State-Variable Planning under Structural Restrictions: Algorithms and Complexity

Peter Jonsson and Christer Bäckström
Department of Computer and Information Science
Linköpings Universitet
581 83 Linköping, Sweden
email: {petej,chrba}@ida.liu.se

Abstract

Computationally tractable planning problems reported in the literature so far 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. The exponential size of this graph, though, makes such restrictions costly to test. Hence, we propose an intermediate approach, using a state variable model for planning and defining restrictions on the separate state-transition graphs for each state variable. We identify such restrictions which can tractably be tested and we present a planning algorithm which is correct and runs in polynomial time under these restrictions. The algorithm has been implemented an it outperforms Graphplan on a number of test instances. In addition, we present an exhaustive map of 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 optimal and non-optimal plan generation problem.

1 Introduction

In this introductory section, we briefly motivate the work, describe our approach and present a characterization of the type of application problems we primarily aim at. The section concludes with an overview of the article.

1.1 Motivations and the Proposed Approach

Computational tractability is a fundamental issue in all problem solving. If a problem is not tractable, we cannot hope to solve an arbitrary instance of it within reasonable time. This is obviously important if our goal is to use computers for solving planning problems. Unfortunately, planning is known to be very hard. Even simple propositional planning is PSPACE-complete [9], which means that it is most likely intractable. Planning with variables ranging over an infinite domain is even undecidable [10, 13]. Until a few years ago the planning community did not show much interest in the formal complexity analysis of planning.

Many planning problems in the manufacturing and process industry are believed to be highly structured, thus allowing for efficient planning if exploiting this structure. 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 you have such a deep understanding of the problem structure, it is better to use other methods than heuristics.

Some tractability results for planning have recently been reported in the literature [5, 6, 9]. However, apart from being very restricted, they are all based on essentially syntactic restrictions on the set of operators. It is appealing to study syntactic restrictions, since they are often easy to define and not very costly to test. To gain any deeper insight into what makes planning problems hard and easy, respectively, we must probably study the structure of the problem, in particular the state-transition graph induced by the operators. To some extent, syntactic restrictions allow us to do this since they undoubtedly have implications for the form this graph takes. However, their value for this purpose seems somewhat limited since many properties that are easy to express as explicit structural restrictions would require horrendous syntactical equivalents. Putting explicit restrictions on the state-transition graph must be performed with great care since this graph is typically exponential in the size of the planning problem instance, making it extremely costly to test arbitrary properties.

In this article, we take an intermediate approach. We adopt the state-variable model SAS$^+$ [6] and define restrictions not on the whole state-transition graph, but on the domain-transition graphs for each state variable in isolation, taking some interaction between the variables into account. This is less costly since each such graph is only of polynomial size. Although not being a substitute for restrictions on the whole state-transition graph, many interesting and useful properties of this graph can be indirectly exploited. In particular, we identify sets of structural restrictions which make planning tractable and which properly generalize certain previously identified tractable SAS^+ problems [5, 6]. Despite being structural, our restrictions can be tested in polynomial time. For planning under these restrictions, we present a polynomial-time, provably correct planner. An empirical study of the algorithm reveals that it is considerably faster than Graphplan on certain test examples.

We also provide a map of the compuational complexity of both optimal and non-optimal plan generation for all combinations of the restrictions, considering also mixed syntactical and structural restrictions. We hope that by including negative results, too, unnecessary work can be avoided in the future.

1.2 Intended Applications

The kind of applications we have had in mind are problems where no human expert has a good understanding of how to plan. Such applications frequently arise in various engineering applications, typically having a large number of simple, specialized operators and being so extensive as to cause computational problems. Many applications of this type arise in the field of *sequential control*, a sub-area of *automatic control* dealing with discrete aspects of control applications. Although most industrial processes are continuous systems, they almost always also contain a superior discrete level. On this superior level the actions can often be viewed as discrete actions even if implemented as continuous processes. Many problems on this level can be viewed as planning. Typical actions can be to open or close a valve or to start or stop a motor.

An interesting application for automated planning is the situation where a process breaks down or is stopped in an emergency situation. At such an event the system may end up in any of a very large number of states and it might require a quite complex plan[1] to bring the system back into the normal operation mode. It is not realistic to have pre-compiled plans for how to start up the process again from each such state. Furthermore, the considerable costs involved when a large industrial process, such as a paper mill, is inactive necessitate a prompt response from the planner. There is typically no human expert who knows how to solve the problem or, even less, write down rules for this. Typically, the (human) operators have some partial understanding and experience which they combine with experimentation and improvisation—eventually getting the system running again (in most cases). Hence, planning from first principles seems a better approach than knowledge-based planning for this type of application, even if difficult. In principle, nothing prevents

[1] Such a plan may also contain parallel structures, of course. The word *sequential* in sequential control should not be interpreted literally.

us from also incorporating expert knowledge to the extent it is available. For further reading about intended applications and motivation, see Klein [24] and Bäckström [4].

1.3 About this Article

This article is compiled from two conference papers [20, 22] and the underlying technical reports [21, 23, 19], which contain the full proofs.

The remainder of this article is organized in the following way. Section 2 defines the SAS$^+$ planning formalism and discusses the main differences between propositional STRIPS and SAS$^+$. Section 3 defines the restrictions that will be considered in this article. Four previously identified syntactical restrictions (P, U, B, S) [6] are presented together with five new structural restrictions (I, A$^-$, A, A$^+$, O). Section 4 presents an algorithm for planning under restrictions I, A and O, together with a small example for illustrating how the algorithm works. The empirical evaluation of the algorithm can be found in Section 5.. Section 6 contains the correctness proofs and a complexity analysis for the planning algorithm. This section can be entirely omitted at a first reading of this article. Section 7 provides an exhaustive map of complexity results for planning under all nine restrictions from Section 3. This map considers both optimal and non-optimal plan generation. Some concluding remarks are presented in Section 8.

2 Basic Formalism

The SAS$^+$ formalism is in principle a version of the propositional STRIPS formalism [14]. This section briefly recasts the main differences between the two formalisms and provides a formal definition of the SAS$^+$ formalism. The reader is referred to previous publications [5, 1, 24, 6] for further background, motivation and examples.

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 for operators we also use a *prevail-condition*, which is a special type of precondition.

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 traditional STRIPS modelling is that the traditional pre-condition is split into two conditions, the pre-condition and the prevail-condition depending on whether the variables are changed by the operator or not. The 'behaviour' of an operator is thus 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 successful execution of the operator. 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.

