for $1 \le j \le m$,

$$\widehat{\bigcup}_{o \in \omega_i} \{\mathsf{prv}(o)[v_j]\} = \widehat{\bigcup}_{o \in \theta_i} \{\mathsf{prv}(o)[v_j]\}. \tag{5.1}$$

By Lemma 5.2.2, $\mathcal{I}(\omega_j) = \mathcal{I}(\theta_j)$ and consequently

$$\widehat{\bigcup}_{o \in \mathcal{I}(\omega_i)} \{\mathsf{pre}(o)[v_j], \mathsf{post}(o)[v_j]\} = \widehat{\bigcup}_{o \in \mathcal{I}(\theta_i)} \{\mathsf{pre}(o)[v_j], \mathsf{post}(o)[v_j]\}. \tag{5.2}$$

For $1 \le j \le m$, the value of X_j returned from $Extend(Ker(\Theta))$ equals

$$\mathcal{R}^{\mathcal{B}}_{v_j} \cup \bigcup_{1 \le i \le m} \widehat{\bigcup}_{o \in \omega_i} \{\mathsf{prv}(o)[v_j]\} \cup \widehat{\bigcup}_{o \in \mathcal{I}(\omega_i)} \{\mathsf{pre}(o)[v_j], \mathsf{post}(o)[v_j]\}.$$

But by the definition of requestable value

$$\mathcal{R}^{\mathcal{B}}_{v_j} = \bigcup_{1 \le i \le m} \widehat{\bigcup}_{o \in \theta_i} \{\mathsf{prv}(o)[v_j]\} \cup \widehat{\bigcup}_{o \in \mathcal{I}(\theta_i)} \{\mathsf{pre}(o)[v_j], \mathsf{post}(o)[v_j]\}.$$

Hence, by equations (1) and (2),

$$X_j = \bigcup_{1 \le i \le m} \widehat{\bigcup}_{o \in \theta_i} \{\mathsf{prv}(o)[v_j]\} \cup \widehat{\bigcup}_{o \in \mathcal{I}(\theta_i)} \{\mathsf{pre}(o)[v_j], \mathsf{post}(o)[v_j]\}.$$

Consequently, for all $1 \le j \le m$, $Extend(Ker(\Theta))[j] = \mathcal{R}^{\mathcal{B}}_{v_j} = Ker(\Theta)[j]$ so $Extend(Ker(\Theta)) = Ker(\Theta)$. $\Box$

We can now prove that *Extend* is a monotonic function on $2^{\mathcal{D}_{v_1}} \times \ldots \times 2^{\mathcal{D}_{v_m}}$. To simplify the proof, we first show that every SAS$^+$-O instance is *prevail-monotonic.*

Definition 5.2.9 A SAS$^+$instance Π is **prevail-monotonic** iff for all $v \in \mathcal{V}$, all $x, y \in \mathcal{D}_v^+$, all $X \subseteq \mathcal{D}_v$ and all $\omega = \langle o_1, \ldots, o_m \rangle$, $\omega' = \langle o'_1, \ldots, o'_n \rangle \in Seqs(\mathcal{O})$, if ω is a shortest path from x to y via X and ω' is a path from x to y via X, then $(\widehat{\bigcup}_{o \in \omega} \mathsf{prv}(o)[v']) \subseteq (\widehat{\bigcup}_{o' \in \omega'} \mathsf{prv}(o')[v'])$ for all $v' \in \mathcal{V}$.

Lemma 5.2.10 Every SAS$^+$-O instance is prevail-monotonic.

Proof: Immediate from the definitions of prevail-monotonicity and prevail-order-preservation. □

Lemma 5.2.11 Let Θ and Θ' be plans over Π. If $Ker(\Theta) \leq_{Ker} Ker(\Theta')$, then $Extend(Ker(\Theta)) \leq_{Ker} Extend(Ker(\Theta'))$.

Proof: Let $Ker(\Theta) = Z = \langle Z_1, \ldots, Z_m \rangle$ and $Ker(\Theta') = Y = \langle Y_1, \ldots, Y_m \rangle$. We first determine the value of $Extend(Ker(\Theta))$. Let $\omega_1, \ldots, \omega_m$ be the shortest paths in G_{v_i} via Z_i for $1 \leq i \leq m$ as computed by line 3-5 of the *Extend* procedure. Let $\overline{Z}^+ = \langle Z_1^+, \ldots, Z_m^+ \rangle$ be the values of all X_i variables after the completion of line 6-8 when *Extend* is called with Z. We see that for $1 \leq i \leq m$:

$$Z_i^+ = Z_i \cup \bigcup_{1 \leq j \leq m} \widehat{\bigcup}_{o \in \omega_j} \{\mathsf{prv}(o)[v_i]\}$$

Further, let $\overline{Z}^{++} = \langle Z_1^{++}, \ldots, Z_m^{++} \rangle$ be the values of all X_i after the completion of line 9-11. Then, for all $1 \leq i \leq m$:

$$Z_i^{++} = Z_i^+ \cup \bigcup_{1 \leq j \leq m} \widehat{\bigcup}_{o \in \mathcal{I}(\omega_j)} \{\mathsf{pre}(o)[v_i], \mathsf{post}(o)[v_i]\}$$

In the same manner, we determine the value of $Extend(Ker(\Theta'))$: Let $\psi_1, \ldots, \psi_m$ be the shortest paths in G_{v_i} via Y_i ($1 \leq i \leq m$) computed by line 3-5 of the *Extend* procedure. Let $\overline{Y}^+ = \langle Y_1^+, \ldots, Y_m^+ \rangle$ be the values of all X_i after the completion of line 6-8. For all $1 \leq i \leq m$:

$$Y_i^+ = Y_i \cup \bigcup_{1 \leq j \leq m} \widehat{\bigcup}_{p \in \psi_j} \{\mathsf{prv}(p)[v_i]\}$$

Let $\overline{Y}^{++} = \langle Y_1^{++}, \ldots, Y_m^{++} \rangle$ be the values of all X_i after the completion of line 9-11. For all $1 \leq i \leq m$:

$$Y_i^{++} = Y_i^+ \cup \bigcup_{1 \leq j \leq m} \widehat{\bigcup}_{p \in \mathcal{I}(\psi_j)} \{\mathsf{pre}(p)[v_i], \mathsf{post}(p)[v_i]\}$$

Select an arbitrary $v_i \in \mathcal{V}$. By our premises, $Z_i \subseteq Y_i$ and ω_i is a shortest path in G_{v_i} from $s_0[v_i]$ to $s_*[v_i]$ via Z_i and ψ_i is a shortest path in G_{v_i} from $s_0[v_i]$ to $s_*[v_i]$ via Y_i so by prevail-monotonicity,

$$\bigcup_{1 \leq j \leq m} \widehat{\bigcup}_{o \in \omega_j} \{\mathsf{prv}(o)[v_i]\} \subseteq \bigcup_{1 \leq j \leq m} \widehat{\bigcup}_{p \in \psi_j} \{\mathsf{prv}(p)[v_i]\}$$

and hence $Z_i^+ \subseteq Y_i^+$. Since $Z_i \subseteq Y_i$, we get $\mathcal{I}(\omega_j) \subseteq \mathcal{I}(\psi_j)$ for $1 \leq j \leq m$ by Lemma 5.2.2 and, thus, $\bigcup_{1 \leq j \leq m} \widehat{\bigcup}_{o \in \mathcal{I}(\omega_j)} \{\mathsf{pre}(o)[v_i], \mathsf{post}(o)[v_i]\}$ is a subset of $\bigcup_{1 \leq j \leq m} \widehat{\bigcup}_{p \in \mathcal{I}(\psi_j)} \{\mathsf{pre}(p)[v_i], \mathsf{post}(p)[v_i]\}$. Hence, $Z_i^{++} \subseteq Y_i^{++}$ for $1 \leq i \leq m$ and, thus, $Extend(Ker(\Theta)) \leq_{Ker} Extend(Ker(\Theta'))$. $\square$

The next theorem ties together Lemma 5.2.11 and Theorem 5.2.8 in order to show that Δ is $\mathcal{R}$-minimal, to be defined below. The proof is a variant of Theorem 1 in Paige & Henglein [1987] which is derived from Tarski [1955].

Definition 5.2.12 A p.o. plan Θ solving Π is *$\mathcal{R}$-minimal* iff for every minimal plan Θ' solving Π, $Ker(\Theta) \leq_{Ker} Ker(\Theta')$.

Theorem 5.2.13 Δ is $\mathcal{R}$-minimal.

Proof: Let $\Theta = \langle \mathcal{B}, \prec_\Theta \rangle$ be an arbitrary minimal plan solving Π. From the algorithm, we know that $Ker(\Delta) = Extend^n(\langle \varnothing, \ldots, \varnothing \rangle)$ for some n. We show that $Extend^n(\langle \varnothing, \ldots, \varnothing \rangle) \leq_{Ker} Ker(\Theta)$ for all $n \geq 0$ by induction over n.
Basis step: If $n = 0$, the claim holds trivially.
Induction hypothesis: Suppose $Extend^k(\langle \varnothing, \ldots, \varnothing \rangle) \leq_{Ker} Ker(\Theta)$ for some $k \geq 0$.
Induction step: We have to show that the claim holds for $k + 1$. By the induction hypothesis, $Extend^k(\langle \varnothing, \ldots, \varnothing \rangle) \leq_{Ker} Ker(\Theta)$ and by Lemma 5.2.11, $Extend(Extend^k(\langle \varnothing, \ldots, \varnothing \rangle)) \leq_{Ker} Extend(Ker(\Theta))$. Furthermore, $Extend(Ker(\Theta)) = Ker(\Theta)$ by Theorem 5.2.8 so $Extend^{k+1}(\langle \varnothing, \ldots, \varnothing \rangle) \leq_{Ker} Ker(\Theta)$. $\square$
The minimality of Δ follows immediately.

Theorem 5.2.14 Δ is minimal.

Proof: Choose $v_i \in \mathcal{V}$ arbitrarily. By Lemma 5.2.13, for every minimal plan Θ solving Π, every path in $Paths(v_i, \Theta)$ must pass every state in $\mathcal{R}_{v_i}^{\mathcal{A}}$. Since Δ contains a shortest such path in G_{v_i} and v_i was chosen arbitrarily, minimality follows. $\square$

5.3 Completeness for SAS$^+$-IAO Instances

We have to show that if *Plan* fails when attempting to solve a SAS$^+$-IAO instance Π, then there cannot exist any plan solving Π. From the definition of *Plan*, it is obvious that *Plan* can fail in only two cases: In the *Extend* procedure and in the irreflexivity test. In Lemma 5.3.10, we show the existence of a structure-preserving function φ from Δ to every other solution for Π. Hence, if Δ is not irreflexive, then there cannot exist any plan solving Π. With this in mind, we show that *Plan* is complete for SAS$^+$-IAO instances in Theorem 5.3.12.

Let $\Delta = \langle \mathcal{A}, \prec \rangle$ be the result of applying the *Plan* algorithm to some SAS$^+$-IAO instance $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$. Assume we have an arbitrary minimal plan $\Theta = \langle \mathcal{B}, \prec_\Theta \rangle$ solving Π. Furthermore, let δ be an arbitrary topological sort of Δ and let θ be an arbitrary topological sort of Θ. For each $1 \leq i \leq m$, let δ_i denote the single member of $Paths(v_i, \delta)$ and let θ_i denote the single member of $Paths(v_i, \theta)$. Let δ_i^k and θ_i^k denote the k:th action in δ_i and θ_i respectively. In order to define φ, we first have to establish some further properties of minimal plans solving Π.

Lemma 5.3.1 Δ and Θ are $\mathcal{R}$-unique.

Proof: Π satisfies restriction A and both Δ and Θ are minimal solutions of Π (Δ is minimal by Theorem 5.2.14 and Θ is minimal by assumption). Hence, by Lemma 5.2.4, Δ and Θ are $\mathcal{R}$-unique. $\square$

Lemma 5.3.2 $Ker(\Delta) = Ker(\Theta)$

Proof: By Theorem 5.2.13, we know that $Ker(\Delta) \leq_{Ker} Ker(\Theta)$. Assume there exists an x such that $x \in Ker(\Theta)[i]$ but $x \notin Ker(\Delta)[i]$. We have two cases:

1. There exists an action $b \in \mathcal{B}$ such that $\mathsf{prv}(b)[v_i] = x$. Assume b affects the variable v_j. Then θ_j (which is a path in G_{v_j} from $s_0[v_j]$ to $s_*[v_j]$ via $\mathcal{R}_{v_j}^{\mathcal{A}}$ since $\mathcal{R}_{v_j}^{\mathcal{A}} \subseteq \mathcal{R}_{v_j}^{\mathcal{B}}$) contains the action b but δ_j (which is a *shortest* path in G_{v_j} from $s_0[v_j]$ to $s_*[v_j]$ via $\mathcal{R}_{v_j}^{\mathcal{A}}$) does not contain any action having the same prevail-conditions as b. In order not to violate restriction O, $|\theta_j| > |\delta_j|$. Since Θ is a minimal plan, there must exist some k, $1 \leq k \leq m$ such that $|\theta_k| < |\delta_k|$ in order to compensate for the length of θ_j. This leads to a contradiction since

- δ_k is a shortest path in G_{v_k} from $s_0[v_k]$ to $s_*[v_k]$ via $\mathcal{R}^{\mathcal{A}}_{v_k}$ by Lemma 5.2.7,
- θ_k is a shortest path in G_{v_k} from $s_0[v_k]$ to $s_*[v_k]$ via $\mathcal{R}^{\mathcal{B}}_{v_k}$ by Lemma 5.2.7,
- $\mathcal{R}^{\mathcal{A}}_{v_k} \subseteq \mathcal{R}^{\mathcal{B}}_{v_k}$.

2. There exists a non-unary action $b \in \mathcal{B}$ such that $\mathsf{pre}(b)[v_i] = x$ or $\mathsf{post}(b)[v_i] = x$. Consequently b is a non-unary action that is not a member of $\mathcal{A}$. As $\mathcal{R}^{\mathcal{A}}_v \subseteq \mathcal{R}^{\mathcal{B}}_v, \mathcal{I}(\omega) \subseteq \mathcal{I}(\psi)$ by Lemma 5.2.2 so $|\theta_i| > |\delta_i|$. Since Θ is a minimal plan, there must exist some k, $1 \leq k \leq m$ such that $|\theta_k| < |\delta_k|$ in order to compensate for the length of θ_i. This leads to a contradiction by the same reasons as in the previous case.

In all three cases the assumption leads to a contradiction and, consequently, $Ker(\Delta) = Ker(\Theta)$. □

Lemma 5.3.3 For $1 \leq k \leq m$, $\delta_k \triangleq \theta_k$.

Proof: By Lemma 5.2.7, δ_k is a shortest path in G_{v_k} from $s_0[v_k]$ to $s_*[v_k]$ via $\mathcal{R}^{\mathcal{A}}_{v_k}$ and θ_k is a shortest path in G_{v_k} from $s_0[v_k]$ to $s_*[v_k]$ via $\mathcal{R}^{\mathcal{B}}_{v_k}$. But $\mathcal{R}^{\mathcal{A}}_{v_k} = \mathcal{R}^{\mathcal{B}}_{v_k}$ by the previous lemma so, by prevail-order-preservation, $\delta_k \trianglelefteq \theta_k$ and $\theta_k \trianglelefteq \delta_k$. By the definition of shortest-path consistency, $\delta_k \triangleq \theta_k$ follows. □

Lemma 5.3.4 $type(\mathcal{I}(\mathcal{A})) = type(\mathcal{I}(\mathcal{B}))$.

Proof: Since $Ker(\Delta) = Ker(\Theta)$, the lemma follows from Lemma 5.2.2. □

Definition 5.3.5 Let $\varphi : \mathcal{A} \to \mathcal{B}$ be defined such that:

- If a is non-unary, then $\varphi(a) = b$ where $b \in \mathcal{B}$ and $type(a) = type(b)$.
- If a is unary, then $a = \delta_i^k$ for some $1 \leq i \leq m$, $1 \leq k \leq |\delta_i|$. Let $\varphi(a) = \theta_i^k$.

Lemma 5.3.6 φ is a well-defined function from $\mathcal{A}$ to $\mathcal{B}$.

Proof: For arbitrary non-unary $a \in \mathcal{A}$, $\varphi(a)$ is uniquely determined by a by Lemma 5.3.4 and the fact that there can only be one non-unary action of each type by irreplaceability. For arbitrary unary $a \in \mathcal{A}$, $\varphi(a)$ is uniquely determined by a because, by Lemma 5.3.3, $|\delta_i| = |\theta_i|$ for $1 \leq i \leq m$. □

Lemma 5.3.7 For all $a \in \mathcal{A}$, $\mathsf{prv}(a) = \mathsf{prv}(\varphi(a))$.