2.1 Problem Instances and Plans

Definition 2.1 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 $\langle \mathsf{pre}, \mathsf{post}, \mathsf{prv} \rangle$, where $\mathsf{pre}, \mathsf{post}, \mathsf{prv} \in \mathcal{S}^+$ denote the *pre-*, *post-* and *prevail-condition*, respectively. If $o = \langle \mathsf{pre}, \mathsf{post}, \mathsf{prv} \rangle$ is a SAS$^+$ operator, we write $\mathsf{pre}(o)$, $\mathsf{post}(o)$ and $\mathsf{prv}(o)$ to denote pre, post and prv, respectively. $\mathcal{O}$ is subject to the following restriction: 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 pre-, post- and prevail-conditions are elements of $\mathcal{S}^+$, which means that they can take on the undefined value for some or all variables. As will be seen further on, this is interpreted as a non-condition—for example, if $\mathsf{prv}(o)[v] = \mathsf{u}$ then there is no prevail-condition on the variable v.

A SAS$^+$ instance implicitly defines a set of operator sequences which we refer to as *plans*.

Definition 2.2 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*.

Note that "operator sequence" and "plan" both denote any sequence of operators, not taking into account whether preconditions are satisfied or not.

Besides the concatenation operator ; we also need to refer to segments of operator sequences.

Definition 2.3 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 $First(\omega) = Last(\omega) = \langle\rangle = Rest(\omega) = \langle\rangle$.

2.2 Results and Validity

We continue by defining the result of applying an operator sequence to a state and defining when an operator sequence solves a SAS$^+$ problem. To begin with, we need an ordering on $\mathcal{S}^+$ that captures the idea of one state being defined in more detail than another state. This ordering will be used for determining when the pre- and prevail-conditions are satisfied for an operator.

Definition 2.4 Let $s \sqsubseteq t$ hold iff the value s is subsumed (or satisfied) by value t, *i.e.* 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 $\langle \mathcal{S}^+, \sqsubseteq \rangle$.

Note that while there is a bottom element in $\langle \mathcal{S}^+, \sqsubseteq \rangle$, there is (usually) no top element. Thus, $s \sqcup t$ is not defined if there is variable v such that $s[v] \neq \mathsf{u}$, $t[v] \neq \mathsf{u}$ and $s[v] \neq t[v]$—there is no value x such that $s[v] \sqsubseteq x$ and $t[v] \sqsubseteq x$. However, we will only use $\sqcup$ when it is defined. To define the result of a plan when applied to a state, we first need to know what the result is of applying a single operator to a state.

Definition 2.5 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 [14]), 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.

We can now state when an operator sequence solves a problem instance.

Definition 2.6 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 $\mathit{Valid}(\langle o_1, \ldots, o_n \rangle, s_0, s_*)$.

2.3 Partially Ordered Plans

Up till now, we have only considered totally ordered plans—sequences of operators. However, planning often only requires a partial ordering of the operators; some operators must be executed in a certain order while others can be executed in arbitrary order.

To define partially ordered plans, we must introduce the concept of *actions*, *i.e.* 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 2.7 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 $TS(\Delta)$. For partial orders, we need the concepts of transitive closure and reduction.

Definition 2.8 Given a strict 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 2.9 Let $\mathcal{O}$ be a set of operators over some set $\mathcal{V}$ of state variables and let $v \in \mathcal{V}$. The function $\mathit{Affects} : \mathcal{V} \times \mathcal{O} \to 2^{\mathcal{O}}$ 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 2.10 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 $\mathit{Paths} : \mathcal{V} \times (2^{\mathcal{A}} \times (2^{\mathcal{A} \times \mathcal{A}})) \to 2^{\mathit{Seqs}(\mathcal{O})}$ is defined as:

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

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 $\mathit{Paths}(v, \langle \mathcal{A}, \prec \rangle)$ may contain more than one sequence. 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

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

2.4 A Remark on the Undefined Value

As previously seen, the undefined value u is used in several ways in the SAS$^+$ formalism:

- In a pre- or prevail-condition, u indicates that there is no such condition for that variable; we do not care about the value at all.
- In a post-condition, u indicates that the operator does not change the value of that variable.
- In an initial state, u indicates that we do not know the value of the variable.
- In a goal state, u indicates that we do not care what the value of the variable is when the plan has been executed.

The multiple roles of u stems from the *action structures* formalism [31] from which SAS$^+$ evolved. Since there is a potential risk of confusion, the reader ought to keep this multiple use of u in mind when the symbol is encountered.

3 Restrictions

The section defines the restrictions to be considered in this article. We present five structural restrictions (I, A$^-$, A, A$^+$ and O) as well as the four previously studied 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 [5, 6]. 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 An operator o is *unary* iff there is exactly one variable v such that $\mathsf{post}(o)[v] \neq \mathsf{u}$.

Definition 3.2 A SAS$^+$ instance $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ is

(P) *Post-unique* iff for all $o, o' \in \mathcal{O}$, whenever $\mathsf{post}(o)[v] = \mathsf{post}(o')[v] \neq \mathsf{u}$ 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 [1, p. 84].

3.2 Prerequisites for Structural Restrictions

Next we will define the new structural restrictions I, A$^-$, A, A$^+$ and O. However, we first need to define some underlying concepts. We will use the SAS$^+$ problem instance Σ defined in Example 3.1 below as an example for demonstrating the various concepts.

Example 3.1 $\Sigma = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ where $\mathcal{V} = \{v_1, v_2\}$ and $\mathcal{D}_{v_1}, \mathcal{D}_{v_2}$ and $\mathcal{O}$ are defined in Tables 1 and 2. For our purposes the exact values of s_0 and s_* do not matter so we leave them unspecified.

In the forthcoming definitions, let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ be an arbitrary SAS$^+$ instance.

For each state variable domain, we define the graph of possible transitions for this domain, the domain-transition graph, without taking the other domains into account. We also define the implied reachability graph that shows whether there is a path or not between the various domain values.

Definition 3.3 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.

Note that the condition "$\mathsf{pre}(o)[v] \sqsubseteq x$" is equivalent to "$\mathsf{pre}(o)[v] = \mathsf{u}$ or $\mathsf{pre}(o)[v] = x$". Hence, there will be an arc from x to y when there is an operator with x as pre-condition and y as post-condition; there will also be an arc from x to y when there is an operator with u as pre-condition and y as post-condition. The latter case will introduce $|\mathcal{D}_v^+|$ arcs, one from each vertex, into the domain-transition graph, including one from the vertex u, which is consistent with the use of u as a "don't know" value—we do not have a pre-condition on v, so the operator can be applied even if we do not know the value of v.

As an example, the domain-transition graphs of Σ are provided in Figure 1. Another way of viewing G_v^X is as the restriction to $X \subseteq \mathcal{D}_v^+$ of the

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

Table 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 2: Operators of Σ.

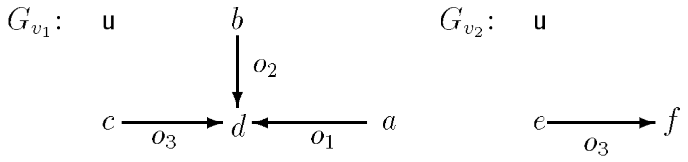


Figure 1: Domain-transition graphs of Σ.

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, *i.e.* 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. Naturally, other vertices (*i.e.* vertices not in X) may be visited along the path.
We continue by defining two distinguished types of values, the requestable values. The set of requestable values plays an important role for defining the structural restrictions. They will also be extensively used in the definition of the planning algorithm.

Definition 3.4 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

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

and the set $\mathcal{R}_v^{\mathcal{O}'}$ of *requestable values* for $\mathcal{O}'$ is defined as

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

Example 3.2 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 $\mathcal{R}_{v_2}^{\mathcal{O}} = \{e, f\}$.

To illustrate the ideas behind the requestable values, suppose that we create a path α from x to y in G_v. The operators in that path may have prevail-conditions on another variable v'. If the operators are non-unary, they may also have pre- and post-conditions on v'. The path "requests" v' to obtain those values in a certain order. Thus, a plan for an instance with $s_0[v] = x$ and $s_*[v] = y$ containing the path α *must* include a path from $s_0[v']$ to $s_*[v']$ *via* those requested values.

3.3 Structural Restrictions

We can now define the structural restrictions.

Definition 3.5 Let $G = \langle V, A\rangle$ be a directed graph and let G' be its associated undirected graph, *i.e.* the symmetric closure of G. The set of *components* of G is the set of largest subgraphs of G such that the associated subgraphs in G' are connected.

Definition 3.6 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 3.3 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.

Note that an operator o with $\mathsf{pre}(o)[v] = \mathsf{u}$ and $\mathsf{post}(o)[v] = x \neq \mathsf{u}$ introduces several arcs into the domain transition graph; specifically, there will be a reflexive arc from x to x since $\mathsf{pre}(o)[v] = \mathsf{u} \sqsubseteq x$. Removing that arc will not split the domain-transition graph into components so operators of this type cannot be irreplaceable.

Definition 3.7 Let $\mathcal{O}$ be a set of operators and let $\omega, \omega' \in \mathit{Seqs}(\mathcal{O})$ 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 $1 \leq j \leq k$.

Intuitively, $\omega \trianglelefteq \omega'$ means that ω' contains operators that require at least the same prevail-conditions as the operators in ω and, furthermore, they are required in the same order. Note that $\langle\rangle \trianglelefteq \omega$ for any $\omega \in \mathit{Seqs}(\mathcal{O})$. Furthermore, if $\omega, \omega', \psi, \psi' \in \mathit{Seqs}(\mathcal{O})$, $\omega \trianglelefteq \omega'$ and $\psi \trianglelefteq \psi'$, then $(\omega; \psi) \trianglelefteq (\omega'; \psi')$.

We can now define the structural restrictions.

Definition 3.8 A 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 $Seqs(\mathcal{O})$, whenever ω is a shortest path from x to y via X and ω' is a path from x to y via X, then $\omega \trianglelefteq \omega'$.

The formal definitions may, perhaps, seem somewhat difficult to understand at a first reading, so the reader might find the following intuitive explanations helpful.

Interference-safeness: If an operator changes more than one variable, then for each such variable it splits the domain of this variable into two partitions and is the only operator which can move between these partitions. This means that if we have to move from one partition to the other for a variable, then we know that we must also do so for all other variables the operator affects, *i.e.* interference between subplans arises. If this side-effect is unacceptable, then we know immediately that there is no solution. Otherwise, this helps by splitting the problem into smaller subproblems.

Acyclicity: For each variable, certain distinguished values must be achieved in a particular order, if they need to be achieved at all. Since these are precisely those values that may cause interference between subplans for different variables, they can be used as synchronization points in the planning process. The variants A^+ and A^- consider fewer and more distinguished values respectively.

Prevail-order-preservation: For each state variable, the sequence of prevail-conditions required along a shortest operator sequence between two values must be the same as or a relaxation of the values requested by any other sequence between the same values. Hence, it can never be easier to generate a plan by generating non-shortest subplans for a variable since this would never cause less interference between the subplans.

Example 3.4 Consider the previous Σ example. By example 3.3, it is easy to see that Σ is interference-safe since o_3 is the only non-unary action and it is irreplaceable wrt. 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.

3.4 Testing Structural Restrictions

Given a SAS$^+$ instance Π, 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 graph 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 [32]. Hence, checking restriction I takes polynomial time. The complexity of checking prevail-order-preservation remains an open question. Fortunately, it is tractable to test this restriction for instances that are acyclic wrt. requestable values, so it is tractable to test the combination AO. First, we define the notion of *weak* prevail-order-preservation. Weak prevail-order-preservation is a version prevail-order-preservation where the sets of intermediate values are required to be empty.

Definition 3.9 A domain-transition graph G_v is *weakly prevail-order-preserving* iff for all $x, y \in \mathcal{D}_v^+$ and for all $\omega = \langle o_1, \ldots, o_m \rangle, \omega' = \langle o'_1, \ldots, o'_n \rangle \in Seqs(\mathcal{O})$, 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 3.10 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 sketch: [2] The only-if part is immediate from the special case when the paths go via the empty set. The if part is a straightforward induction over the size of the subsets of $G_v^{\mathcal{R}_v^{\mathcal{O}}}$. □

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

Definition 3.11 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 3.12 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.

[2]Full proofs can be found in the technical reports listed in Section 1

Lemma 3.13 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, both $\omega \trianglelefteq \omega'$ and $\omega' \trianglelefteq \omega$ hold so $\omega \triangleq \omega'$. $\Box$

We now present an algorithm, *Check-SPC*, for checking shortest-path consistency. The algorithm can be found in Figure 2. Note that the **for**-loop in line 3 is not technically necessary for the algorithm to function properly, it is merely inserted to allow for a simpler correctness proof.

Lemma 3.14 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 correspond 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 $\omega = \langle o \rangle$ and $\psi = \langle p \rangle$. Since the algorithm did not fail in line 2, $\mathsf{prv}(o)[v] = \mathsf{prv}(o')[v]$ and $\omega \triangleq \psi$ follow.
Induction hypothesis: For arbitrary $x \in \mathcal{D}_v^+$, $y \in \mathcal{D}_v$ and for all pairs of shortest paths ω, ψ from x to y of length $n \leq k$ 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 into two non-empty paths, ω_1, ω_2 such that $\omega = (\omega_1; \omega_2)$ and $\mathsf{post}(Last(\omega_1))[v] = r$. Since ω is a shortest path from x to y, it follows that 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 achieve

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

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

Based on *Check-SPC*, we can similarly devise an algorithm for checking weak prevail-order-preservation. The algorithm, *Check-WO*, can be found in Figure 3.

Lemma 3.15 Let v be a shortest-path-consistent state variable. Then, *Check-WO* accepts iff G_v is weakly prevail-order-preserving.

Proof: Analogous to the proof of Lemma 3.14. □

Consider algorithm *Check-O* in Figure 3.4. The next theorem shows that it accepts if and only if the given SAS$^+$ instance (which must satisfy restriction A) is prevail-order-preserving.