Proof: If a is non-unary the result trivially follows from the construction of φ. Suppose a is unary and a affects v_i, $1 \leq i \leq m$. Then, a is a member of δ_i and $\varphi(a)$ is a member of θ_i. Furthermore, if a equals δ_i^k, then $\varphi(a)$ equals θ_i^k by the construction of φ. Obviously, $\mathsf{prv}(a) = \mathsf{prv}(\varphi(a))$ follows from the fact that $\delta_i \triangleq \theta_i$ by Lemma 5.3.3. $\square$

Lemma 5.3.8 Let $a \in \mathcal{A}$ be such that $x = \mathsf{post}(a)[v_i] \in \mathcal{R}^{\mathcal{A}}_{v_i}$. Then, $\mathsf{post}(\varphi(a))[v_i] = x$.

Proof: If a is non-unary the result trivially follows from the construction of φ. Suppose a is unary. Obviously, a is a member of δ_i and $\varphi(a)$ is a member of θ_i. Suppose a equals δ_i^k. Let R be the requestable values that are achieved by δ_i/k. Since δ_i is a shortest path from $s_0[v_i]$ to $s_*[v_i]$ via $\mathcal{R}^{\mathcal{A}}_{v_i}$ in G_{v_i}, δ_i/k is a shortest path from $s_0[v_i]$ to x via R which follows from acyclicity wrt. requestable values. Let b be the unique (by $\mathcal{R}$-uniqueness) action in θ_i that achieves x and suppose b is the l:th action in θ_i. We know that θ_i is a shortest path from $s_0[v_i]$ to $s_*[v_i]$ via $\mathcal{R}^{\mathcal{B}}_{v_i}$ in G_{v_i}, but by Lemma 5.3.2, $\mathcal{R}^{\mathcal{B}}_{v_i} = \mathcal{R}^{\mathcal{A}}_{v_i}$. Hence, by restriction A, θ_i/l is a shortest path from $s_0[v_i]$ to x via R. Since both δ_i/k and θ_i/l are shortest paths, $k = l$ and the lemma follows. $\square$

Lemma 5.3.9 If $a = \delta_i^k$ for some $1 \leq i \leq m$, $1 \leq k \leq |\delta_i|$ then $\varphi(a) = \theta_i^k$.

Proof: If a is unary then the result follows trivially from the construction of φ and the fact that $|\delta_i| = |\theta_i|$ so assume $\varphi(a)$ is non-unary. Let $x = \mathsf{post}(\varphi(a))[v_i]$ and let R be the requestable values that are achieved by δ_i/k. Since δ_i is a shortest path from $s_0[v_i]$ to $s_*[v_i]$ via $\mathcal{R}^{\mathcal{A}}_{v_i}$ in G_{v_i}, δ_i/k is a shortest path from $s_0[v_i]$ to x via R. Let b be the unique (by $\mathcal{R}$-uniqueness) action in θ_i that achieves x and suppose b is the l:th action in θ_i. We know that θ_i is a shortest path from $s_0[v_i]$ to $s_*[v_i]$ via $\mathcal{R}^{\mathcal{B}}_{v_i}$ in G_{v_i}, but by Lemma 5.3.2, $\mathcal{R}^{\mathcal{B}}_{v_i} = \mathcal{R}^{\mathcal{A}}_{v_i}$. Hence, by restriction A, θ_i/l is a shortest path from $s_0[v_i]$ to x via R. Since both δ_i/k and θ_i/l are shortest paths, $k = l$ and the lemma follows. $\square$

Lemma 5.3.10 For arbitrary $a, b \in \mathcal{A}$, if $a \prec^- b$ then $\varphi(a) \prec_\Theta \varphi(b)$.

Proof: The *Plan* algorithm orders actions in two cases: the ordering of the α_i action sequences in lines 9-10 and the prevail-condition ordering in lines 11-16.

The first case happens when $a, b \in \mathcal{A}$ and a immediately precedes b in some α_i. Consequently, $a = \delta_i^k$, $b = \delta_i^{k+1}$ for some k. By Lemma 5.3.9, $\varphi(a) = \theta_i^k$ and $\varphi(b) = \theta_i^{k+1}$ so $\varphi(a) \prec_\Theta \varphi(b)$.

The second case is slightly more complex as it has several different possibilities. Suppose $a, b, c \in \mathcal{A}$ and $\mathsf{prv}(b)[v_i] = x \neq \mathsf{u}$ for some variable v_i. The following orderings can be introduced by lines 11-16.

1. a is the unique (by $\mathcal{R}$-minimality) action such that $\mathsf{post}(a)[v_i] = x$ and c is an action such that it affects v_i and is ordered immediately after a in α_i.
2. $x = s_0[v_i]$ and c is the first action of α_i
3. $x = \mathsf{post}(a)[v_i]$ and a is the last action affecting v_i
4. there are no actions at all affecting v_i

In these cases, the actions are ordered such that

1. $a \prec^- b \prec^- c$
2. $b \prec^- c$
3. $a \prec^- b$
4. no specific ordering is introduced

We show that if the first case holds, then $\varphi(a) \prec_\Theta \varphi(b) \prec_\Theta \varphi(c)$. The three other cases are analogous. By Lemma 5.3.7, $\mathsf{prv}(\varphi(b))[v_i] = s$. By Lemma 5.3.8 and $\mathcal{R}$-minimality, $\varphi(a)$ is the only action in $\mathcal{B}$ such that $\mathsf{post}(\varphi(a))[v_i] = s$. Hence, $\varphi(a) \prec_\Theta \varphi(b)$. Furthermore, $a = \delta_i^k$ and $c = \delta_i^{k+1}$ for some k. By Lemma 5.3.9, $\varphi(a) = \theta_i^k$ and $\varphi(c) = \theta_i^{k+1}$. Hence, $\varphi(b)$ must be ordered before $\varphi(c)$ since $\varphi(c)$ affects v_i and $\varphi(a) \prec_\Theta \varphi(c)$. □

To prove that there cannot exist any solution solving Π if Δ is not irreflexive is trivial with the aid of φ:

Lemma 5.3.11 If Δ is not irreflexive, then there exists no plan solving Π.

Proof: If Δ is not irreflexive, then there exist actions $a_1, \ldots, a_n \in \mathcal{A}$ such that $a_1 \prec^- \ldots \prec^- a_n \prec^- a_1$. However, by Lemma 5.3.10, there exists a function φ from $\mathcal{A}$ to $\mathcal{B}$ such that if $a, b \in \mathcal{A}$ and $a \prec^- b$, then $\varphi(a) \prec_\Theta$

$\varphi(b)$. Hence, $\varphi(a_1), \ldots, \varphi(a_n) \in \mathcal{B}$ and $\varphi(a_1) \prec_\Theta \ldots \prec_\Theta \varphi(a_n) \prec_\Theta \varphi(a_1)$. Consequently, Θ is not a plan solving Π. The lemma follows by contradiction. $\square$

We summarize the completeness of *Plan* in the following theorem:

Theorem 5.3.12 If there exists some plan solving Π, then algorithm *Plan* returns a plan solving Π.

Proof: We prove that if the algorithm fails, then there is no plan solving Π. Suppose the algorithm fails but there exists some minimal plan $\Theta = \langle \mathcal{B}, \prec_\Theta \rangle$ solving Π. The algorithm can fail in two cases: In line 4 or in line 18. We consider the cases one at a time:

i) Assume *Plan* fails when attempting to construct a path in G_{v_i} from $s_0[v_i]$ to $s_*[v_i]$ via some set X_{v_i} of requestable values. Suppose *Plan* fails after n iterations of the **while**-loop and let X_i^n denote the value of X_i in the n:th iteration. We begin by showing that $X_i^n \subseteq \mathcal{R}_{v_i}^{\mathcal{B}}$ by induction over n.
Basis step: $X_i^1 = \varnothing$ so the claim holds trivially.
Induction hypothesis: Suppose $X_i^k \subseteq \mathcal{R}_{v_i}^{\mathcal{B}}$ for some $k \geq 1$.
Induction step: We have to show that the claim holds for $k+1$. By the induction hypothesis, $X_i^k \subseteq \mathcal{R}_{v_i}^{\mathcal{B}} = Ker(\Theta)[i]$. By the definition of the algorithm, $X_i^{k+1} = Extend(\langle X_1^k, \ldots, X_i^k, \ldots, X_m^k \rangle)[i]$. Hence, by Lemma 5.2.11,

$$Extend(\langle X_1^k, \ldots, X_i^k, \ldots, X_m^k \rangle)[i] \subseteq Extend(Ker(\Theta))[i]$$

but by Lemma 5.2.8, $Extend(Ker(\Theta)) = Ker(\Theta)$. So,

$$X_i^{k+1} = Extend(\langle X_1^k, \ldots, X_i^k, \ldots, X_m^k \rangle)[i] \subseteq Extend(Ker(\Theta))[i] = \mathcal{R}_{v_i}^{\mathcal{B}}.$$

If the algorithm fails in line 5 of the *Extend* procedure, then for some $v_i \in \mathcal{V}$ there does not exist any path in G_{v_i} from $s_0[v_i]$ to $s_*[v_i]$ via X_i. However, $X_i \subseteq \mathcal{R}_v^{\mathcal{B}}$ throughout the computation so by Theorem 5.2.13, every member of *Paths*(v_i, Θ) must pass every value in X_i which contradicts the existence of Θ.

ii) By Lemma 5.3.11, there cannot exist any plan solving Π, which contradicts the existence of Θ. $\square$

5.4 Complexity Analysis

In this section we analyse the worst-case time complexity of the *Plan* algorithm using straightforward, non-optimal assumptions about data structures. As we do not attempt proving an optimal bound on the complexity, we make the following assumptions about data representation: States are represented as arrays of values, operators as triples of states, actions as tuples of labels and operators, sets as lists and relations as adjacency lists. With this representation, we can find a shortest path between two values in a domain-transition graph G_v in $O(|\mathcal{D}_v^+||\mathcal{O}|)$ time, for example, by using Dijkstra's algorithm [Dijkstra, 1959]. (This follows from the fact that, in the worst case, one single operator can introduce $\mathcal{D}_v^+$ arcs into G_v.) Naturally, we can also decide if there exists a path between two values in a domain-transition graph in $O(|\mathcal{D}_v^+||\mathcal{O}|)$ time.

We begin by determining the time complexity of *Extend* and *Instantiate* in Lemmata 5.4.4 and 5.4.5, continue with the irreflexivity test in Lemma 5.4.7 and finally show the worst-case time complexity of *Plan* in Theorem 5.4.8. Let $\Delta = \langle \mathcal{A}, \prec \rangle$ be the result of applying the *Plan* algorithm on some arbitrary SAS$^+$-IA instance $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$.

Lemma 5.4.1 The *Shortest-Path* algorithm runs in $O(|\mathcal{O}||\mathcal{D}_v^+|^4)$ time.

Proof: Line 2 takes $O(|\mathcal{O}||\mathcal{D}_v^+|^2)$ time since $|X| \leq |\mathcal{D}_v^+|$. By the same reason, line 5 runs in $O(|\mathcal{O}||\mathcal{D}_v^+|^3)$. The loop in line 4 can iterate $|\mathcal{D}_v^+|$ times at most so line 2-9 runs in $O(|\mathcal{O}||\mathcal{D}_v^+|^2 + |\mathcal{O}||\mathcal{D}_v^+|^4) = O(|\mathcal{O}||\mathcal{D}_v^+|^4)$. Finally, line 11-12 runs in $O(|\mathcal{O}||\mathcal{D}_v^+|^2)$ time so the complete algorithm takes $O(|\mathcal{O}||\mathcal{D}_v^+|^4)$ time. □

Lemma 5.4.2 For arbitrary $v \in \mathcal{V}$, $x, y \in \mathcal{D}_v^+$ and $X \subseteq \mathcal{R}_v^{\mathcal{O}}$, if α is a shortest path from x to y via X, then α contains at most two arcs labelled with the same operator.

Proof: Let $\prec$ denote the ordering of the operators in α. Suppose $a, b, c \in \alpha$ and $type(a) = type(b) = type(c) = t$. Further, suppose without loss of generality that $a \prec^+ b \prec^+ c$. Now, suppose there does not exist any action $d \in \alpha$ such that $a \prec^+ d \prec^+ c$ and $\mathsf{post}(d)[v] \in X$. Then we could replace the part of the path in G_v starting with a and ending with c by a single action of type t and still have a path in G_v from x to y via X. This contradicts the minimality of α, so there must exist an action $d \in \alpha$ such that $a \prec^+ d \prec^+ c$ and $\mathsf{post}(d)[v] \in X$. Suppose there exists an action

$e \in \alpha$ such that $a \prec^+ e \prec^+ c$ and $\mathsf{post}(e)[v] \in X$ but $\mathsf{post}(e)[v] \neq \mathsf{post}(d)[v]$. Then, there exists a path in G_v from $\mathsf{post}(e)[v]$ to $\mathsf{post}(d)[v]$ because $e \prec^+ c$ and $a \prec^+ d$ and $type(c) = type(a)$. This contradicts that $G_v^{\mathcal{R}_v^{\mathcal{O}}}$ is acyclic and, hence, $\mathsf{post}(d)[v]$ is the only value in X that is achieved by any action between a and c. Now, suppose $a \prec^+ d \prec^+ b \prec^+ c$. Then we could replace the path between b and c by a single action of type t and still have a path in G_v from x to y via X. However, this contradicts that α is a shortest path. We get the same contradiction if we assume $a \prec^+ b \prec^+ d \prec^+ c$. Hence, our initial assumption that there can exist three actions of the same type in α is erroneous and the lemma follows. □

Corollary 5.4.3 Δ contains at most two actions of each type.

Lemma 5.4.1 and Corollary 5.4.3 are combined in the following lemma to show the complexity of *Extend*.

Lemma 5.4.4 Extend has a worst-case time complexity of $O(|\mathcal{O}||\mathcal{V}|^2 + |\mathcal{O}||\mathcal{V}|(\max_{v\in\mathcal{V}} |\mathcal{D}_v^+|)^4)$.

Proof: By Lemma 5.4.1, line 3-5 runs in

$$O(\sum_{v\in\mathcal{V}}(|\mathcal{O}||\mathcal{D}_v^+|^4)) = O(|\mathcal{O}|\sum_{v\in\mathcal{V}}(|\mathcal{D}_v^+|^4)) = O(|\mathcal{O}||\mathcal{V}|(\max_{v\in\mathcal{V}}|\mathcal{D}_v^+|)^4$$

By Corollary 5.4.3, both line 6-8 and line 9-11 run in $O(|\mathcal{V}^2||\mathcal{O}|)$ time under the assumption that lines 8 and 11 takes $O(1)$ time. Consequently, *Extend* runs in $O(|\mathcal{V}|^2|\mathcal{O}| + |\mathcal{O}||\mathcal{V}|(\max_{v\in\mathcal{V}}(|\mathcal{D}_v^+|))^4)$ time. □

The importance of Corollary 5.4.3 is once again stressed as it gives us a neat way of showing the complexity of *Instantiate:*

Lemma 5.4.5 *Instantiate* has a worst-case time complexity of $O(|\mathcal{V}|^2|\mathcal{O}|^2)$.

Proof: The outer **for** loop makes $|\mathcal{V}|$ turns and the inner **for** loop makes at most $2|\mathcal{O}|$ turns by Corollary 5.4.3. The test in line 5 can be performed in $O(|\mathcal{V}||\mathcal{O}|)$ time by Corollary 5.4.3 so consequently, *Instantiate* runs in $O(|\mathcal{V}|^2|\mathcal{O}|^2)$ time. □

Testing irreflexivity can be done with topological sorting.

Lemma 5.4.6 Testing if a directed graph $G = \langle V, E\rangle$ is irreflexive can be done in $O(|V| + |E|)$ time.

Proof: See Mehlhorn [1984, pages 4–7]. $\square$

By using the trivial upper bound on the size of $\prec$, we can use Lemma 5.4.6 to decide the time complexity of the irreflexivity test:

Lemma 5.4.7 The irreflexivity test in line 18 has a worst-case time complexity of $O(|\mathcal{O}|^2)$.

Proof: $\prec\subseteq \mathcal{A}\times\mathcal{A}$ which implies that $|\prec| \leq (2|\mathcal{O}|)^2 = 4|\mathcal{O}|^2$ by Corollary 5.4.3. By Lemma 5.4.6, the irreflexitivity test runs in $O(|\mathcal{A}| + |\prec|) \subseteq O(|\mathcal{O}| + |\mathcal{O}|^2) = O(|\mathcal{O}|^2)$ time. $\square$

By now, we have all prerequisites for proving the complexity of *Plan*:

Theorem 5.4.8 The *Plan* algorithm has a worst-case time complexity of $O(|\mathcal{V}|^3|\mathcal{O}|^2 \max_{v\in\mathcal{V}}(|\mathcal{D}_v^+|)^5)$.