Theorem 3.16 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 3.13. Consequently, *Check-O* cannot fail in line 3. Furthermore, v is weakly prevail-order-preserving by Theorem 3.10 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 3.14. 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 3.15. Furthermore, we know that $G_v^{\mathcal{R}_v^{\mathcal{O}}}$ is acyclic. Hence, by Theorem 3.10, 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 [12]. (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 leave the following theorem without proof.

procedure *Check-SPC*(v)
if there exist two operators $o, o' \in \mathcal{O}$ such that all 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;
for $i \leftarrow 2$ **to** $|\mathcal{D}_v^+|$ **do**
  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**
    let ω be any such path
    for $z \in \mathcal{D}_v - \{x, y\}$ **do**
      $\gamma_1 \leftarrow$ a shortest path from x to z in G_v;
      $\gamma_2 \leftarrow$ a shortest path from z to y in G_v;
      if $|(\gamma_1; \gamma_2)| = |\omega|$ and not $\omega \triangleq (\gamma_1; \gamma_2)$ **then reject**;
accept;

Figure 2: Algorithm for testing shortest-path consistency.

procedure *Check-WO*(v)
for $i \leftarrow 2$ **to** $|\mathcal{D}_v^+|$ **do**
  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**
    let ω be any such path
    for $z \in \mathcal{D}_v - \{x, y\}$ **do**
      $\gamma_1 \leftarrow$ a shortest path from x to z in G_v;
      $\gamma_2 \leftarrow$ a shortest path from z to y in G_v;
      if $\omega \not\trianglelefteq (\gamma_1; \gamma_2)$ **then reject**;
accept;

Figure 3: Algorithm for testing weak prevail-order-preservation

procedure *Check-O*(Π)
comment $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ *satisfies restriction A.*
for $v \in \mathcal{V}$ **do**
  Check-SPC(v);
  Check-WO(v);
accept;

Figure 4: Algorithm for testing prevail-order-preservation.

Theorem 3.17 *Check-O* has a worst-case time complexity of $O(|\mathcal{V}|^2|\mathcal{O}|^2M^5)$ where $M = \max_{v\in\mathcal{V}} |\mathcal{D}_v^+|$.

The complexity figure given in Theorem 3.17 is a coarse estimate. By a more careful analysis and by using improved algorithms, this figure could probably be lowered substantially.

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 then illustrated with a small workshop example. The 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 [6].

4.1 The Algorithm and its Complexity

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

Lemma 4.1 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 $Paths(v, \langle \mathcal{B}, \prec_\Theta \rangle)$, the operator sequence $type(\alpha)$ 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 $type(\alpha)$ is a path in G_v from $s_0[v]$ to $s_*[v]$ (A proof for this can be found in [24, 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 lemma 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 arises and the lemma follows.

□

This is part of 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 5 and 6) 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 values 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, consequently, *Plan* fails. 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 for v_i in the next iteration. Similarly, each non-unary operator o 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 $\mathit{Extend} : 2^{\mathcal{D}_{v_1}} \times \ldots \times 2^{\mathcal{D}_{v_m}} \rightarrow 2^{\mathcal{D}_{v_1}} \times \ldots \times 2^{\mathcal{D}_{v_m}}$, *i.e.* ignoring the side-effect on $\omega_1, \ldots, \omega_m$, this process corresponds to constructing a fixpoint for *Extend* in $2^{\mathcal{D}_{v_1}} \times \ldots \times 2^{\mathcal{D}_{v_m}}$. 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 in such a way 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 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. It turns out that there always is 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 prevail-condition. Finally, if the transitive closure $\prec^+$ of $\prec$ is irreflexive, then $\langle \mathcal{A}, \prec \rangle$ is returned and otherwise *Plan* fails.

We continue by stating the worst-case time complexity of the *Plan* algorithm using straightforward, non-optimal assumptions about data structures.

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
  $\langle X_1, \ldots, X_m \rangle \leftarrow$ *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;
$\mathcal{A} \leftarrow \{a \in \alpha_i \mid 1 \leq i \leq m\}$;
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 $a \prec a_1$;
if $\prec$ is irreflexive **then return** $\langle \mathcal{A}, \prec \rangle$;
else fail;

function *Extend* ($\langle X_1, \ldots, X_m \rangle$)
comment *The variables* $\omega_1, \ldots, \omega_m$ *are implicitly used as output parameters.*
for $1 \leq i \leq m$ **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 $1 \leq i \leq m$ **do**
  for $1 \leq j \leq m$ and $o \in \omega_j$ **do**
    $X_i \leftarrow X_i \cup \{\mathsf{prv}(o)[v_i]\} - \{\mathsf{u}\}$;
for $1 \leq i \leq m$ **do**
  for $1 \leq j \leq m$ 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 $\langle X_1, \ldots, X_m \rangle$;

Figure 5: Planning algorithm and subroutine Extend.

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 $X \neq \emptyset$ **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. Dijkstra's algorithm.*
comment *Assume* $\overline{X} = \langle \overline{X}_1, \ldots, \overline{X}_n \rangle$.
for $i \leftarrow 1$ **to** $n - 1$ **do**
  $\omega \leftarrow (\omega; SP(\overline{X}_i, \overline{X}_{i+1}))$;
return ω;

procedure *Instantiate*
comment *Modifies* $\alpha_1, \ldots, \alpha_m$
for $1 \leq i \leq m$ **do**
  Assume $\omega_i = \langle o_1, \ldots, o_k \rangle$
  for $1 \leq l \leq k$ **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 6: Subroutines Shortest-Path and Instantiate.

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. Remember that we can find a shortest path between two values in a domain-transition graph G_v in $O(|\mathcal{D}_v^+||\mathcal{O}|)$ time, 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.

Under these assumptions, we can prove the complexity of *Plan*:

Theorem 4.2 The *Plan* algorithm has a worst-case time complexity of $O(|\mathcal{V}|^3|\mathcal{O}|^2M^5)$ where $M = \max_{v\in\mathcal{V}} |\mathcal{D}_v^+|$.

Proof sketch: It is straightforward to show that *Extend* and *Instantiate* have worst-case complexities of $O(|\mathcal{O}|^2|\mathcal{V}|^2M^4)$ and $O(|\mathcal{V}|^2|\mathcal{O}|^2)$ respectively. Deciding whether a directed graph $\langle V, A\rangle$ is irreflexive or not can be done in $O(|V|+|E|)$ time [28, p. 4-7]. Hence, the irreflexivity test can be carried out in $O(|\mathcal{O}|^2)$ time since $\mathcal{A}$ can contain two actions of each type at most. The **repeat** loop in lines 3-5 of *Plan* can call *Extend* maximally $\sum_{v\in\mathcal{V}} |\mathcal{D}_v^+| \leq |\mathcal{V}|M$ times. Hence, lines 3-5 run in $O(|\mathcal{V}|^3|\mathcal{O}|^2M^5)$ which dominates the running time of the algorithm. □

The complexity of *Plan* can probably be improved by employing a more careful analysis and using better data structures.