Proof: The **repeat** loop in line 3-5 can call *Extend* maximally $\sum_{v\in\mathcal{V}}|\mathcal{D}_v^+|$ times so, by Lemma 5.4.4, line 3-5 runs in

$$O((\sum_{v\in\mathcal{V}}|\mathcal{D}_v^+|)(|\mathcal{O}||\mathcal{V}|^2 + |\mathcal{O}||\mathcal{V}|\max_{v\in\mathcal{V}}(|\mathcal{D}_v^+|)^4)) =$$

$$= O(|\mathcal{V}|\max_{v\in\mathcal{V}}(|\mathcal{D}_v^+|)|\mathcal{O}||\mathcal{V}|^2 + |\mathcal{V}|\max_{v\in\mathcal{V}}(|\mathcal{D}_v^+|)(|\mathcal{O}||\mathcal{V}|\max_{v\in\mathcal{V}}(|\mathcal{D}_v^+|)^4))$$

time which equals $O(|\mathcal{O}||\mathcal{V}|^3\max_{v\in\mathcal{V}}(|\mathcal{D}_v^+|) + |\mathcal{O}||\mathcal{V}|^2\max_{v\in\mathcal{V}}(|\mathcal{D}_v^+|)^5)$ time. Clearly, $O(|\mathcal{V}|^3|\mathcal{O}|^2\max_{v\in\mathcal{V}}(|\mathcal{D}_v^+|)^5)$ dominates the **repeat** loop. Furthermore, it dominates lines 9-16 running in $O(|\mathcal{O}||\mathcal{V}|)$ time, it dominates the irreflexivity test which runs in $O(|\mathcal{O}|^2)$ time by Lemma 5.4.7 and it dominates the *Instantiate* procedure which runs in $O(|\mathcal{V}|^2|\mathcal{O}|^2)$ time by Lemma 5.4.5. Consequently, the algorithm runs in $O(|\mathcal{V}|^3|\mathcal{O}|^2\max_{v\in\mathcal{V}}(|\mathcal{D}_v^+|)^5)$ time. $\square$

The algorithm does not compute the transitive closure of the order since this is a relatively costly operation and most applications are not likely to require transitivity. The transitive closure can be computed in $O(|\mathcal{A}|^3) \subseteq O(|\mathcal{O}|^3)$ time. (Mehlhorn [1984, pages 7–9].)

Chapter 6

Complexity Results

This chapter presents an exhaustive complexity map for planning under all combinations of our previously studied syntactical and structural restrictions. We will only discuss the plan generation problem (finding a solution) and not consider the plan existence problem (deciding whether a solution exists), since we are ultimately interested in actually generating a solution.

6.1 Preliminaries

In this chapter, we will consider the SAS* formalism instead of the SAS$^+$ formalism. The reason is that all previously described polynomial-time SAS$^+$ planners require that the problem instance satisfy both restriction I and A. This mixes badly with operators having u as precondition. For example, restriction I prevents the existence of a path in G_v from u to any state that is the precondition of a non-unary action and restriction A prevents the existence of two requestable values which are both reachable from u. Another reason is that, as we will see in Example 6.2.1, SAS*-P instances are always SAS*-O instances whereas SAS$^+$-P instances are not necessarily SAS$^+$-O instances. If we had studied the SAS$^+$ case instead of SAS*, the presentation would had been even more complex and cluttered with obscure special cases. Hence, we believe that it is sufficient to study the SAS* formalism.

We begin by formally defining the problems we will consider. We define the plan existence problems as well as the plan generation problems since some of the hardness results are stated in terms of the corresponding plan existence problems.

Definition 6.1.1 Given a SAS* instance Π, we have the following problems:

The **plan existence problem (PE)** decides whether a solution for Π exists or not. The **bounded plan existence problem (BPE)** takes an integer $k \geq 0$ as an additional parameter and decides whether a solution for Π of length k or shorter exists or not. The **plan generation problem (PG)** finds a solution for Π or answers that no solution exists. The **bounded plan generation problem (BPG)** takes an integer $k \geq 0$ as an additional parameter and finds a solution for Π of length k or shorter or answers that no such solution exists.

The complexity results will be presented in lattices of the type shown in Figure 6.1 which can be viewed as a three-dimensional cube. The figure is to be interpreted in the following way: The top-element of each diamond-shaped sublattice corresponds to a combination of restrictions on the SAS* problem defined by selecting at most one restriction from each of the sets $\{\mathrm{A}^+/\mathrm{A}, \mathrm{A}^-\}$, $\{\mathrm{P}, \mathrm{O}\}$ and $\{\mathrm{U}, \mathrm{I}\}$. As will be shown later on, this makes sense because every unary instance is interference-safe, every post-unique instance is prevail-order-preserving and so on. Furthermore, there is no need to distinguish between the A^+ and A restrictions since every result holding for SAS*-A^+ hold also for SAS*-A instances and vice versa. These restrictions are marked along the three axes in the figure, where "-" denotes that neither of the two restrictions on an axis applies. The other three points in each sublattice further specialize the top element by adding one or both of the restrictions B and S, as shown in the enlarged sublattice. As an example of how to interpret the lattice, the SAS*-SA$^-$O problem is indicated explicitly. These lattices are perhaps quite detailed and awkward to interpret, so a more easily intelligible summary will be given later on.

6.2 Restriction Relationships

Before stating the complexity results, we have to investigate the relationships between different problem restrictions. We begin by studying the domain-transition graphs of post-unique SAS* instances.

Definition 6.2.1 A **simple cycle** is a one-component graph such that every vertex have incoming and outgoing degree one.

Definition 6.2.2 A directed graph $G = \langle V, E \rangle$ is a **P-graph** iff for every component $C = \langle W, F \rangle$ in G,

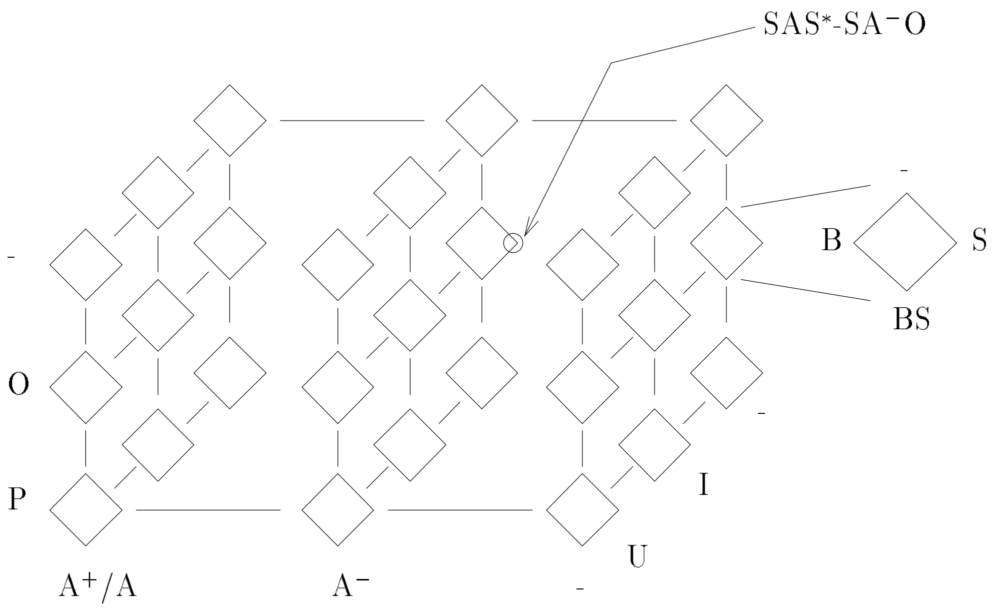


Figure 6.1: The lattice of restricted SAS problems.*

1. there is some subset $W' \subseteq W$ (the **center** of C) s.t. W' is either a singleton or C restricted to W' is a simple cycle, and

2. each $x \in W'$ is the root of a (possible empty) directed tree in $C' = \langle W, F - \{\langle x, y\rangle \mid x, y \in W'\}\rangle$,

Theorem 6.2.3 The set of operators affecting a SAS* variable v is post-unique (wrt. to v) iff G_v is a P-graph.

Proof: *if:* Obvious by noticing that post-uniqueness holds iff every vertex of G_v has incoming degree 0 or 1.

only-if: We concentrate on an arbitrary component $C = \langle W, F\rangle$ of G_v and study two cases.

1. C is acyclic. Then, there must exist some $w \in W$ with incoming degree zero. Suppose there are two (or more) vertices $v, w \in W$ with incoming

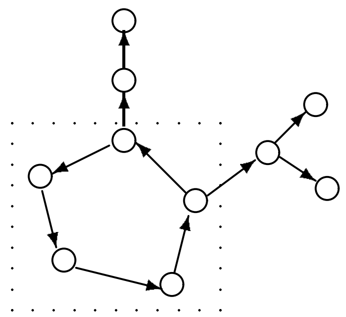


Figure 6.2: A sample one-component P-graph. The center is within the dotted box.

degree zero. C is connected so there has to be a vertex $z \in W$ such that there exists disjoint paths from v to z and from w to z. Hence, $deg^{in}(z) \geq 2$, contradicting post-uniqueness. Consequently, there is exactly one vertex $v \in W$ with incoming degree zero. Suppose C is not a tree. This can only be the case if some vertex has incoming degree greater than one which contradicts post-uniqueness. Hence, C is a tree rooted in v, and C is a P-graph by definition.

2. C is not acyclic. It is obvious that if C contains a cycle, then this cycle is simple because otherwise it would have some vertex with incoming degree greater than one, thus contradicting post-uniqueness. Suppose there exist two simple cycles A and B in C with at least one vertex not in common. C is connected, so there has to be a path either from some vertex in A to some vertex in B or vice versa. In both cases, there must exist a vertex with incoming degree greater than one. Hence, there can only be one cycle in C. Denote this cycle with A. Arbitrarily choose one vertex $x \in A$ and let X be the largest subgraph of C such that the vertices in X are reachable from x but are not on the cycle A. Suppose X is not a directed tree rooted in x. By the construction of X, this can be the case only if some vertex in X has incoming degree greater than one, thus contradicting post-uniqueness. Since x is an arbitrary vertex in A, C is a P-graph.

Since C was arbitrarily selected, the theorem follows. □

Lemma 6.2.4 Let $\langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ be a SAS*-P instance. An operator $o \in \mathcal{O}$ is irreplaceable wrt. $v \in \mathcal{V}$ iff it is not part of a cycle in G_v.

Proof: Obvious from the fact that G_v is a P-graph. $\square$

Before proving the main theorem of this subsection, we need the following definition.

Definition 6.2.5 A path $\omega = \langle o_1, \ldots, o_k \rangle$ in a domain-transition graph G_v is **loop-free** iff $\mathsf{post}(o_k)[v] \neq \mathsf{post}(o_l)[v]$ for $k \neq l$.

We can now state the relations that hold between different sets of restrictions. Statements of the type SAS*-X⊆SAS*-Y in the following theorem should be read as every SAS* instance satisfying restriction X also satisfies restriction Y. In the following, we will use this theorem several times without explicit reference.

Theorem 6.2.6 The following subproblem relations hold:

1. SAS*-A$^+$ ⊆ SAS*-A	2. SAS*-A ⊆ SAS*-A$^-$
3. SAS*-U ⊆ SAS*-I	4. SAS*-US ⊆ SAS*-A
5. SAS*-S ⊆ SAS*-A$^-$	6. SAS*-PA ⊆ SAS*-I
7. SAS*-P ⊆ SAS*-O	

Proof: 1,2,3: Trivial
4: Follows immediately from the fact that $|\mathcal{R}_v^{\mathcal{O}}| \leq 1$ for all $v \in \mathcal{V}$.
5: If Π is single-valued, then $|\mathcal{P}_v^{\mathcal{O}}| \leq 1$ for all $v \in \mathcal{V}$.
6: For every variable $v \in \mathcal{V}$, restriction A prevents that a non-unary operator is on a cycle of G_v so the inclusion immediately follows from Lemma 6.2.4.
7: Arbitrarily choose $x \in \mathcal{D}_v$, $y \in \mathcal{D}_v^+$ and $X \subseteq \mathcal{D}_v$. Let ω be a shortest path from x to y via X and let ψ be an arbitrary path from x to y via X. Let $C = \langle W, F \rangle$ be the component of G_v where x is a member and let W' be the center of C. We have two cases:

1. $y \neq \mathsf{u}$. If $x \notin W'$, then there exists a unique path from x to y by Theorem 6.2.3 and, obviously, $\omega \trianglelefteq \psi$. Suppose $x \in W'$. We have four cases to investigate.

    - ω is loop-free and ψ is loop-free. As a consequence of Theorem 6.2.3, $\omega = \psi$ and $\omega \trianglelefteq \psi$ follows.
    - ω is loop-free but ψ is not. The only cycle that ψ can loop in is the center of C. Hence, by Theorem 6.2.3, ω is a subpath of ψ and $\omega \trianglelefteq \psi$.

- ω is not loop-free and ψ is not loop-free. As in the previous case, the only cycle that ω and ψ can loop in is the center of C so $\omega \trianglelefteq \psi$ follows immediately.
- ω is not loop-free but ψ is. This case cannot occur since ω is a shortest path from x to y via X.

2. $y = \mathsf{u}$. Let o be the last operator in ω, $z = \mathsf{post}(o)[v]$ and observe that $z \in X$ since ω is a minimal path from x to u via X. If $x \notin W'$, then there exists a unique path from x to z by Theorem 6.2.3 so ψ must contain ω as a subpath. Clearly, $\omega \trianglelefteq \psi$. Suppose $x \in W'$. We have four cases to investigate.

   - ω is loop-free and ψ is loop-free. Since $z \in Y$, ω must be a subpath of ψ as a consequence of Theorem 6.2.3. Hence, $\omega \trianglelefteq \psi$ follows.
   - ω is loop-free but ψ is not. The only cycle that ψ can loop in is the center of C. Hence, by Theorem 6.2.3 and the fact that $z \in X$, ω is a subpath of ψ and $\omega \trianglelefteq \psi$.
   - ω is not loop-free and ψ is not loop-free. As in the previous case, the only cycle that ω and ψ can loop in is the center of C so $\omega \trianglelefteq \psi$ follows immediately.
   - ω is not loop-free but ψ is. This case cannot occur since ω is a shortest path from x to u via X.

□

A picture summarizing the results in Theorem 6.2.6 can be found in Figure 6.3. By the previous theorem, SAS*-P instances are always SAS*-O instances. Unfortunately, it is not the case that SAS$^+$-P instances are necessarily SAS$^+$-O instances.

Example 6.2.1 Suppose that we have a variable in a SAS$^+$ instance with three values, 1, 2 and 3, and three operators $o_1, \ldots, o_3$ affecting v s.t. $\mathsf{pre}(o_i)[v] = \mathsf{u}$ and $\mathsf{post}(o_i)[v] = i$, $1 \leq i \leq 3$. Clearly, the operators affecting v are post-unique. Furthermore, suppose that these operators have incompatible prevail-conditions. We then have two shortest paths from u to 3 via $\{1, 2\}$, namely $\omega_1 = \langle o_1, o_2, o_3 \rangle$ and $\omega_2 = \langle o_2, o_1, o_3 \rangle$. By our assumption, $\omega_1 \not\trianglelefteq \omega_2$ so we do not have prevail-order-preservation although we have post-uniqueness.

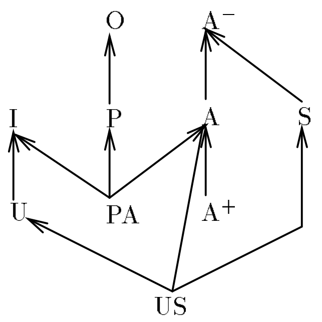


Figure 6.3: Summary of Theorem 6.2.6.

6.3 Tractability Results

The first tractability result to be presented already appears in a previous publication, as indicated below.

Theorem 6.3.1 [Bäckström and Nebel, 1993] **PG** for SAS$^+$-US is polynomial.

Our second tractability result concerns the SAS$^+$-IA$^-$O class. We show that it is tractable by making a Turing reduction using the SAS$^+$-IAO algorithm. In other words, we will devise a method that transforms an arbitrary SAS$^+$-IA$^-$O instance into an equivalent SAS$^+$-IAO instance, solve it with the SAS$^+$-IAO algorithm and convert the solution back to a solution of the original problem. We begin by presenting the transform.

Definition 6.3.2 Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ be a SAS$^+$-I instance and let $o \in \mathcal{O}$ be a non-unary operator. Then, $New_o = \{o'\} \cup \{o_v^{\mathsf{pre}}, o_v^{\mathsf{post}} \mid \mathsf{post}(o)[v] \neq \mathsf{u}\}$ where

$$\begin{aligned}
\mathsf{pre}(o')[v] &= \begin{cases} s' & \text{if } \mathsf{pre}(o)[v] = s \neq \mathsf{u} \\ \mathsf{u} & \text{otherwise} \end{cases} \\
\mathsf{post}(o')[v] &= \begin{cases} t' & \text{if } \mathsf{post}(o)[v] = t \neq \mathsf{u} \\ \mathsf{u} & \text{otherwise} \end{cases} \\
\mathsf{prv}(o')[v] &= \mathsf{prv}(o)[v]
\end{aligned}$$

$$\mathsf{pre}(o_v^{\mathsf{pre}})[w] = \begin{cases} \mathsf{pre}(o)[w] & \text{if } v = w \\ \mathsf{u} & \text{otherwise} \end{cases}$$

$$\mathsf{post}(o_v^{\mathsf{pre}})[w] = \begin{cases} s' & \text{if } v = w \text{ and } \mathsf{pre}(o)[w] = s \neq \mathsf{u} \\ \mathsf{u} & \text{otherwise} \end{cases}$$

$$\mathsf{prv}(o_v^{\mathsf{pre}})[w] = \mathsf{u}$$

$$\mathsf{pre}(o_v^{\mathsf{post}})[w] = \begin{cases} t' & \text{if } v = w \text{ and } \mathsf{post}(o)[w] = t \neq \mathsf{u} \\ \mathsf{u} & \text{otherwise} \end{cases}$$

$$\mathsf{post}(o_v^{\mathsf{post}})[w] = \begin{cases} \mathsf{post}(o)[w] & \text{if } v = w \\ \mathsf{u} & \text{otherwise} \end{cases}$$

$$\mathsf{prv}(o_v^{\mathsf{post}})[w] = \mathsf{u}$$

The **A-transform** of Π, $Atr(\Pi)$, is defined as $Atr(\Pi) = \langle \mathcal{V}, (\mathcal{O} - \{o \in \mathcal{O} \mid o \text{ is non-unary}\}) \cup \bigcup\{New_o \mid o \in \mathcal{O} \text{ and } o \text{ is non-unary}\}, s_0, s_*\rangle$ with the associated variable domains $\mathcal{D}_v$, for $v \in \mathcal{V}$, changed to range over the new operators.

By Lemma 5.1.2, the previous definition is sound. Note that the A-transform can easily be carried out in polynomial time. The effect of the A-transform is shown in Figure 6.4. The upper part of Figure 6.4 depicts the domain-transition graphs of two variables v_1 and v_2. The operator o is non-unary and affects both v_1 and v_2. Suppose the marked value in the domain-transition graph of v_1 is a requestable value. Then s_1, which is a requestable value, forms a cycle with the marked value. Hence, we do not have acyclicity wrt. requestable values in this example. The lower part of Figure 6.4 shows the domain-transition graphs for v_1 and v_2 after the A-transform. We can see that the incorporation of the o^{pre} and o^{post} operators (henceforth referred to as *artificial* operators) has made the variables acyclic wrt. requestable states. This leads us to the following lemma.

Lemma 6.3.3 Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_*\rangle$ be a SAS$^+$ instance. Then, Π is a SAS$^+$-IA$^-$O instance iff $Atr(\Pi) = \langle \mathcal{V}, \mathcal{O}', s_0, s_*\rangle$ is a SAS$^+$-IAO instance.

Proof: In the proof, let $\mathcal{D}_v$ denote the domain of the variable v in Π and let $\mathcal{D}'_v$ denote the domain of v in $Atr(\Pi)$. Similarily, let G_v denote the domain-transition graph of variable v in Π and let G'_v denote the domain-transition graph of v in $Atr(\Pi)$.

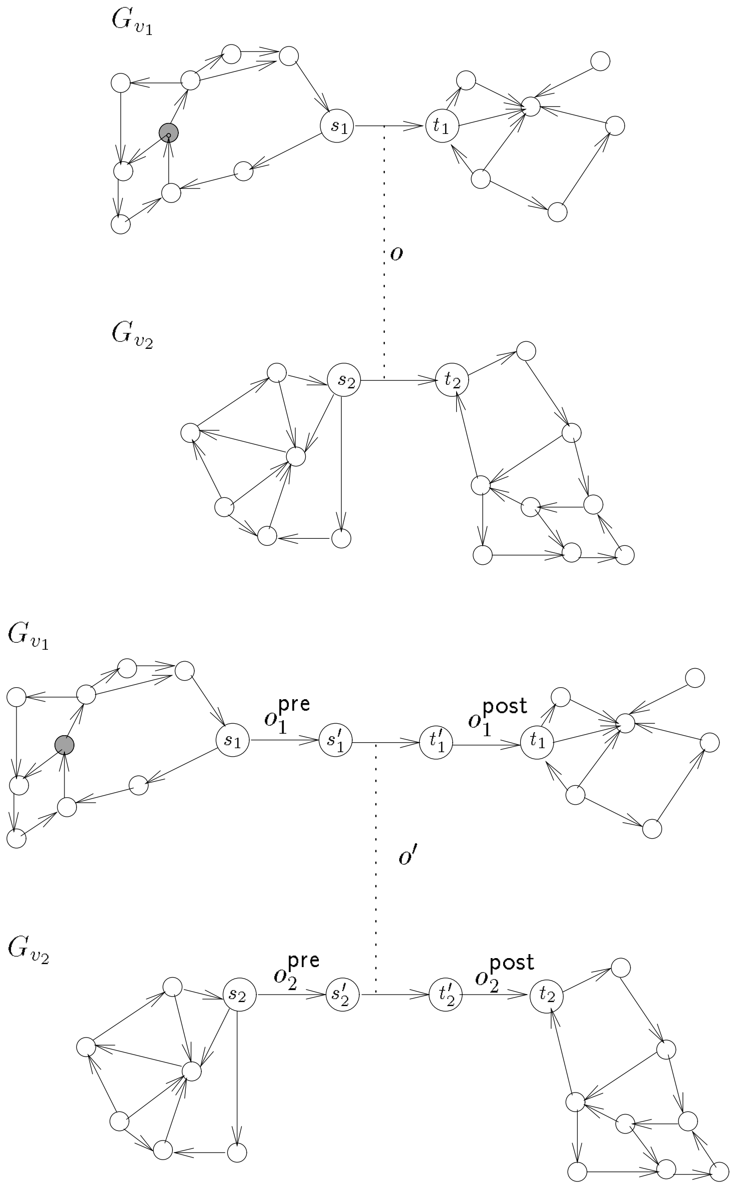


Figure 6.4: Domain-transition graphs for v_1 and v_2 before and after the A-transform

only-if: Obviously, $Atr(\Pi)$ is interference-safe by its construction since Π is interference-safe. Furthermore, it is easy to see that the pre- and post-conditions of any non-unary operator in $Atr(\Pi)$ cannot be on any cycle by interference-safeness. Since we know that Π is acyclic wrt. prevail-requestable values, no values in $\mathcal{P}_v^{\mathcal{O}'}$, $v \in \mathcal{V}$ are members of some cycle in G_v. Hence, G'_v is acyclic wrt. requestable values for all $v \in \mathcal{V}$. Before showing that $Atr(\Pi)$ is prevail-order-preserving, note that it is sufficient, by Theorem 3.3.2, to show that $Atr(\Pi)$ is weakly prevail-order-preserving since $Atr(\Pi)$ satisfies restriction A.

Arbitrarily choose a variable $v \in \mathcal{V}$. We show that there does not exist any $x \in \mathcal{D}'_v$, $y \in \mathcal{D}'_v$, a shortest path ω from x to y and some other path ψ from x to y such that $\omega \ntrianglelefteq \psi$. The proof is by induction over n, the length of the shortest path from x to y.

Basis step: Arbitrarily choose $x, y \in \mathcal{D}'_v$ such that $n = 0$. Then $\omega = \langle\rangle$ and consequently $\omega \trianglelefteq \psi$.

Induction hypothesis: Arbitrarily choose $x, y \in \mathcal{D}'_v$ such that $n = k$, $k \geq 0$. Assume that for all shortest paths ω from x to y and for all paths ψ from x to y, $\omega \trianglelefteq \psi$.

Induction step: Arbitrarily choose $x, y \in \mathcal{D}'_v$ such that $n = k + 1$, $k \geq 0$. Let ω be an arbitrary shortest path from x to y in G'_v and let ψ be an arbitrary path from x to y in G'_v. If ψ contains no non-unary or artificial operators, then ω does not contain any either. This means that both ω and ψ goes through regions of the domain-transition graph that are unaffected by the A-transform. Hence, $\omega \trianglelefteq \psi$, since we know that Π is prevail-order-preserving, Now, assume that ψ is of the form

$$(\psi_1; o^{\mathsf{pre}}; o'; o^{\mathsf{post}}; \psi_2)$$

where o^{pre} and o^{post} are artificial operators introduced by the A-transform and o' is a non-unary operator modified by the transform. Remember that ω is a shortest path from x to y in G'_v where $y \neq \mathsf{u}$. By interference-safeness, ω must be of the form

$$(\omega_1; o^{\mathsf{pre}}; o'; o^{\mathsf{post}}; \omega_2)$$

where ω_1 is a shortest path from x to $\mathsf{pre}(o^{\mathsf{pre}})[v]$ and ω_2 is a shortest path from $\mathsf{post}(o^{\mathsf{post}})$ to y. By the induction hypothesis, $\omega_1 \trianglelefteq \psi_1$ and $\omega_2 \trianglelefteq \psi_2$ and, consequently, $\omega \trianglelefteq \psi$. Hence, $Atr(\Pi)$ is prevail-order-preserving. Cases where

$\psi = (\psi_1; o^{\mathsf{pre}})$, $\psi = (\psi_1; o^{\mathsf{pre}}; o')$ or $\psi = (\psi_1; o^{\mathsf{pre}}; o'; o^{\mathsf{post}})$ are analogous. Hence, if Π is a SAS$^+$-IA$^-$O instance, $Atr(\Pi)$ is a SAS$^+$-IAO instance.

if: Arbitrarily choose $v \in \mathcal{V}$ and $x, y \in \mathcal{D}_v$. Let $X \subseteq \mathcal{D}_v$. Let ω be a shortest path from x to y via X in G_v and let ψ be an arbitrary path from x to y via X in G_v. Let ω' be like ω but extended with artificial operators and modified non-unary operators as in the A-transform. Obviously, ω' is a shortest-path from x to y via X in G'_v. Similarily, let ψ' be like ψ but with modified non-unary operators and extended with artificial operators. Naturally, ψ' is a path from x to y via X in G'_v. By assumption, $\omega' \trianglelefteq \psi'$. By irreplaceability, ψ contains (at least) the same non-unary operators as ω and they appear in the same order in ψ as in ω. Suppose we have k non-unary operators $o_1, \ldots, o_k$ in ω such that

$$\omega = (\omega_1; o_1; \omega_2; \ldots; \omega_k; o_k; \omega_{k+1}).$$

Then, we can divide ψ such that

$$\psi = (\psi_1; o_1; \psi_2; \ldots; \psi_k; o_k; \psi_{k+1})$$

where $\psi_1, \ldots, \psi_k$ does not contain any non-unary operators. Clearly,

$$\omega' = (\omega_1; o_1^{\mathsf{pre}}; o_1; o_1^{\mathsf{post}}; \omega_2; \ldots; \omega_k; o_k^{\mathsf{pre}}; o_k; o_1^{\mathsf{post}}; \omega_{k+1})$$

and

$$\psi' = (\psi_1; p_1^{\mathsf{pre}}; p_1; p_1^{\mathsf{post}}; \psi_2; \ldots; \psi_k; p_k^{\mathsf{pre}}; p_k; p_1^{\mathsf{post}}; \psi_{k+1})$$

where $o_i^{\mathsf{pre}}, o_i^{\mathsf{post}}, p_i^{\mathsf{pre}}, p_i^{\mathsf{post}}$, $1 \leq i \leq k$ are artificial operators. We know that $\omega' \trianglelefteq \psi'$, but we also know that for $1 \leq i \leq k$, $o_i^{\mathsf{pre}} = p_i^{\mathsf{pre}}$, $o' = p'$ and $o_i^{\mathsf{post}} = p_i^{\mathsf{post}}$. Consequently, $\omega_i \trianglelefteq \psi_i$ for $1 \leq i \leq k+1$ and

$$\omega = (\omega_1; o_1; \omega_2; \ldots; \omega_k; o_k; \omega_{k+1}) \trianglelefteq (\psi_1; p_1; \psi_2; \ldots; \psi_k; p_k; \psi_{k+1}) = \psi.$$

□

Lemma 6.3.4 Let Π be a SAS$^+$-IA$^-$O instance. Then, Π has a solution iff $Atr(\Pi)$ has a solution.

Proof: Trivial from the construction of $Atr(\Pi)$. □

Theorem 6.3.5 Bounded plan generation for SAS$^+$-IA$^-$O is polynomial.

Proof: Let Π be an arbitrary SAS$^+$-IA$^-$O instance. By Lemma 6.3.3, $\Pi' = Atr(\Pi)$ is a SAS$^+$-IAO instance. By our earlier results on SAS$^+$-IAO planning, we can generate a plan for Π' or report that no such plan exists in polynomial time. By Lemma 6.3.4, we know that Π has a solution iff Π' has one. Furthermore, we can take a minimal plan ω' for Π', remove the artificial operators and substitiute the modified non-unary operators with the original ones. By the construction of Atr, this is a valid and minimal plan for Π. Both the A-transform and the SAS$^+$-IAO planning can be carried out in polynomial time so the theorem follows. $\square$

In order to make the SAS$^+$-IA$^-$O algorithm practically applicable, we need a polynomial-time algorithm that can decide if a SAS$^+$-IA$^-$ instance is prevail-order-preserving or not.

Theorem 6.3.6 If a SAS$^+$-IA$^-$ instance is prevail-order-preserving or not can be decided in polynomial time.

Proof: Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ be a SAS$^+$-IA$^-$ instance. Let $\Pi' = Atr(\Pi)$, let G'_v denote the domain-transition graph of v in $Atr(\Pi)$ and let $\mathcal{D}'_v$ denote the domain of the variable v in Π'. We know that $Atr(\Pi)$ is a SAS$^+$-IA instance. Hence, we can use the polynomial-time algorithm *Check-O* from chapter 3 to decide whether $Atr(\Pi)$ is prevail-order-preserving or not. By Lemma 6.3.3, $Atr(\Pi)$ is a SAS$^+$-IAO instance iff Π is a SAS$^+$-IA$^-$O instance and the theorem follows. $\square$

A summary of the tractability results can be found in Figure 6.5. The results are derived from the tractability of SAS*-IA$^-$O and SAS*-US by using the relations in Theorem 6.2.6.

6.4 Intractability Results

For the intractability results we have to distinguish between those problems that are inherently intractable, *ie.* can be proven to take exponential time, and those which are NP-equivalent (*ie.* intractable unless P=NP)[1]. See

[1] Since we consider the search problem (generating a solution) and not the decision problem (whether a solution exists) we cannot use the term NP-complete, which is used only for decision problems. A search problem is NP-easy if it can be Turing reduced to some NP-complete problem, NP-hard if some NP-complete problem can be reduced to it and NP-equivalent if it is both NP-easy and NP-hard. Loosely speaking, NP-equivalence is to search problems what NP-completeness is to decision problems.

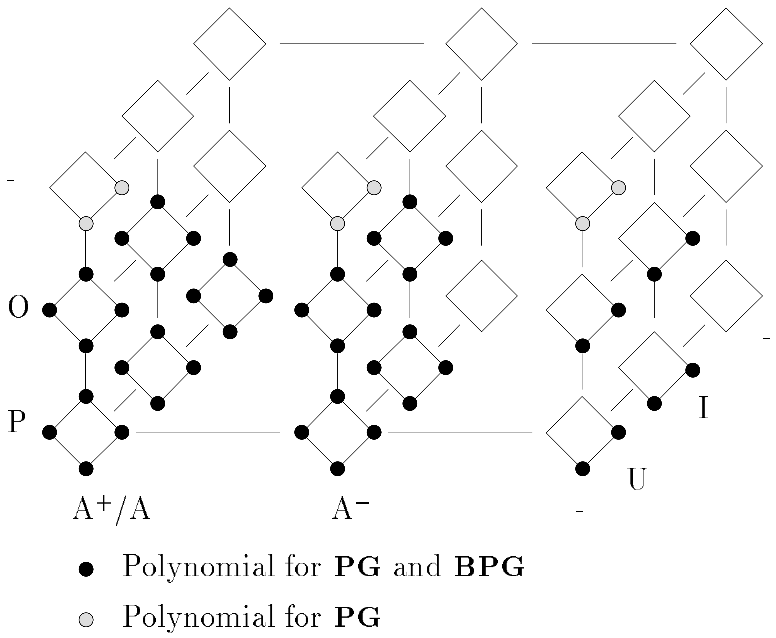


Figure 6.5: Tractability results.

Johnson [1990] or Garey and Johnson [1979] for formal details. We begin by showing some results, concering NP-easiness.

Lemma 6.4.1 The length of minimal solutions are bounded by $|\mathcal{V}| \cdot (1 + \max_{v \in \mathcal{V}} \mathcal{D}_v)^2$ for SAS*-A instances.