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 on. There are also two workstations: a lathe and a drill. To simplify matters, we will consider only one single workpiece. (It has been suggested by Sandewall [30] 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.) 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 (*i.e.* not worked on), shape 1 or shape 2 without a hole and shape 2 with a hole. Note also that operator Shape2 is tougher on the cutting tool than Shape1 is—only the former allows 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 3, and nine operators, as shown in Table 4. This example is a SAS$^+$-IAO instance, but it does not satisfy either of the P, U and S restrictions in Bäckström and Nebel [6].

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

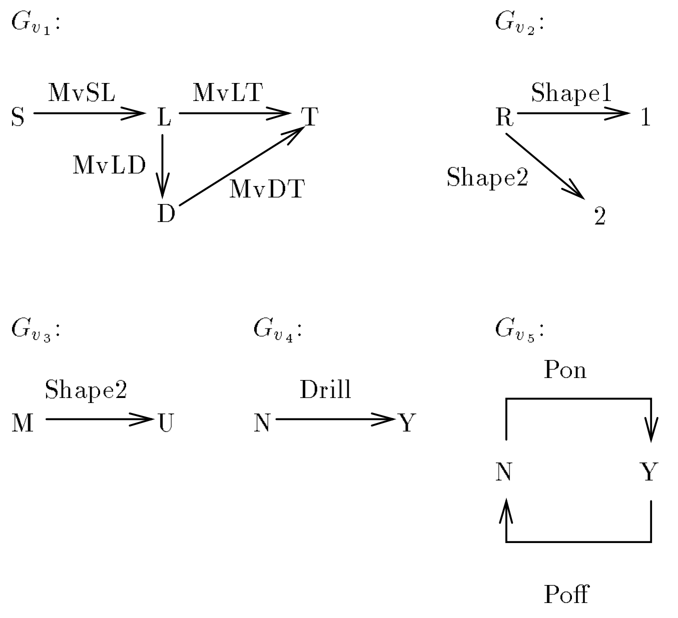


Figure 7: Domain-transition graphs for the workshop example. The undefined value is excluded.

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.

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

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

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

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 8.

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

The maximally tractable problem under the previous, syntactical restrictions is the SAS$^+$-PUS problem [6]. Before proceeding with the correctness of algorithm *Plan*, we show that the SAS$^+$-IAO problem is strictly more general than the SAS$^+$-PUS problem [2, 6]. 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.

Lemma 4.3 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. $\square$

Lemma 4.4 If Π is a unary SAS$^+$instance, then Π is interference-safe.

Proof: Immediate from the definition of interference-safeness. $\square$

Lemma 4.5 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 and 3.10, 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:

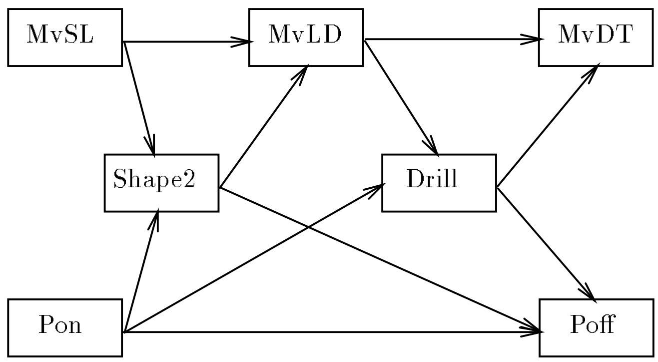


Figure 8: The final partial-order plan (only $\prec$, not $\prec^+$, is shown).

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. □

Theorem 4.6 Every SAS$^+$-PUS instance is a SAS$^+$-IAO instance.

Proof: Immediate from Lemmata 4.3, 4.4 and 4.5. □

Theorem 4.7 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 $\mathsf{prv}(o_2)[v_2] = f$. □

Corollary 4.8 The SAS$^+$IAO class is strictly more general than the SAS$^+$-PUS class.

5 Empirical Results

As a complement to the complexity analysis of algorithm *Plan*, we have also performed an empirical study. The algorithm was implemented in C++ with the aid of the LEDA data type and algorithm library[3]. Details concerning the implementation, known as Sasplan, can be found in Kvarnström [26]. In our experiment, we compared Sasplan with Graphplan [8] which is recognized as one of the fastest propositional planners available. It must be clearly understood that any comparison between these two planners is by necessity unfair: Graphplan is a general-purpose propositional planner which is applicable to a wider range of problems than Sasplan.

We compared the planners on three domains and the detailed results can be found below. The experiments were performed on a SUN Sparcstation 10[4]. If a planner failed to solve a given instance within approximately 90 seconds, the experiment was aborted. Such aborted experiments are marked

[3]Information about LEDA can be found at `http://mpi-sb.mpg.de/LEDA`. Version 3.4 was used in our experiments.

[4]SUN and Sparcstation 10 are trademarks of SUN Microsystems.

n	Graphplan	Sasplan
20	0.8	0.4
40	5.0	1.2
60	15.6	2.7
80	38.8	5.2
90	75.0	6.8
100	*	8.8
150	*	25.0
200	*	54.5

Table 5: Empirical results of applying algorithm *Plan* on the D^1S^1 domain. The running time (in seconds) is the mean value of five sample runs.

with an asterisk in the tables containing the results. It should be noted that we have required that Sasplan test the restrictions I, A and O in each and every experiment. Since Sasplan must perform these checks in a real-world planning situation, it is reasonable that the tests are performed also in the experimental setting.

5.0.1 The D^1S^1 Domain

The D^1S^1 domain was invented by Barrett and Weld [7]. A D^1S^1 domain of size n, $n \geq 2$, consists of $2n$ propositional atoms $I_1, \ldots, I_n$ and $G_1, \ldots, G_n$ together with n operators $A_1, \ldots, A_n$. In STRIPS-style notation, the operators are defined as follows:

$$(\texttt{:action}\ A_1\ \texttt{:precond}\ \{I_1\}\ \texttt{:add}\ \{G_1\});$$

$$(\texttt{:action}\ A_i\ \texttt{:precond}\ \{I_i\}\ \texttt{:add}\ \{G_i\}\ \texttt{:delete}\ \{I_{i-1}\}),\ i \geq 2.$$

The initial state is $\{I_1, \ldots, I_n\}$ and the goal state $\{G_1, \ldots, G_n\}$. It is easy to see that the plan $A_1, A_2, \ldots, A_n$ is a solution to the instance. A D^1S^1 domain can trivially be converted to an equivalent SAS$^+$ instance by replacing each propositional atom with a state variable with domain $\{0, 1\}$ and modifiying the operators accordingly. It is easily verifiable that the resulting SAS$^+$ instance satisfies restricitons I, A and O.

Blum and Furst [8] showed that Graphplan is competitive with the best performance reported on the D^1S^1 domain [7]. In Table 5, we can see that Sasplan outperforms Graphplan on this domain.