Proof: Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ be an arbitrary SAS*-A instance and let ω be a minimal solution to Π. Let $\mathcal{B}$ be the set of operators in ω. By Lemma 5.2.4, ω is $\mathcal{R}$-unique. Let $\omega_i = \mathit{Paths}(v_i, \omega)$ for $1 \leq i \leq m$. We show that $|\omega_i| \leq (1 + |\mathcal{D}_v|)^2$ for each $1 \leq i \leq m$ and, hence, $\omega \leq |\mathcal{V}| \cdot (1 + \max_{v \in \mathcal{V}} |\mathcal{D}_v|)^2$ Arbitrarily choose $1 \leq i \leq m$. We have two cases to consider according to the two cases in the definition of $\mathcal{R}$-uniqueness. First assume that $s_0[v] \notin \mathcal{R}^{\mathcal{B}}_v$. Hence, for every s in $\mathcal{R}^{\mathcal{B}}_{v_i}$, there exists exactly one operator o in ω such that $\mathsf{post}(o)[v] = s$. Note that $|\mathcal{R}^{\mathcal{B}}_{v_i}| \leq |\mathcal{D}_{v_i}|$ and assume that the values in $\mathcal{R}^{\mathcal{B}}_{v_i}$ are achieved in the order $r_1, \ldots, r_k$, $1 \leq k \leq |\mathcal{D}_{v_i}|$ by ω_i. Suppose that there exists a p, $1 \leq p \leq k - 1$ such that the subsequence ψ of ω_i that begins with the value r_p and ends in the value r_{p+1} contains more than $|\mathcal{D}_{v_i}|$ operators. Obviously, ψ cannot be a loop-free path from r_p to r_{p+1} in G_{v_i}. Let ψ' be a loop in ψ, that is a subsequence of ψ such that $\mathsf{post}(p')[v_i] = \mathsf{post}(p'')[v_i]$ where p' and p'' are the first and last operator of ψ' respectively. Since Π satisfies restriction A, there can only be unary operators in ψ'. Since we know that no requestable values are achieved by ψ', we can remove it from ω and still have a valid plan which contradicts the minimality of ω. Consequently, $|\psi| \leq |\mathcal{D}_v|$. Similarily, we can show that the paths from $s_0[v]$ to r_1 and r_k to $s_*[v]$ are bounded by $|\mathcal{D}_v|$. Hence, $|\omega_i| \leq (1 + |\mathcal{D}_{v_i}|)^2$ for $1 \leq i \leq m$. The case when $s_0[v_i] \notin \mathcal{R}^{\mathcal{B}}_{v_i}$ is analogous. □

Lemma 6.4.2 Minimal solutions are always of polynomially bounded length for SAS*-UA⁻ instances.

Proof: If a SAS* instance Π satisfies restrictions U and A^-, then it satisfies A as well since it has no non-unary operators. The lemma follows immediately from Lemma 6.4.1. □

Lemma 6.4.3 Minimal solutions are always polynomially bounded for SAS*-IA⁻ instances.

Proof: Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ be an instance of the SAS*-IA⁻ plan generation problem and suppose $\omega = \langle o_1, \ldots, o_m \rangle$ is a minimal solution to Π. Since every non-unary operator is irreplaceable, there is at most one such operator of each type in ω. That is, there is at most $|\mathcal{O}|$ non-unary actions in ω. Hence, we can divide ω the following way:

$$\omega = \langle \omega_1; p_1; \omega_2; \ldots; p_k; \omega_{k+1} \rangle$$

where p_k is the k:th non-unary operator ($0 \leq k \leq |\mathcal{O}|$) and ω_l ($0 \leq l \leq k+1$) is a plan fragment only containing unary operators. Each ω_l is then a solution to some SAS*-UA$^-$ problem instance and, by Theorem 6.4.2, every ω_l is bounded by $|\mathcal{V}| \cdot (1 + \max_{v \in \mathcal{V}} \mathcal{D}_v)^2$. Hence, ω itself is bounded by $|\mathcal{O}||\mathcal{V}| \cdot (1 + \max_{v \in \mathcal{V}} \mathcal{D}_v)^2$ □

Corollary 6.4.4 **BPG** is NP-easy for SAS*-A, SAS*-UA$^-$ and SAS*-IA$^-$.

In the following hardness results, we use three problems, Minimum Cover, Exact Cover by 3-Sets and Satisfiability, all three known to be NP-complete [Garey and Johnson, 1979], as the basis for our reductions. They are defined as follows:

Definition 6.4.5 An instance of the Minimum Cover problem is given by a finite set $X = \{x_1, \ldots, x_m\}$, a set $C = \{C_1, \ldots, C_n\}$ of subsets of X and a positive integer $K \leq |C|$. The question is whether there exists a cover for X, *i.e.*, a subset $C' \subseteq C$ with $|C'| \leq K$ such that every element of X belongs to at least one member of C'.

Definition 6.4.6 An instance of the Exact Cover by 3-Sets (X3C) problem is given by a finite set $X = \{x_1, \ldots, x_{3m}\}$ and a set $C = \{C_1, \ldots, C_n\}$ of 3-element subsets of X. The question is whether there exists an exact cover for X, *i.e.*, a subset $C' \subseteq C$ such that every element of X belongs to exactly one member of C'.

Definition 6.4.7 An instance of the Satisfiability (SAT) problem is given by a set $U = u_1, \ldots, u_m$ of boolean variables and a set $C = \{c_1, \ldots, c_n\}$ of clauses over U. The question is whether there is a satisfying truth assignment for C.

Theorem 6.4.8 **BPE** is NP-hard for SAS*-UBSA$^+$.

Proof: Proof by reduction from Minimum Cover. Let $X = \{x_1, \ldots, x_m\}$ be a set, let $C = \{C_1, \ldots, C_n\}$ be a set of subsets of X and let K be an integer. Define a SAS*-UBSA$^+$ instance $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ such that

- $\mathcal{V} = \{x_k \mid 1 \leq k \leq m\} \cup \{c_k \mid 1 \leq k \leq n\}$;
- $\mathcal{D}_v = \{0, 1\}$ for all $v \in \mathcal{V}$;
- $\mathcal{O} = \{o_k^+ \mid 1 \leq k \leq n\} \cup \{o_{k,l} \mid 1 \leq k \leq n \text{ and } x_l \in C_k\}$, where for $1 \leq k \leq n$ and $v \in \mathcal{V}$,

$$\begin{aligned}
\mathsf{pre}(o_k^+)[v] &= \begin{cases} 0 & \text{if } v = c_k \\ \mathsf{u} & \text{otherwise} \end{cases} \\
\mathsf{post}(o_k^+)[v] &= \begin{cases} 1 & \text{if } v = c_k \\ \mathsf{u} & \text{otherwise} \end{cases} \\
\mathsf{prv}(o_k^+)[v] &= \mathsf{u}
\end{aligned}$$

  and for $1 \leq k \leq n$, $x_l \in C_k$ and $v \in \mathcal{V}$,

$$\begin{aligned}
\mathsf{pre}(o_{k,l})[v] &= \begin{cases} 0 & \text{if } v = x_l \\ \mathsf{u} & \text{otherwise} \end{cases} \\
\mathsf{post}(o_{k,l})[v] &= \begin{cases} 1 & \text{if } v = x_l \\ \mathsf{u} & \text{otherwise} \end{cases} \\
\mathsf{prv}(o_{k,l})[v] &= \begin{cases} 1 & \text{if } v = c_k \\ \mathsf{u} & \text{otherwise} \end{cases}
\end{aligned}$$

- $s_0[c_k] = 0$ for $1 \leq k \leq n$, $s_0[x_l] = 0$ for $1 \leq l \leq m$,
- $s_*[c_k] = \mathsf{u}$ for $1 \leq k \leq n$, $s_*[x_l] = 1$ for $1 \leq l \leq m$.

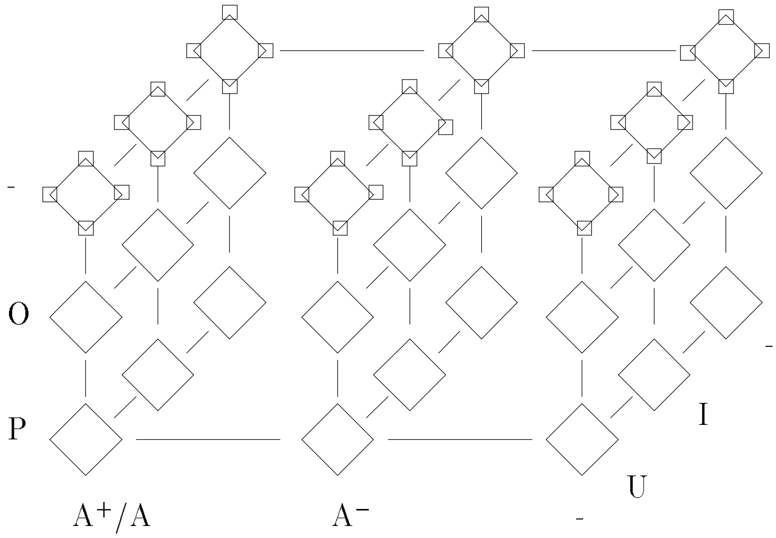
It is obvious that X has a cover C' such that $|C'| \leq K$ iff there is a plan of size $|X| + K$ or less solving Π. □

The problems affected by Theorem 6.4.8 are shown in Figure 6.6.

Theorem 6.4.9 PE is NP-hard for SAS*-BSA$^+$O.

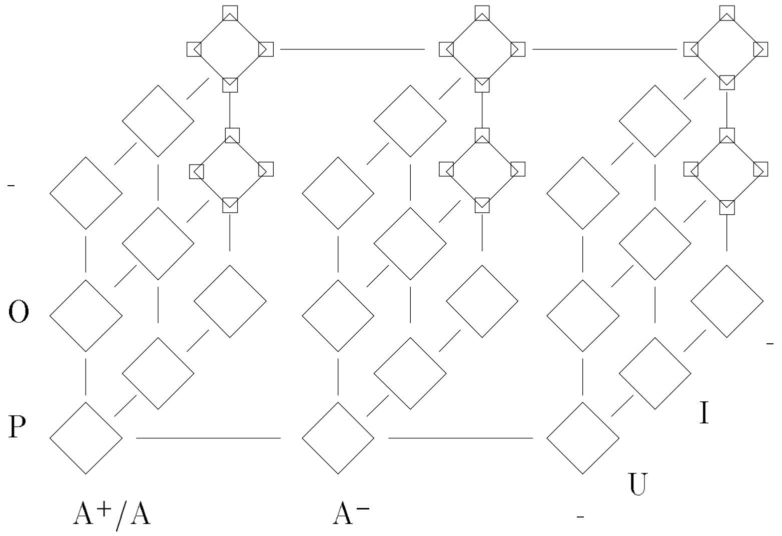
Proof: Proof by reduction from X3C. Let $X = \{x_1, \ldots, x_{3m}\}$ be a set, let $C = \{C_1, \ldots, C_n\}$ be a set of 3-element subsets of X. Define a SAS*-BSA$^+$O instance $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ such that

- $\mathcal{V} = \{x_k \mid 1 \leq k \leq 3m\}$;
- $\mathcal{D}_v = \{0, 1\}$ for all $v \in \mathcal{V}$;



□ NP-hard for **BPG**

Figure 6.6: Hardness for **BPG** *by SAS*-UBSA+ (Theorem 6.4.8.)*



□ NP-hard for **PG**

Figure 6.7: Hardness for **PG** *by SAS*-BSA$^+$O (Theorem 6.4.9.)*

- $\mathcal{O} = \{o_k^+ \mid 1 \leq l \leq n\}$ where for $1 \leq k \leq 3m$ and $1 \leq l \leq n$,

$$\begin{aligned}
\mathsf{pre}(o_l^+)[x_k] &= \begin{cases} 0 & \text{if } x_k \in C_l \\ \mathsf{u} & \text{otherwise} \end{cases} \\
\mathsf{post}(o_l^+)[x_k] &= \begin{cases} 1 & \text{if } x_k \in C_l \\ \mathsf{u} & \text{otherwise} \end{cases} \\
\mathsf{prv}(o_l^+)[v] &= \mathsf{u}
\end{aligned}$$

- $s_0[x_k] = 0$ for $1 \leq k \leq 3m$,
- $s_*[x_k] = 1$ for $1 \leq k \leq 3m$.

Observe that restriction O is trivially satisfied since no operator has any prevail-conditions. By the construction of the instance, X has an exact cover iff there is a plan solving Π so the theorem follows. □

Figure 6.7 shows the problems being NP-hard by the previous theorem.

Theorem 6.4.10 **PE** is NP-hard for SAS*-UBA$^+$.

Proof: Proof by reduction from SAT. Let $U = \{u_1, \ldots, u_m\}$ be a set of boolean variables and let $C = \{C_1, \ldots, C_n\}$ be a set of clauses over U. Define a SAS*-UBA+ instance $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ such that

- $\mathcal{V} = \{u_k \mid 1 \leq k \leq m\} \cup \{c_l \mid 1 \leq l \leq n\}$.
- $\mathcal{D}_{u_k} = \{F, T\}$ for $1 \leq k \leq m$ and $\mathcal{D}_{c_l} = \{F, T\}$ for $1 \leq l \leq n$.
- $\mathcal{O} = \{o_k^+ \mid 1 \leq k \leq m\} \cup \{o_{l,p} \mid 1 \leq l \leq n \text{ and } u_p \in C_l\}$ where for $1 \leq i \leq m$, $1 \leq j \leq n$ and $u_p \in C_l$,

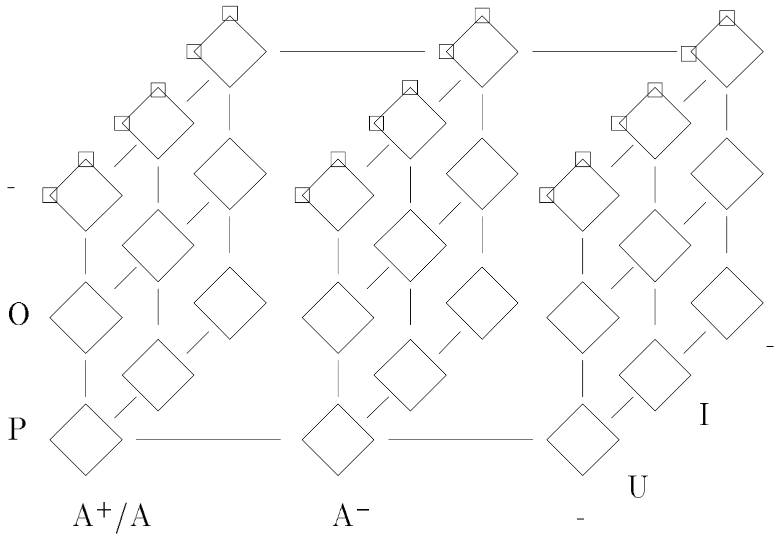
$$\begin{aligned}
\mathsf{pre}(o_k^+)[u_i] &= \begin{cases} F & \text{if } i = k \\ \mathsf{u} & \text{otherwise} \end{cases} \\
\mathsf{pre}(o_k^+)[c_j] &= \mathsf{u} \\
\mathsf{post}(o_k^+)[u_i] &= \begin{cases} T & \text{if } i = k \\ \mathsf{u} & \text{otherwise} \end{cases} \\
\mathsf{post}(o_k^+)[c_j] &= \mathsf{u} \\
\mathsf{prv}(o_k^+)[u_i] &= \mathsf{u} \\
\mathsf{prv}(o_k^+)[c_j] &= F
\end{aligned}$$

$$\begin{aligned}
\mathsf{pre}(o_{l,p})[u_i] &= \mathsf{u} \\
\mathsf{pre}(o_{l,p})[c_j] &= \begin{cases} F & \text{if } j = l \\ \mathsf{u} & \text{otherwise} \end{cases} \\
\mathsf{post}(o_{l,p})[u_i] &= \mathsf{u} \\
\mathsf{post}(o_{l,p})[c_j] &= \begin{cases} T & \text{if } j = l \\ \mathsf{u} & \text{otherwise} \end{cases} \\
\mathsf{prv}(o_{l,p})[u_i] &= \begin{cases} F & \text{if } i = l \text{ and } \overline{u_i} \in C_l \\ T & \text{if } i = l \text{ and } u_i \in C_l \\ \mathsf{u} & \text{otherwise} \end{cases} \\
\mathsf{prv}(h_{l,p})[c_j] &= \mathsf{u}
\end{aligned}$$

- $s_0[u_k] = F$ for $1 \leq k \leq m$. $s_0[c_l] = F$ for $1 \leq l \leq n$.
- $s_*[u_k] = \mathsf{u}$ for $1 \leq k \leq m$. $s_*[c_l] = T$ for $1 \leq l \leq n$.

Obviously there exists a satisfying truth assignment for C iff there exists a plan solving Π. $\square$

Figure 6.8 shows which problems that are NP-hard by Theorem 6.4.10.