5.0.2 The Tunnel Example

The tunnel example is a toy domain that has been used in control theory. It assumes a tunnel (see Figure 9) divided into n sections such that the light can be switched on and off independently in these. The only light switches for a section are located at the two ends of that section. It is also assumed that one can only pass through a section if the light is on in that section. As a typical instance of this problem assume that all lights are off and the task is to turn the light on in the innermost section while not turning on any other light. This can be achieved by going into the tunnel, repeatedly switching on the light in each new section encountered until reaching the innermost section. Then leave the tunnel again, repeatedly switching off the light in each section, except the innermost one, when leaving it.

Modelling this problem as a SAS$^+$ instance is straightforward and we follow the model suggested by Klein [24]. Define n state variables $v_1, \ldots, v_n$ such that

$$v_i = \begin{cases} 0 & \text{when the light in section } i \text{ is off} \\ 1 & \text{when the light in section } i \text{ is on} \end{cases}$$

for $i = 1, \ldots, n$. For each state variable v_i we define two actions On_i and Off_i with the obvious meanings: On_i turns the light on in section i and Off_i turns it off. The operator On_i has the following definition:

$$\begin{aligned}
\mathsf{pre}(On_i)[v_k] &= \begin{cases} 0 & \text{if } k = i \\ \mathsf{u} & \text{otherwise} \end{cases} \\
\mathsf{post}(On_i)[v_k] &= \begin{cases} 1 & \text{if } k = i \\ \mathsf{u} & \text{otherwise} \end{cases} \\
\mathsf{prv}(On_i)[v_k] &= \begin{cases} 1 & \text{if } 1 \leq k \leq i-1 \\ \mathsf{u} & \text{otherwise} \end{cases}
\end{aligned}$$

The Off_i operator is defined analogously. Finally, we have the following initial and goal states:

- $s_0[v_i] = 0$ for $1 \leq i \leq n$;
- $s_*[v_n] = 1$ and $s_*[v_i] = 0$ for $1 \leq i \leq n-1$.

Unfortunately, this problem cannot be directly modelled as a STRIPS instance suitable for Graphplan. The reason for this is that Graphplan does not support negative goals. However, we can use a transformation proposed by Bäckström [3] to overcome this problem. The trick is to represent each state variable v_i with two propositions v_i-0 and v_1-1. The interpretations of these new propositions is that $v_i = 1$ iff v_i-1 is true and v_1-0 is false and analogously for $v_i = 0$. By using this representation, we get the following STRIPS definition of On_i:

$$(\texttt{:action}\, On_i \ \texttt{:precond}\ \{v_k\text{-}1 \mid 1 \leq k \leq i-1\}\ \texttt{:add}\ \{v_i\text{-}1\})\ \texttt{:del}\ \{v_i\text{-}0\}).$$

n	Graphplan	Sasplan
10	0.6	0.2
15	11.7	0.3
17	65.0	0.4
20	*	0.6
50	*	3.4
100	*	17.0
150	*	44.6

Table 6: Empirical results of applying algorithm *Plan* on the tunnel example. The running time is the mean value of five sample runs.

The initial state is then $\{v_k\text{-}0 \mid 1 \leq k \leq n\}$ and the goal state $\{v_n\text{-}1\} \cup \{v_k\text{-}0 \mid 1 \leq k \leq n-1\}$.

Since we were forced to use this transformation, the comparison between Graphplan and Sasplan in Table 6 is not completely fair; Graphplan must handle a problem containing twice as many propositions as the problem Sasplan considers. However, this discrepancy does not explain the remarkable difference in performance.

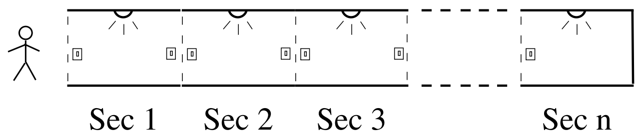


Figure 9: The tunnel example.

5.0.3 Random Instances

Our last experiment considers a certain type of random planning instances. Let $0 \leq \delta \leq 1$ and define the problem RAND(δ) of size n as an instance consisting of n propositional atoms $P_1, \ldots, P_n$ together with n operators $A_1, \ldots, A_n$. Define sets Ψ_i^δ such that they with probability δ contain atom P_k, $1 \leq k \leq i-1$ and define the operators $A_1, \ldots, A_n$ as follows:

$$(\texttt{:action } A_i \texttt{ :precond } \Psi_i^\delta \texttt{ :add } \{P_i\}).$$

The initial state is $\varnothing$ and the goal state $\{P_1, \ldots, P_n\}$. While our two other test domains have unique minimal solutions, this domain has (usually) many different minimal solution. It is straightforward to verify that, for instance, the plan $A_1, A_2, \ldots, A_n$ is a solution to the instance. However, it may or may not be a minimal solution to the instance depending on the sets $\Psi_1^\delta, \ldots, \Psi_n^\delta$.

	Graphplan		Sasplan	
n	$\delta = 0.2$	$\delta = 0.5$	$\delta = 0.2$	$\delta = 0.5$
50	0.5	1.6	0.9	1.1
100	2.9	12.7	4.1	4.9
150	8.2	59.7	11.0	13.1
200	22.6	*	24.1	27.5
250	71.9	*	43.0	49.6
300	*	*	71.1	81.3

Table 7: Empirical results of applying algorithm *Plan* on random instances. The running time is the mean value of ten sample runs. For each sample run, a new instance of RAND(δ) was generated.

Note that the breadth-first search strategy of Graphplan should be well suited for this kind of problems.

A RAND(δ) instance can trivially be converted to an equivalent SAS$^+$ instance by replacing each propositional atom with a state variable with domain $\{0, 1\}$ and modifiying the operators accordingly. The resulting SAS$^+$ instance satisfies restricitons I, A and O (in fact, it also satisfies P, U, B and S).

In the experiment we used two choices of δ. In the first experiment, where $\delta = 0.2$, Graphplan was actually faster than Sasplan on small instances. As the sizes increased, the gap was narrowed and, eventually, Sasplan became faster than Graphplan. By choosing $\delta = 0.5$, Sasplan were consistently faster than Graphplan.

6 Correctness of Planning Algorithm

This section contains all correctness and complexity results for algorithm *Plan* in the previous section. Let A be a planning algorithm. A is said to be *sound* iff whenever it generates a plan ω for a SAS$^+$ instance Π, ω is a solution to Π. A is *complete* iff for every solvable SAS$^+$ instance Π, A generates a plan when applied to Π.

We begin by stating 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.

6.1 Soundness for SAS$^+$-IA Instances

We first prove that algorithm *Plan* is sound for SAS$^+$-IA instances. 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 $Valid(\omega, s_0, s_*)$. We begin by showing correctness of *Shortest-Path*.

Lemma 6.1 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: Trivial. □

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

Lemma 6.2 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 $\mathsf{post}(o)[v] = y \neq \mathsf{u}$.

Lemma 6.3 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 partial-order plan solving Π. If $o \in \mathcal{O}$ is an irreplaceable operator, then there cannot exist more than one action of type o in $\mathcal{B}$.

We continue by defining *separability.*

Definition 6.4 Let Π be a SAS$^+$ instance. A partial-order plan $\Delta = \langle \mathcal{A}, \prec \rangle$ is *separable wrt.* Π iff for each topological sort ω 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 $Paths(v, \omega) = \{(\alpha; \beta)\}$.
- if there does not exist any $a \in \mathcal{A}$ such that $\mathsf{prv}(a)[v] \neq \mathsf{u}$ then $Paths(v, \omega)$ is a path in G_v from $s_0[v]$ to $s_*[v]$.

We can now show that if a partial-order plan is separable, then it solves Π.

Theorem 6.5 A partial-order plan $\Delta = \langle \mathcal{A}, \prec \rangle$ for a SAS$^+$ problem instance Π solves Π if Δ is separable wrt. Π.

Proof sketch: 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 partial-order plan, at least one such α exists. The proof consists of two parts: Showing that $Valid(\alpha, s_0, \perp)$ holds and showing that $s_* \sqsubseteq result(s_0, \alpha)$. Showing that $Valid(\alpha, s_0, \perp)$ is fairly straightforward by induction over all initial sequences of α. Proving that $s_* \sqsubseteq result(s_0, \alpha)$ is an easy analysis of how the actions in α are ordered. □

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 6.6 Δ is a partial-order plan.

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

Lemma 6.7 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 of *Extend* 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 of *Plan* 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 lines 11-16 in the algorithm. Finally, we know that $(\beta_i; \gamma_i) = \alpha_i = \delta_{v_i}$ and, consequently, Δ is separable. $\Box$

We can now show that *Plan* is sound for SAS$^+$-IA instances:

Theorem 6.8 The algorithm *Plan* 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 6.6, 6.7 and Theorem 6.5 $\Box$

6.2 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 6.21 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 6.22.

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 6.9 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 $Ker : 2^{\mathcal{O}} \to 2^{\mathcal{D}_{v_1}} \times \ldots \times 2^{\mathcal{D}_{v_m}}$ is defined as

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

where $\mathcal{O}' \subseteq \mathcal{O}$. Ker can be extended to operate on sets of actions in the natural way. 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 6.10 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 in G_v and let ψ be a shortest path from x to y via Y in G_v. Then the following holds:

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

Proof: The first part is an easy proof by contradiction. The second part follows trivially from the first. □

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

Definition 6.11 A partial-order plan $\Theta = \langle \mathcal{B}, \prec \rangle$ is $\mathcal{R}$-*unique* iff for every $v \in \mathcal{V}$, either

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 $\mathsf{post}(b)[v] = x$; or

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

Proof sketch: The proof boils down to showing that if two actions a and b generates the same requestable value, then there is an equivalent, but shorter, plan that only contains a. The transformation from the longer to the shorter plan turns out to be trivial in the presence of restriction A. It should be noted that this lemma holds for arbitrary SAS$^+$-A instances, not only SAS$^+$-IAO instances. $\square$

$\mathcal{R}$-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 6.13 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 $\theta_i = \langle \beta_0^i; \beta_1^i; \ldots; \beta_r^i \rangle$ 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 $\mathcal{R}_{v_i}^{\mathcal{B}}$. Since $R \subseteq \mathcal{R}_{v_i}^{\mathcal{B}}$, 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. $\square$

The previous lemma leads to the following definition.

Definition 6.14 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 $R \subseteq \mathcal{R}_{v_i}^{\mathcal{B}}$, $\theta_i // R$ denotes the unique subdivision $\langle \beta_0^i, \beta_1^i, \ldots, \beta_r^i \rangle$ 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 $z = s_0[v_i] \in R$ then $\mathsf{post}(Last(\beta_j^i))[v_i] \in R - \{z\}$ 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 6.15 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 $1 \le i \le m$. 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, 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 $s_*[v_i] = \mathsf{u}$ is similar.)

Then there exists some p, $0 \le p \le k$ 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}. $\square$

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

Theorem 6.16 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 6.15, 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 $1 \le j \le m$,

$$\widehat{\bigcup}_{o \in \omega_i} \{\mathsf{prv}(o)[v_j]\} = \widehat{\bigcup}_{o \in \theta_i} \{\mathsf{prv}(o)[v_j]\}. \tag{1}$$

By Lemma 6.10, $\mathcal{I}(\omega_j) = \mathcal{I}(\theta_j)$ and consequently

$$\widehat{\bigcup}_{o \in \mathcal{I}(\omega_i)} \{\mathsf{pre}(o)[v_j], \mathsf{post}(o)[v_j]\} = \widehat{\bigcup}_{o \in \mathcal{I}(\theta_i)} \{\mathsf{pre}(o)[v_j], \mathsf{post}(o)[v_j]\}. \tag{2}$$

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

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

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

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

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

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 6.17 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 6.18 Every SAS$^+$-O instance is prevail-monotonic.

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

Lemma 6.19 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 lines 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 lines 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 lines 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 6.10 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, $Extend(Ker(\Theta)) \leq_{Ker} Extend(Ker(\Theta'))$. □

The next theorem ties together Lemma 6.19 and Theorem 6.16 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 [29] which is derived from Tarski [33].

Definition 6.20 A partial-order plan Θ solving Π is *$\mathcal{R}$-minimal* iff for every minimal plan Θ' solving Π, $Ker(\Theta) \leq_{Ker} Ker(\Theta')$.

Theorem 6.21 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

6.19, $Extend(Extend^k(\langle\varnothing,\ldots,\varnothing\rangle)) \leq_{Ker} Extend(Ker(\Theta))$. As a consequence, $Extend(Ker(\Theta)) = Ker(\Theta)$ by Theorem 6.16 and it follows that $Extend^{k+1}(\langle\varnothing,\ldots,\varnothing\rangle) \leq_{Ker} Ker(\Theta)$. □

The minimality of Δ follows immediately.

Theorem 6.22 Δ is minimal.

Proof: Choose $v_i \in \mathcal{V}$ arbitrarily. By Lemma 6.21, for every minimal plan Θ solving Π, every path in $Paths(v_i, \Theta)$ must pass every state in $\mathcal{R}^{\mathcal{A}}_{v_i}$. Since Δ contains a shortest such path in G_{v_i} and v_i was chosen arbitrarily, minimality follows. □

6.3 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 6.32, we show that for every solution Θ for Π, there exists a structure-preserving function φ from Δ to Θ. Hence, if Δ is not irreflexive, then there cannot exist any plan solving Π. With this fresh in mind, we show that *Plan* is complete for SAS$^+$-IAO instances in Theorem 6.34.