□ NP-hard for **PG**

Figure 6.8: Hardness for **PG** *by SAS*-UBA+ (Theorem 6.4.10.)*

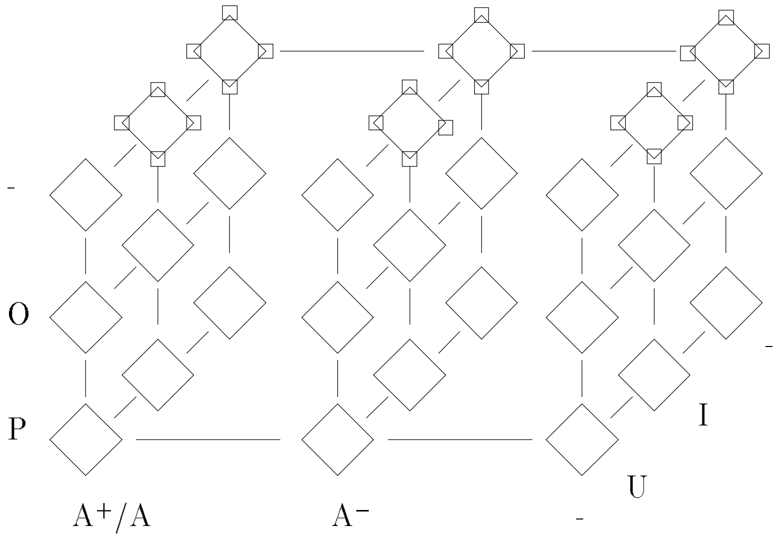
Theorem 6.4.11 **PE** is NP-hard for SAS*-BSIA$^+$.

Proof: Proof by reduction from SAT. Let $U = \{u_1, \ldots, u_m\}$ be a set of boolean variables and let $C = \{C_1, \ldots, C_n\}$ be a set of clauses over U. Define a SAS*-BSIA$^+$ instance $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ such that

- $\mathcal{V} = \{u_k^F, u_k^T \mid 1 \leq k \leq m\} \cup \{c_l \mid 1 \leq l \leq n\}$.
- $\mathcal{D}_{u_k^F} = \{F, B\}$, $\mathcal{D}_{u_k^T} = \{B, T\}$ and $\mathcal{D}_{c_l} = \{F, T\}$ for $1 \leq k \leq m$ and $1 \leq l \leq n$.
- $\mathcal{O} = \{o_k^+ \mid 1 \leq k \leq m\} \cup \{o_{l,p} \mid 1 \leq l \leq n \text{ and } u_p \in C_l\}$ where for $1 \leq i \leq m$, $1 \leq j \leq n$ and $u_p \in C_l$,

$$
\begin{aligned}
\mathsf{pre}(o_k^+)[u_i^F] &= \begin{cases} F & \text{if } i = k \\ \mathsf{u} & \text{otherwise} \end{cases} \\
\mathsf{pre}(o_k^+)[u_i^T] &= \begin{cases} B & \text{if } i = k \\ \mathsf{u} & \text{otherwise} \end{cases} \\
\mathsf{pre}(o_k^+)[c_j] &= \mathsf{u} \\
\mathsf{post}(o_k^+)[u_i^F] &= \begin{cases} B & \text{if } i = k \\ \mathsf{u} & \text{otherwise} \end{cases} \\
\mathsf{post}(o_k^+)[u_i^T] &= \begin{cases} T & \text{if } i = k \\ \mathsf{u} & \text{otherwise} \end{cases} \\
\mathsf{post}(o_k^+)[c_j] &= \mathsf{u} \\
\mathsf{prv}(o_k^+)[u_i] &= \mathsf{u} \\
\mathsf{prv}(o_k^+)[c_j] &= F
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{pre}(o_{l,p})[u_i] &= \mathsf{u} \\
\mathsf{pre}(o_{l,p})[c_j] &= \begin{cases} F & \text{if } j = l \\ \mathsf{u} & \text{otherwise} \end{cases} \\
\mathsf{post}(o_{l,p})[u_i] &= \mathsf{u} \\
\mathsf{post}(o_{l,p})[c_j] &= \begin{cases} T & \text{if } j = l \\ \mathsf{u} & \text{otherwise} \end{cases} \\
\mathsf{prv}(o_{l,p})[u_i^F] &= \begin{cases} F & \text{if } i = l \text{ and } \overline{u_i} \in C_l \\ \mathsf{u} & \text{otherwise} \end{cases} \\
\mathsf{prv}(o_{l,p})[u_i^T] &= \begin{cases} T & \text{if } i = l \text{ and } u_i \in C_l \\ \mathsf{u} & \text{otherwise} \end{cases} \\
\mathsf{prv}(o_{l,p})[c_j] &= \mathsf{u}
\end{aligned}
$$



□ NP-hard for **PG**

Figure 6.9: Hardness for **PG** *by SAS*-BSIA+ (Theorem 6.4.11.)*

- $s_0[u_k^F] = F$ and $s_0[u_k^T] = B$ for $1 \leq k \leq m$. $s_0[c_l] = F$ for $1 \leq l \leq n$.
- $s_*[u_k^F]s_*[u_k^T] = \mathsf{u}$ for $1 \leq k \leq m$. $s_*[c_l] = T$ for $1 \leq l \leq n$.

It is obvious that there is a satisfying truth assignment for C iff there exists a plan solving Π. □

The problems that are NP-hard by the previous theorem are indicated in Figure 6.9. Finally, we state a result about inherently intractable problems, previously found in the literature as indicated. The effect on the problems in the lattice is shown in Figure 6.10.

Theorem 6.4.12 [Bäckström and Nebel, 1993] Both SAS*-PUB and SAS*-PBS have instances with exponentially sized minimal solutions (and are thus inherently intractable).

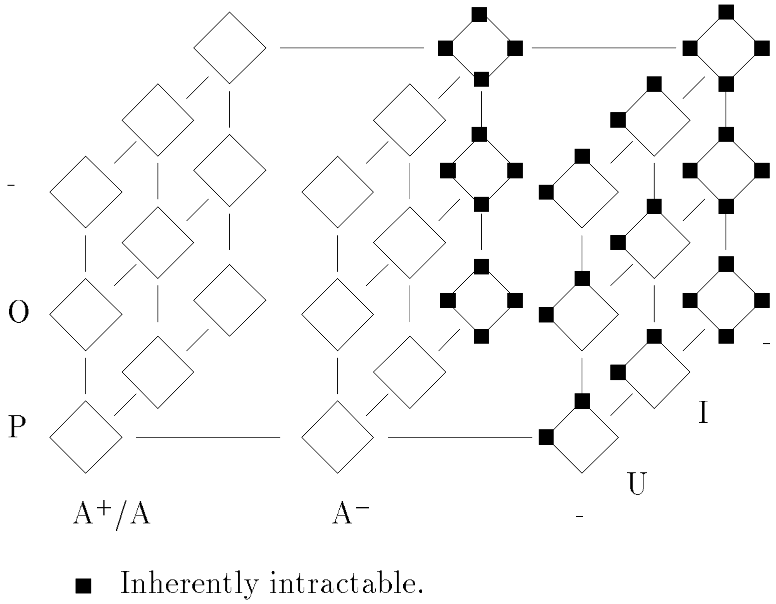


Figure 6.10: Intractability for **PG** *by SAS*-PUB and SAS*-PBS. (Theorem 6.4.12)*

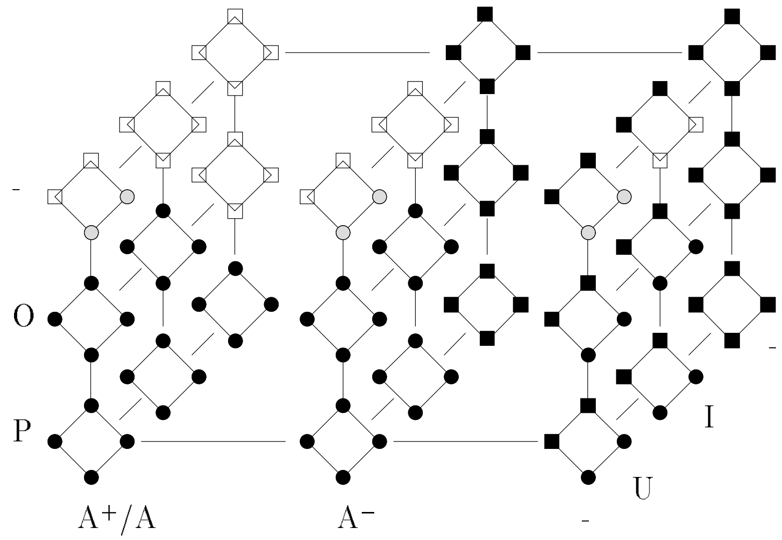
	O	-
US	**BPG** tractable	**PG** tractable
IA^-	**BPG** tractable	Intractable
-	Intractable	Intractable

Table 6.1: Simplified map over the complexity of SAS plan generation*

6.5 Summary

By combining the hardness and easiness results of the previous sections, we get an exhaustive map over the complexity results. The resulting lattice is shown in Figure 6.11.

By inspecting the lattice more closely, we can provide a simple characterization of when the problems are tractable and intractable respectively. The simplified presentation is shown in Table 6.1. We see that if an instance Π does not satisfy the I and A^- restrictions, then we cannot solve it in polynomial-time with any of the methods presented in this thesis. If it satisfies IA^- but not restriction O, then it must satisfy US in order to be tractable. Note that, in this case, we still cannot find a minimal solution to the instance in polynomial time. Finally, if Π is prevail-order-preserving, then we can find a minimal solution in polynomial time by using the A-transform and the SAS^+-IAO algorithm.



● Polynomial for **PG** and **BPG**

○ Polynomial for **PG**, NP-equivalent for **BPG**

□ NP-equivalent for **PG** and **BPG**

■ Inherently intractable.

Figure 6.11: Complexity of SAS plan generation*

Chapter 7

The LEGO Car Factory

In this chapter, we show how to plan for a semi-realistic miniature version of an industrial process, the LEGO car factory [Strömberg, 1991]. The chapter is structured as follows. First the LEGO car factory is described in Section 7.1 and we show how to model it using the SAS$^+$ formalism. A extended version of the SAS$^+$-IAO algorithm is presented in Section 7.2 and in Section 7.3 we apply this algorithm to the LEGO car factory.

7.1 The LEGO Car Factory

Our application example is an automated assembly line for LEGO cars [Strömberg, 1991], which is used for undergraduate laboratory sessions in digital control at the Department of Electrical Engineering at Linköping University. The students are faced with the task of writing a program to control this assembly line using the graphical language GRAFCET [IEC, 1988]. GRAFCET is tailored to implementing industrial sequential control and it resembles Petri Nets.

The main operations for assembling a LEGO car are shown in Figure 7.1. The assembly line consists of two similar halves, the first mounting the chassis parts on the chassis (see Figure 7.2) and the second mounting the top (see Figure 7.3).

The first half of the LEGO car factory is presented in Figure 7.2. The chassis is initially stored up-side down in the chassis magazine (*cm*). It enters the conveyor belt by using the chassis feeder (*c-feeder*), and is transported to the chassis parts magazine (*cpm*) where the chassis parts are fed onto the chassis using the chassis parts feeder (*cp-feeder*). The chassis is then transported to the chassis press (*cp*), where the chassis is pressed together.

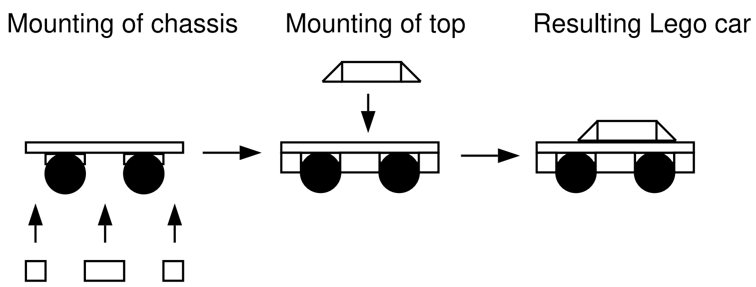


Figure 7.1: Assembling a LEGO car.

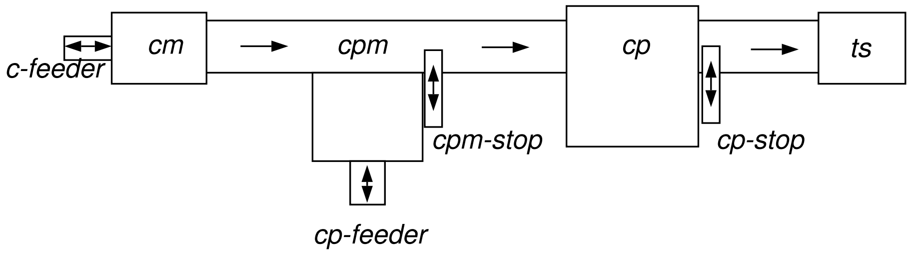


Figure 7.2: The first half of the LEGO car factory.

It is then transported to the turn station (*ts*) where the chassis is turned upright and enters the second half of the factory (Figure 7.3) where it is placed on the chassis lift (*cl*). It is lifted up, placed on the conveyor belt (*ocvB*) and transported to the top magazine (*tm*) where a top is fed onto the chassis by the top feeder (*t-feeder*). The chassis is then transported to the top press (*tp*) where the top is pressed tight onto the chassis. From there it is transported to the end of the conveyor belt (*sf*) and placed so that the storage feeder (*st-feeder*) can push the chassis into a buffer storage (*st*).

The conveyor belt used to transport the chassis runs continously. Hence, stopper bars (*cpm-stop, cp-stop, tm-stop, tp-stop*) are pushed out in front of the chassis at the four work-stations *cpm, cp, tm* and *tp*, holding the car fixed, sliding on the belt (see Figure 7.2 and Figure 7.3). The car cannot pass the work-station until the corresponding stopper bar is withdrawn.

Figure 7.4 shows one of the work-stations in more detail, namely the one where the top is put onto the chassis (*tm* in Figure 7.3). The chassis is held fixed at the top storage (A) by the stopper bar (B). The tops are stored in a pile and the feeder (C) is used to push out the lowermost top onto the

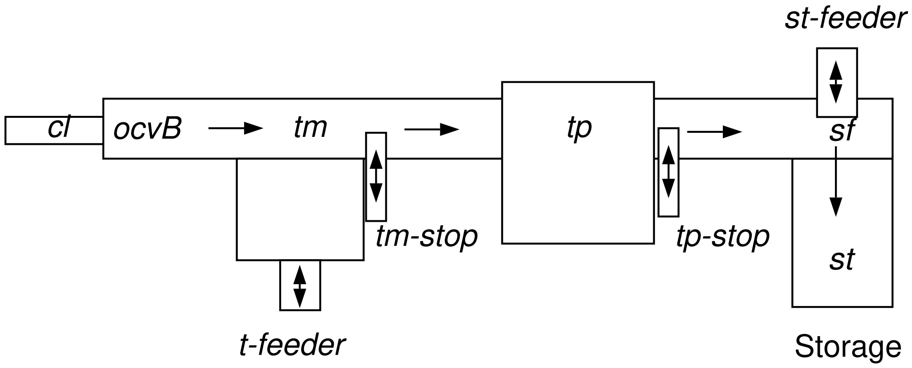


Figure 7.3: The second half of the LEGO car factory.

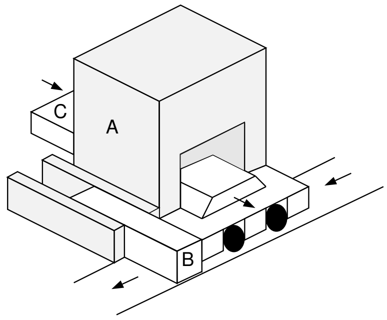


Figure 7.4: Putting the top onto the chassis.

chassis. When the top is on the chassis, the feeder is withdrawn and then the stopper bar is withdrawn, thus allowing the chassis to move on to the next work-station.

We continue by modelling the LEGO car factory as a SAS$^+$ instance. The first problem is to decide which state variables to use. Once this is done, it is straighforward to define the operators. We use the state variables shown in Table 7.1.

The variable *pos* gives the position of the chassis, and the corresponding positions are given in Figure 7.2 and Figure 7.3. The stopper bars and the corresponding variable names are also marked in these figures, as well as the variable names for the feeders. For the feeders and the stopper bars the value *ext* means that the feeder (or stopper bar) is extended, while *rtr* means that it is retracted. The variable *turner* tells if the turner (*ts* in Figure 7.2) is turned towards the first half of the factory (A) or towards the second half

Variable	Values
pos	cm, cpm, cp, ts, cl, ocvB, tm, tp, sf, st
turner	A, B
cp-status	off, on, pressed
t-status	off, on, pressed
cp-press	down, up
t-press	down, up
clift	down, up
c-feeder, cp-feeder, t-feeder, st-feeder	ext, rtr
cpm-stop, cp-stop, tm-stop, tp-stop	ext, rtr

Table 7.1: State variables $\mathcal{V}$ and their associated domains of values $\mathcal{D}_v$.

of the factory (B). The two variables *cp-status* and *t-status* give the status of the chassis parts and the top, respectively. The other variables should be obviuos from the table and the figures.

Using the variables defined in Table 7.1 we can define operators as in Tables 7.2 and 7.3. Additionally there are two operators for each feeder and each stopper bar for retracting and extending the feeder or stopper bar respectively. The operators corresponding to the chassis feeder are denoted *extend-c-feeder* and *retract-c-feeder*. For instance, the pre-condition of *extend-c-feeder* is that *c-feeder* = *rtr*, the post-condition is that *c-feeder* = *ext* and there is no prevail-condition. The other operators corresponding to the feeders and stopper bars are denoted in a similar manner.
By studying the instance, it is easy to see that it satisfies both restriction I and O. However, the proposed modelling does not satisfy the A restriction since several of the variables are not acyclic wrt. requestable values. In the next section, we modify the SAS$^+$-IAO algorithm to be able to generate plans for this modelling of the LEGO car factory.

7.2 Planning Algorithm

In this section we present the extended version of the SAS$^+$-IAO algorithm that can handle the LEGO car factory. As already mentioned, the idea is to partition the original SAS$^+$ instance into two separate instances, both

Operator	Pre	Post	Prevail
cm2cpm	pos = cm	pos = cpm	c-feeder = ext cp-feeder = rtr cpm-stop = ext
cpm2cp	pos = cpm	pos = cp	cpm-stop = rtr cp-stop = ext cp-press = up
cp2ts	pos = cp	pos = ts	turner = A cp-stop = rtr cp-press = up
ts2cl	pos = ts	pos = cl	turner = B clift = down
cl2ocvB	pos = cl	pos = ocvB	clift = up
ocvB2tm	pos = ocvB	pos = tm	tm-stop = ext t-feeder = rtr
tm2tp	pos = tm	pos = tp	tm-stop = rtr tp-stop = ext t-press = up
tp2sf	pos = tp	pos = sf	tp-stop = rtr st-feeder = rtr t-press = up
sf2st	pos = sf	pos = st	st-feeder = ext
put-cp	cp-status = off	cp-status = on	pos = cpm cp-feeder = ext
press-cp	cp-status = on	cp-status = pressed	cp-press = down pos = cp
put-top	t-status = off	t-status = on	pos = tm t-feeder = ext
press-top	t-status = on	t-status = pressed	pos = tp t-press = down

Table 7.2: Operators with prevail-conditions.

Operator	Pre	Post	Prevail
A2B	turner = A	turner = B	-
B2A	turner = B	turner = A	-
cl-down	clift = up	clift = down	-
cl-up	clift = down	clift = up	-
cp-press-down	cp-press = up	cp-press = down	-
cp-press-up	cp-press = down	cp-press = up	-
t-press-down	t-press = up	t-press = down	-
t-press-up	t-press = down	t-press = up	-

Table 7.3: Operators without prevail-conditions.

satisfying the IAO restrictions. One of the new instances is then solved and its solution constitutes a skeleton to be filled in by solving subproblems from the second instance. This process is referred to as *interweaving* and can be viewed as a restricted variant of the more general concept *refinement*, as used in hierarchical abstraction [Knoblock, 1994]. In fact, the whole method we use can be viewed as a variant of two-level abstraction. However, while hierarchical abstraction is a general method which is not formally well understood—it can at some occasions speed up planning considerably [Knoblock, 1991b] and at other occasions be disastrous [Bäckström and Jonsson, 1995]—our, more restricted method, is provably correct and runs in polynomial time.

Before presenting the method formally, we need a number of definitions. We begin by showing how a SAS$^+$ problem instance can be restricted to take only a subset of the variables into account.

Definition 7.2.1 Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ be a SAS$^+$ instance, $s \in \mathcal{S}^+$ and $\mathcal{V}' \subseteq \mathcal{V}$. Then, the **restriction wrt.** $\mathcal{V}'$ of states and operators are defined as follows:

- $s \Cap \mathcal{V}' = \langle s[v_{i_1}], \ldots, s[v_{i_n}] \rangle$ where $\mathcal{V}' = \{v_{i_1}, \ldots v_{i_n}\}$
- Let $o \in \mathcal{O}$. Then, $o \Cap \mathcal{V}' = \langle \mathsf{pre}(o) \Cap \mathcal{V}', \mathsf{post}(o) \Cap \mathcal{V}', \mathsf{prv}(o) \Cap \mathcal{V}', \mathsf{name}(o) \rangle$.
- $\mathcal{O} \Cap \mathcal{V}' = \{o \Cap \mathcal{V}' | o \in \mathcal{O}\}$.

The restriction of Π wrt. $\mathcal{V}'$ can now be defined as

$$\Pi \Cap \mathcal{V}' = \langle \mathcal{V}', \mathcal{O} \Cap \mathcal{V}', s_0 \Cap \mathcal{V}', s_* \Cap \mathcal{V}' \rangle.$$

Having a method for restricting operators, we also need a method for reconstructing restricted operators. Here, we make use of the name each operator is equipped with.

Definition 7.2.2 Let o be an operator and $\mathcal{O}$ a set of operators. Then, $\downarrow^{\mathcal{O}}(o)$ is defined as the unique operator $o' \in \mathcal{O}$ such that $\mathsf{name}(o) = \mathsf{name}(o')$. We generalize $\downarrow^{\mathcal{O}}$ to operate on plans in the obvious way, namely $\downarrow^{\mathcal{O}}((o_1;\ldots;o_n)) = (\downarrow^{\mathcal{O}}(o_1);\ldots;\downarrow^{\mathcal{O}}(o_n))$.

Observe that we have assumed that every operator has a unique name so the previous definition is sound. We continue by defining a certain partitioning of planning instances having, as we will see, favourable properties.

Definition 7.2.3 Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ be a SAS$^+$ instance and let $\mathcal{V}_1$, $\mathcal{V}_2$ be disjoint subsets of $\mathcal{V}$. Then, $\mathcal{V}_1$ is **independent** of $\mathcal{V}_2$ iff every operator $o \in \mathcal{O}$ affecting some variable in $\mathcal{V}_1$ satisfies $\mathsf{pre}(o) \Cap \mathcal{V}_2 = \mathsf{post}(o) \Cap \mathcal{V}_2 = \mathsf{prv}(o) \Cap \mathcal{V}_2 = \bot$.

An immediate consequence of the previous definition is that every operator $o \in \mathcal{O}$ affecting some variable in $\mathcal{V}_2$ satisfies $\mathsf{pre}(o) \Cap \mathcal{V}_1 = \mathsf{post}(o) \Cap \mathcal{V}_1 = \bot$.

Definition 7.2.4 Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ be a SAS$^+$ instance. Then Π is **reachable** iff for any two states $s_1, s_2 \in \mathcal{S}$ the planning problem $\langle \mathcal{V}, \mathcal{O}, s_1, s_2 \rangle$ is solvable.

Reachability means that every problem instance can be solved regardless of the initial and goal states. Thus, reachability is the same as a strongly connected state transition graph. Note that even if reachability only depends on $\mathcal{V}$ and $\mathcal{O}$ we say that Π is reachable, for convenience.

We can now show that if a planning instance Π is split into two independent problems where one of the subproblems is solvable and the other is reachable, then the original instance has a solution.

Theorem 7.2.5 Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ be a SAS$^+$ instance and let $\mathcal{V}_1, \mathcal{V}_2$ be disjoint subsets of $\mathcal{V}$ such that $\mathcal{V}_1 \cup \mathcal{V}_2 = \mathcal{V}$. If $\mathcal{V}_1$ is independent of $\mathcal{V}_2$, $\Pi \Cap \mathcal{V}_1$ is reachable and $\Pi \Cap \mathcal{V}_2$ is solvable then Π is solvable.

Proof: Suppose there exists a plan $\omega = \langle o_1; \ldots; o_n \rangle$ solving $\Pi \Cap \mathcal{V}_2$. Let ω_0 be a plan solving the SAS$^+$ instance $\langle \mathcal{V}_1, \mathcal{O}, s_0, \mathsf{prv}(\downarrow^{\mathcal{O}}(o_1)) \rangle \Cap \mathcal{V}_1$. Define recursively ω_k, $1 \leq k \leq n-1$ such that ω_k is a solution to the instance $\langle \mathcal{V}_1, \mathcal{O}, \mathit{Result}((\omega_0; \ldots; \omega_{k-1}), s_0 \Cap \mathcal{V}_1), \mathsf{prv}(\downarrow^{\mathcal{O}}(o_{k+1})) \Cap \mathcal{V}_1 \rangle$. Finally, let ω_n be a solution to the instance $\langle \mathcal{V}_1, \mathcal{O}, \mathit{Result}((\omega_0; \ldots; \omega_{n-1}), s_0 \Cap \mathcal{V}_1), s_* \rangle \Cap \mathcal{V}_1$. Since $\Pi \Cap \mathcal{V}_1$ is reachable, $\omega_0, \ldots, \omega_n$ all exist. Now, let us consider the plan $\omega' = \downarrow^{\mathcal{O}}(\omega_0; o_1; \omega_1; \ldots; \omega_{n-1}; o_n; \omega_n)$. By the construction of $\omega_0, \ldots, \omega_{n-1}$, all pre- and prevail-conditions of $\downarrow^{\mathcal{O}}(o_1), \ldots, \downarrow^{\mathcal{O}}(o_n)$ are satisfied because $\omega = (o_1; \ldots; o_n)$ is a valid plan for $\Pi \Cap \mathcal{V}_2$ by assumption. Furthermore, all pre- and prevail-conditions of the operators in $\downarrow^{\mathcal{O}}(\omega_0; \ldots; \omega_n)$ are satisfied because $(\omega_0; \ldots; \omega_n)$ is a valid plan for $\Pi \Cap \mathcal{V}_1$ and $\mathcal{V}_1$ is independent of $\mathcal{V}_2$. It remains to show that $s_* \sqsubseteq \mathit{Result}(\omega', s_0)$ but this follows immediately since we know that $(o_1; \ldots; o_n)$ is a valid plan for $\Pi \Cap \mathcal{V}_2$ and $(\omega_0; \ldots; \omega_n)$ is a valid plan for $\Pi \Cap \mathcal{V}_1$ and $\mathcal{V}_1$ is independent of $\mathcal{V}_2$. $\square$

An algorithm implementing the method in the proof of Theorem 7.2.5 is shown in Figure 7.5. The sets $\mathcal{V}_1$ and $\mathcal{V}_2$ are such that $\Pi \Cap \mathcal{V}_1$ and $\Pi \Cap \mathcal{V}_2$ both satisfy the IAO restriction in the previous section. *Plan* is the algorithm for SAS$^+$-IAO plan generation.

1 **procedure** $\mathit{Plan2}(\langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle), \mathcal{V}_1, \mathcal{V}_2$;
2 $\langle o_1, \ldots, o_n \rangle \leftarrow \mathit{Plan}(\langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle \Cap \mathcal{V}_2)$
3 $\omega_0 \leftarrow \mathit{Plan}(\langle \mathcal{V}, \mathcal{O}, s_0, \mathsf{prv}(o_1) \rangle \Cap \mathcal{V}_1)$
4 **for** $k = 1, \ldots, n-1$ **do**
5 $\quad \omega_k \leftarrow \mathit{Plan}(\langle \mathcal{V}_1, \mathcal{O}, \mathit{Result}((\omega_0; \ldots; \omega_{k-1}), s_0 \Cap \mathcal{V}_1), \mathsf{prv}(o_{k+1}) \rangle \Cap \mathcal{V}_1)$
6 $\omega_n \leftarrow \mathit{Plan}(\langle \mathcal{V}_1, \mathcal{O}, \mathit{Result}((\omega_0; \ldots; \omega_{n-1}), s_0 \Cap \mathcal{V}_1), s_* \rangle \Cap \mathcal{V}_1)$
7 **return** $(\omega_0; o_1; \omega_1; o_2; \ldots; \omega_{n-1}; o_n; \omega_n)$

Figure 7.5: Planning algorithm

It is obvious from the proof of Theorem 7.2.5 that the algorithm is correct, and since *Plan* is polynomial the resulting algorithm is polynomial.

Theorem 7.2.6 The *Plan2* algorithm runs in $O(|\mathcal{V}|^3 |\mathcal{O}|^4 \max_{v \in \mathcal{V}}(|\mathcal{D}_v^+|)^5)$ time.

Proof: By Corollary 5.4.3, the length of the plan generated in line 2 is bounded from above by $2|\mathcal{O}|$. Since *Plan* runs in $O(|\mathcal{V}|^3 |\mathcal{O}|^2 \max_{v \in \mathcal{V}}(|\mathcal{D}_v^+|)^5)$ time by Theorem 5.4.8, the theorem follows. $\square$

7.3 Planning for the LEGO Car Factory

We can now apply the planning algorithm in Figure 7.5 to the LEGO car factory. First we define the two subsets defining the partition of the problem. Using the variables defined in Table 7.1 we can define two subsets as follows:

$$\mathcal{V}_1 = \{\text{turner, cp-press, t-press, clift, c-feeder, cp-feeder, t-feeder, st-feeder, cpm-stop, cp-stop, tm-stop, tp-stop}\}$$

$$\mathcal{V}_2 = \{\text{pos, cp-status, t-status}\}$$

Hence, the operators in Table 7.3 together with all operators for extending or retracting feeders or stopper bars are the operators that affect variables in $\mathcal{V}_1$. Furthermore the operators given in Table 7.2 are the operators that affect variables in $\mathcal{V}_2$.

It is easy to see that $\Pi \Cap \mathcal{V}_1$ and $\Pi \Cap \mathcal{V}_2$ satisfies the IAO restriction and that $\mathcal{V}_1$ is independent of $\mathcal{V}_2$. Furthermore, for every $v \in \mathcal{V}_1$ there are operators for setting each value (for example there is one operator for extending the c-feeder and one operator for retracting it). Obviously $\Pi \Cap \mathcal{V}_1$ is reachable, and the algorithm in Figure 7.5 can be applied.

Depending on how we choose the initial state and the goal state we can plan for different cases. Here we show a plan for normal operation, *ie.* the goal is to assembly a LEGO car. Clearly, it is straightforward to modify this to plan for other situations, such as error recovery. Here the goal state is that the chassis should be in the buffer storage ($pos = st$) and the top and chassis parts should be pressed onto the chassis (*cp-status* = *pressed* and *t-status* = *pressed*). All other state variables are undefined and can thus have any value.

Suppose that the initial state is given as follows. The chassis is placed in the chassis magazine ($pos = cm$), there is no chassis parts on the chassis (*cp-status* = *off*) and there is no top on the chassis (*t-status* = *off*). Furthermore the turner is turned towards the first half of the factory (*turner* = A), all feeders and stopper bars are extended and the chassis press and the top press are both in their down position.

Applying the algorithm in Figure 7.5 results in a plan as in Figure 7.6. The solid arrows indicate the ordering of the actions generated in line 2 of the algorithm. The dashed arrows indicate the ordering of actions introduced by the interweaving process.

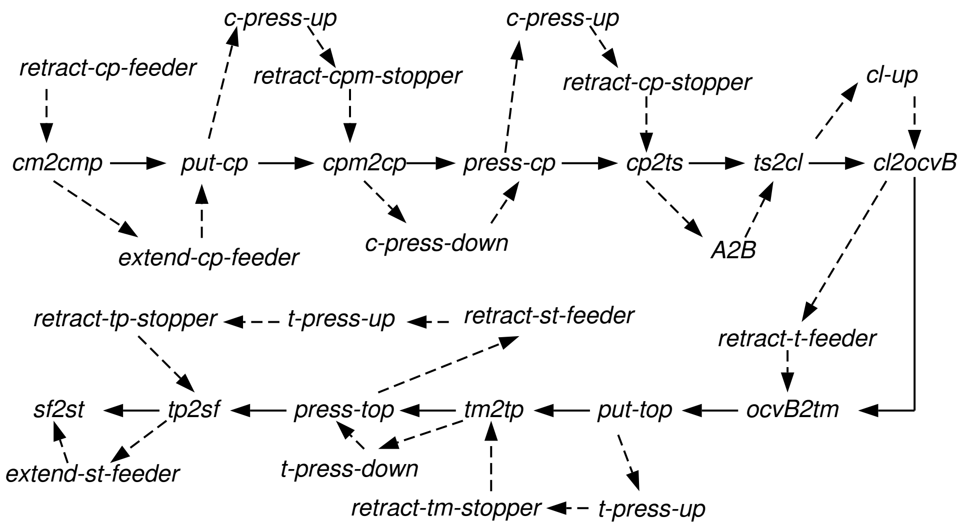


Figure 7.6: Resulting plan for the LEGO car factory.

Chapter 8

Concluding Remarks

We begin this chapter with a summary of the thesis and its major contributions and continue with a discussion of how this research may be continued.

8.1 Summary

We have identified two sets of restrictions, IAO and $IA^{-}O$, allowing for the generation of minimal plans in polynomial time for a planning formalism, SAS^{+}, using multi-valued state variables. This extends the tractability borderline for planning, by allowing for more general problems than previously reported in the literature to be solved tractably. In contrast to most restrictions in the literature, ours are structural restrictions. However, they are restrictions on the transition graph for each state variable in isolation, rather than for the whole state space, so they can be tested in polynomial time.

By analysing the complexity of plan generation under all combinations of restrictions, considering both these new structural restrictions and the previously analysed syntactical ones, we can conclude that SAS^{+}-$IA^{-}O$ bounded plan generation is maximally tractable under the nine different restrictions studied in this thesis. We have also shown SAS^{*}-US (and consequently SAS^{+}-US) unbounded plan generation cannot be further generalized with preserved tractability by replacing US with any combination of our studied syntactical or structural restrictions. By providing some additional hardness results, we have built a map over the complexity of planning for all combinations of both the syntactical and structural restrictions, considering both bounded and unbounded plan generation.

Furthermore, we have applied our previous results on tractable planning to an application example in automatic control—an assembly line for LEGO cars. This example does not quite fit within the restriction of any previously presented tractable planning class, including the new SAS$^+$-IAO and SAS$^+$-IA$^-$O classes. This fact provided feedback for modifying the theory and we have presented an extended variant of the previous SAS$^+$-IAO class and its associated algorithm, capable of handling the LEGO car factory.

8.2 Future Work

This section discusses some ideas for how to continue the work presented in this thesis.

The most time-consuming part of the SAS$^+$-IAO algorithm is, probably, the fixpoint computation. In the present version of the algorithm, the fixpoint for the *Extend* function is computed by Jacobi iteration, which is known to be unnecessary inefficient in many cases. By replacing the Jacobi iteration with some more elaborate method, we would probably improve the complexity considerably, at least in the average case. Possible candidates are, for instance, *Gauss-Seidel* iteration [Todd, 1962] and *chaotic* iteration [Cousot and Cousot, 1977]. The different characteristics of these methods makes it a non-trivial task to choose the right strategy and to fine-tune it properly. Clearly, this is a possible track for future research.

The SAS$^+$-IA$^-$O problem is a maximal tractable problem wrt. the restrictions studied in this thesis if we want to generate minimal plans. Hence, we cannot remove any of the restrictions on this problem and hope for retained tractability. However, by altering the restrictions, it might be possible to find new tractable classes of problems. We could, for example, modify the algorithm such that the sets $X_1, \ldots, X_m$ of requestable values be partially ordered multisets, allowing a prevailcondition to be produced and destroyed several times—thus relaxing the A restriction.

A natural addition to the previous idea would be to redefine *Extend* as a non-deterministic procedure. Although requiring search and, thus, probably sacrificing tractability, we believe this to be an intriguing alternative to ordinary search-based planning. With this algorithm, the search would take place in a space of (partially ordered) requestable values. Obviously, this is a new approach to planning in comparison with traditional search-based planning which performs search in a space of plans.

Finally, the applicability of tractable planning to real-world problems must be further investigated. We have seen, in the case of the LEGO car factory, that real-world problems can give us important insights into the structure of naturally arising problems. Similarily, the restrictions that define tractable problems can give us valuable hints about how to build and model systems in order to make planning for them tractable. Preferably, we should have a comprehensive view on planning that ranges from the construction of the system, over the systems modelling to the choice and implementation of appropriate planning algorithms. To achieve this, a deeper understanding of the structure of real-world problems is essential.

References

[AAAI-90, 1990] American Association for Artificial Intelligence. *Proceedings of the 8th (US) National Conference on Artificial Intelligence (AAAI-90)*, Boston, MA, USA, August 1990. MIT Press.

[AAAI-91, 1991] American Association for Artificial Intelligence. *Proceedings of the 9th (US) National Conference on Artificial Intelligence (AAAI-91)*, Anaheim, CA, USA, July 1991. AAAI Press/MIT Press.

[AAAI-92, 1992] American Association for Artificial Intelligence. *Proceedings of the 10th (US) National Conference on Artificial Intelligence (AAAI-92)*, San José, CA, USA, July 1992.

[AAAI-93, 1993] American Association for Artificial Intelligence. *Proceedings of the 11th (US) National Conference on Artificial Intelligence (AAAI-93)*, Washington DC, USA, July 1993.

[AAAI-94, 1994] American Association for Artificial Intelligence. *Proceedings of the 12th (US) National Conference on Artificial Intelligence (AAAI-94)*, Seattle, WA, USA, July–August 1994.

[Allen *et al.*, 1990] James Allen, James Hendler, and Austin Tate, editors. *Readings in Planning.* Morgan Kaufmann, San Mateo, CA, 1990.

[Almgren, 1994] Roland Almgren. *Topological Modelling of Assembly Systems for General Applicability of Automatic Assembly Planning.* PhD thesis, Linköping University, Linköping, Sweden, 1994.

[Baase, 1988] Sara Baase. *Computer Algorithms: Introduction and Analysis.* Addison Wesley, Reading, MA, 2nd edition, 1988.

[Bäckström and Jonsson, 1995] Christer Bäckström and Peter Jonsson. Planning with abstraction hierarchies can be exponentially less efficient. In

Chris Mellish, editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, Montréal, PQ, Canada, August 1995. Morgan Kaufmann. To appear.

[Bäckström and Klein, 1990] Christer Bäckström and Inger Klein. Planning in polynomial time. In G Gottlob and W Nejdl, editors, *Expert Systems in Engineering: Principles and Applications. International Workshop*, volume 462 of *Lecture Notes in Artificial Intelligence*, pages 103–118, Vienna, Austria, September 1990. Springer.

[Bäckström and Klein, 1991a] Christer Bäckström and Inger Klein. Parallel non-binary planning in polynomial time. In Reiter and Mylopoulos [1991], pages 268–273.

[Bäckström and Klein, 1991b] Christer Bäckström and Inger Klein. Planning in polynomial time: The SAS-PUBS class. *Computational Intelligence*, 7(3):181–197, August 1991.

[Bäckström and Nebel, 1993] Christer Bäckström and Bernhard Nebel. Complexity results for SAS$^+$ planning. In Ruzena Bajcsy, editor, *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 1430–1435, Chambéry, France, August–September 1993. Morgan Kaufmann.

[Bäckström and Nebel, 1996] Christer Bäckström and Bernhard Nebel. Complexity results for SAS$^+$ planning. *Computational Intelligence*, 1996.

[Bäckström, 1988] Christer Bäckström. A representation of coordinated actions characterized by interval valued conditions. In Zbigniew W Ras and Lorenza Saitta, editors, *Proceedings of the 3rd International Symposium on Methodologies for Intelligent systems (ISMIS-88)*, pages 220–229, Torino, Italy, October 1988. North-Holland.

[Bäckström, 1992a] Christer Bäckström. *Computational Complexity of Reasoning about Plans.* Doctoral dissertation, Linköping University, Linköping, Sweden, June 1992.

[Bäckström, 1992b] Christer Bäckström. Equivalence and tractability results for SAS$^+$ planning. In Bill Swartout and Bernhard Nebel, editors, *Proceedings of the 3rd International Conference on Principles on Knowledge Representation and Reasoning (KR-92)*, pages 126–137, Cambridge, MA, USA, October 1992. Morgan Kaufmann.

[Bäckström, 1994] Christer Bäckström. Planning using transformation between equivalent formalisms: A case study of efficiency. In David Wilkins, editor, *Comparative Analysis of AI Planning Systems*, 1994. Held in conjunction with AAAI-94 [1994].

[Bäckström, 1995] Christer Bäckström. Expressive equivalence of planning formalisms. *Artificial Intelligence, Special Issue on Planning and Scheduling*, 1995. To appear.

[Biundo *et al.*, 1992] Susanne Biundo, Dietmar Dengler, and Jana Koehler. Deductive planning and plan reuse in a command language environment. Research Report RR-92-11, German Research Center for Artificial Intelligence (DFKI),, Saarbrücken, Germany, March 1992.

[Bylander, 1991] Tom Bylander. Complexity results for planning. In Reiter and Mylopoulos [1991], pages 274–279.

[Bylander, 1992] Tom Bylander. Complexity results for serial decomposability. In AAAI-92 [1992], pages 729–734.

[Bylander, 1993] Tom Bylander. An average case analysis of planning. In AAAI-93 [1993], pages 480–485.

[Bylander, 1994] Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69:165–204, 1994.

[Chapman, 1987] David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

[Christensen, 1990] Jens Christensen. A hierarchical planner that generates its own hierarchies. In AAAI-90 [1990], pages 1004–1009.

[Cousot and Cousot, 1977] Patrick Cousot and Radhia Cousot. Automatic synthesis of optimal invariant assertions: Mathematical foundations. *SIGPLAN Notices*, 12(8):1–12, 1977.

[Currie and Tate, 1991] Ken Currie and Austin Tate. O-Plan: The open planning architecture. *Artificial Intelligence*, 52:49–86, 1991.

[Dean and Boddy, 1988] Thomas Dean and Mark Boddy. Reasoning about partially ordered events. *Artificial Intelligence*, 36:375–399, 1988.

[Dijkstra, 1959] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, 1, 1959.

[Erol *et al.*, 1992] Kutluhan Erol, Dana S Nau, and V S Subrahmanian. On the complexity of domain-independent planning. In AAAI-92 [1992], pages 381–386.

[Erol *et al.*, 1994a] Kutluhan Erol, James Hendler, and Dana Nau. HTN planning: Complexity and expressivity. In AAAI-94 [1994], pages 1123–1128.

[Erol *et al.*, 1994b] Kutluhan Erol, James Hendler, and Dana Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In Kristian Hammond, editor, *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS'94)*, pages 249–254, Chicago, IL, USA, June 1994. AAAI Press.

[Fikes and Nilsson, 1971] Richard E Fikes and Nils J Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Fikes *et al.*, 1972] Richard E Fikes, Peter E Hart, and Nils J Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

[Floyd, 1962] R. W. Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.

[Garey and Johnson, 1979] Michael Garey and David Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.

[Green, 1969] Cordell Green. Application of theorem proving to planning. In Donald E Walker and Lewis M Norton, editors, *Proceedings of the 1st International Joint Conference on Artificial Intelligence (IJCAI-69)*, pages 219–239, Washington, DC, USA, May 1969. William Kaufmann. Reprinted in Allen *et al* [1990], pages 67–87.

[Gupta and Nau, 1991] Naresh Gupta and Dana S Nau. Complexity results for blocks-world planning. In AAAI-91 [1991], pages 629–633.

[IEC, 1988] IEC. Preparation of function charts for control systems. Publication 848:1988, IEC, 1988.

[Johnson *et al.*, 1988] David S Johnson, Mihalis Yannakakis, and Christos H Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27:119–123, 1988.

[Johnson, 1990] David S Johnson. A catalog of complexity classes. In van Leeuwen [1990], chapter 2, pages 67–161.

[Jonsson and Bäckström, 1994a] Peter Jonsson and Christer Bäckström. Complexity results for state-variable planning under mixed syntactical and structural restrictions. In Philippe Jorrand, editor, *Proceedings of the 6th International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA-94)*, Sofia, Bulgaria, September 1994. World Scientific Publishing.

[Jonsson and Bäckström, 1994b] Peter Jonsson and Christer Bäckström. Complexity results for state-variable planning under mixed syntactical and structural restrictions. Research report LiTH-IDA-R-95-17, Department of Computer and Information Science, Linköping University, 1994.

[Jonsson and Bäckström, 1994c] Peter Jonsson and Christer Bäckström. Tractable planning with state variables by exploiting structural restrictions. In AAAI-94 [1994], pages 998–1003.

[Jonsson and Bäckström, 1994d] Peter Jonsson and Christer Bäckström. Tractable planning with state variables by exploiting structural restrictions. Research report LiTH-IDA-R-95-16, Department of Computer and Information Science, Linköping University, 1994.

[Klein *et al.*, 1995] Inger Klein, Peter Jonsson, and Christer Bäckström. Tractable planning for an assembly line. Research Report LiTH-ISY-R-1717, Department of Electrical Engineering, Linköping University, Linköping, Sweden, jan 1995.

[Klein, 1993] Inger Klein. *Automatic Synthesis of Sequential Control Charts.* Doctoral dissertation, Linköping University, Linköping, Sweden, April 1993.

[Knoblock, 1991a] Craig A. Knoblock. *Automatically Generating Abstraction Hierarchies for Problem Solving.* PhD thesis, Carnegie-Mellon University, Pittsburgh, PA, USA, May 1991. Available as Technical Report CMU-CS-91-120.

[Knoblock, 1991b] Craig A Knoblock. Search reduction in hierarchical problem solving. In AAAI-91 [1991], pages 686–691.

[Knoblock, 1994] Craig A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68:243–302, 1994.

[Korf, 1987] Richard E Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33:65–88, 1987.

[Levesque, 1988] Hector J Levesque. Logic and the complexity of reasoning. *Journal of Philosophical Logic*, 17:355–389, 1988.

[Mädler, 1992] Fritz Mädler. Towards structural abstraction. In James Hendler, editor, *Artificial Intelligence Planning Systems: Proceedings of the 1st International Conference (AIPS'92)*, pages 163–171, College Park, MD, USA, June 1992. Morgan Kaufmann.

[Márkus and Váncza, 1994] András Márkus and Jósef Váncza. Inference and optimization methods for manufacturing process planning. In Anthony G. Cohn, editor, *Proceedings of the 11th European Conference on Artificial Intelligence (ECAI-94)*, pages 595–599, Amsterdam, Netherlands, August 1994. Wiley.

[McAllester and Rosenblitt, 1991] David McAllester and David Rosenblitt. Systematic nonlinear planning. In AAAI-91 [1991], pages 634–639.

[McCarthy and Hayes, 1969] John McCarthy and Patrick J Hayes. Some philosophical problems from the standpoint of AI. *Machine Intelligence*, 4, 1969. Reprinted in [Webber and Nilsson, 1981], pages 431–450.

[Mehlhorn, 1984] Kurt Mehlhorn. *Graph Algorithms and NP-Comleteness*, volume 2. Springer, Berlin, 1984.

[Paige and Henglein, 1987] Robert Paige and Fritz Henglein. Mechanical translation of set theoretic problem specifications into efficient RAM code—A case study. *Journal of Symbolic Computation*, 4:207–232, 1987.

[Reiter and Mylopoulos, 1991] Ray Reiter and John Mylopoulos, editors. *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, Sydney, Australia, August 1991. Morgan Kaufmann.

[Sacerdoti, 1974] Earl D Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135, 1974.

[Sacerdoti, 1975] Earl D Sacerdoti. The non-linear nature of plans. In *Proceedings of the 4th International Joint Conference on Artificial Intelligence (IJCAI-75)*, Tbilisi, USSR, September 1975. IJCAI, William Kaufmann.

[Sandewall, 1987] Erik Sandewall. The pipelining transformation on plans for manufacturing cells with robots. In John McDermott, editor, *Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI-87)*, Milano, Italy, August 1987. Morgan Kaufmann.

[Schwartz and Sharir, 1990] Jacob T. Schwartz and Micha Sharir. Algorithmic motion planning in robotics. In van Leeuwen [1990], chapter 8, pages 391–430.

[Smith and Peot, 1993] David E. Smith and Mark A. Peot. Postponing threats in partial-order planning. In AAAI-93 [1993], pages 500–506.

[Strömberg, 1991] Jan-Erik Strömberg. Styrning av LEGO-bilfabrik. andra omarbetade upplagan. Department of Electrical Engineering, Linköping University, 1991.

[Tarski, 1955] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[Tate, 1977] Austin Tate. Generating project networks. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI-77)*, pages 888–893, Cambridge, MA, USA, August 1977. Reprinted in Allen *et al* [1990], pages 291–296.

[Todd, 1962] J. Todd. *Survey of Numerical Analysis.* McGraw Hill, New York, 1962.

[van Leeuwen, 1990] Jan van Leeuwen, editor. *Handbook of Theoretical Computer Science: Algorithms and Complexity*, volume A. Elsevier, Amsterdam, 1990.

[Webber and Nilsson, 1981] Bonnie Lynn Webber and Nils J Nilsson, editors. *Readings in Artificial Intelligence.* Morgan Kaufmann, 1981.

[Wilkins, 1988] David E Wilkins. *Practical Planning.* Morgan Kaufmann, San Mateo, CA, 1988.

[Yang and Tenenberg, 1990] Qiang Yang and Josh D Tenenberg. ABTWEAK: Abstracting a nonlinear, least commitment planner. In AAAI-90 [1990], pages 204–209.

[Yang, 1990] Qiang Yang. Formalizing planning knowledge for hierarchical planning. *Computational Intelligence*, 6(1):12–24, 1990.