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

Proof: Trivial by Lemma 6.12. □

Lemma 6.24 $Ker(\Delta) = Ker(\Theta)$

Proof: By Theorem 6.21, we know that $Ker(\Delta) \leq_{Ker} Ker(\Theta)$. Assume there exists an x and an i 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 $\mathcal{R}^{\mathcal{A}}_{v_j}$ since $\mathcal{R}^{\mathcal{A}}_{v_j} \subseteq \mathcal{R}^{\mathcal{B}}_{v_j}$) 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 $\mathcal{R}^{\mathcal{A}}_{v_j}$) 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 6.15,
   - 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 6.15,
   - $\mathcal{R}^{\mathcal{A}}_{v_k} \subseteq \mathcal{R}^{\mathcal{B}}_{v_k}$.
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 6.10 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 both cases the assumption leads to a contradiction and, consequently, $Ker(\Delta) = Ker(\Theta)$. □

Lemma 6.25 For $1 \leq k \leq m$, $\delta_k \triangleq \theta_k$.

Proof: Straightforward. □

Lemma 6.26 $type(\mathcal{I}(\mathcal{A})) = type(\mathcal{I}(\mathcal{B}))$.

Proof: Obvious by Lemma 6.10. □

Definition 6.27 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 6.28 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 6.26 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 6.25, $|\delta_i| = |\theta_i|$ for $1 \leq i \leq m$. □

Lemma 6.29 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 6.25. □

Lemma 6.30 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 6.24, $\mathcal{R}^{\mathcal{B}}_{v_i} = \mathcal{R}^{\mathcal{A}}_{v_i}$. 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. □

Lemma 6.31 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: Similar to Lemma 6.30. □

Lemma 6.32 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 occurs 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 6.31, $\varphi(a) = \theta_i^k$ 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 outcomes. 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 6.29, $\mathsf{prv}(\varphi(b))[v_i] = x$. By Lemma 6.30 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 6.31, $\varphi(a) = \theta_i^k$ 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 6.33 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 6.32, 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. □

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

Theorem 6.34 If there exists some partial-order 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: $X_i^1 = \emptyset$ 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 6.19,

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

but by Lemma 6.16, $Extend(Ker(\Theta)) = Ker(\Theta)$. 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 6.21 every member of $Paths(v_i, \Theta)$ must pass every value in X_i which contradicts the existence of Θ.

ii) By Lemma 6.33, there cannot exist any plan solving Π, which contradicts the existence of Θ. □

We can now conclude this section.

Theorem 6.35 *Plan* is sound, complete and generate minimal solutions for SAS$^+$-IAO instances. Furthermore, *Plan* is sound for SAS$^+$-IA instances.

7 Complexity Results

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

7.1 Preliminaries

For technical reasons, we will need a restricted variant of the SAS$^+$ formalism in this section. It is defined as follows.

Definition 7.1 An instance $\langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ of the SAS* problem is an instance of the SAS$^+$ problem satisfying the following two restrictions.

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 $\mathsf{post}(o)[v] = \mathsf{u}$.

One of the reasons 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 7.1, 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 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. The definitions given here follows the definitions given in Bäckström [3] and Bäckström & Nebel [6].

Definition 7.2 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 10 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$^+$ holds 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 intelligible summary will be given later on.

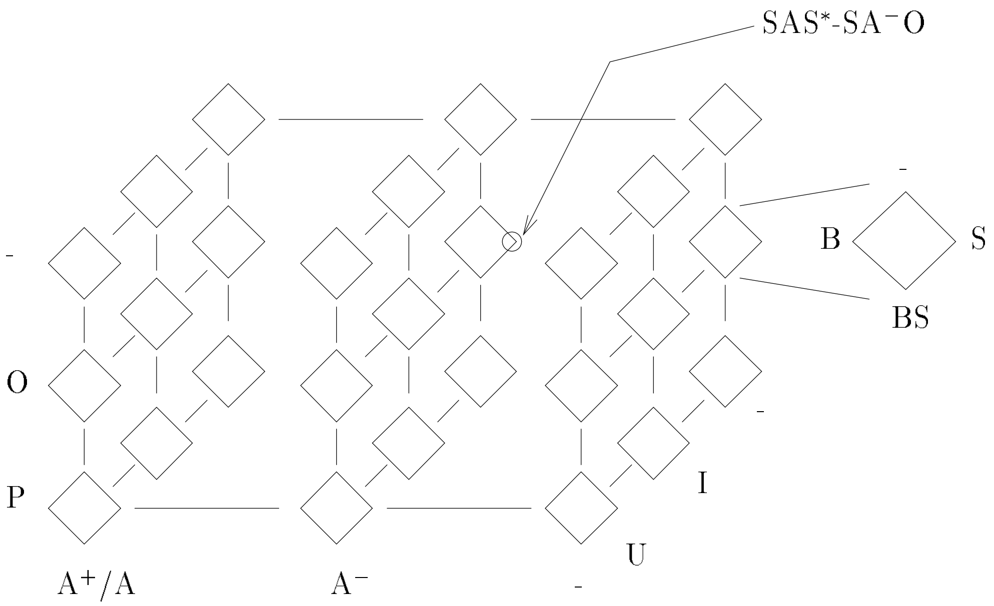


Figure 10: The lattice of restricted SAS* problems.

7.2 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 7.3 A *simple cycle* is a one-component graph such that every vertex has incoming and outgoing degree one.

Definition 7.4 A directed graph $G = \langle V, A\rangle$ is a *P-graph* iff for every component $C = \langle W, B\rangle$ in G,

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 (possibly empty) directed tree in $C' = \langle W, F - \{\langle x, y\rangle \mid x, y \in W'\}\rangle$,

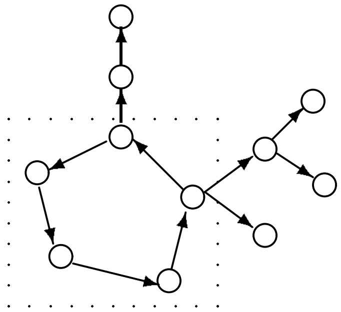


Figure 11: A sample one-component P-graph. The center is within the dotted box.

Theorem 7.5 The set of operators affecting a SAS* variable v is post-unique (wrt. to v) iff G_v is a P-graph.

Proof: Straightforward. $\Box$

Lemma 7.6 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. $\Box$

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

Definition 7.7 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$\subseteq$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 7.8 The following subproblem relations hold:

1. SAS*-A$^+$ $\subseteq$ SAS*-A
2. SAS*-A $\subseteq$ SAS*-A$^-$
3. SAS*-U $\subseteq$ SAS*-I
4. SAS*-US $\subseteq$ SAS*-A
5. SAS*-S $\subseteq$ SAS*-A$^-$
6. SAS*-PA $\subseteq$ SAS*-I
7. SAS*-P $\subseteq$ 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 7.6.
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 in G_v and let ψ be an arbitrary path from x to y via X in G_v. 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, $y \neq \mathsf{u}$ and $y = \mathsf{u}$. We begin with the case $y \neq \mathsf{u}$. If $x \notin W'$, then there exists a unique path from x to y by Theorem 7.5 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 7.5, $\omega = \psi$ 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 7.5, 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.

The proof for the case $y = \mathsf{u}$ is similar. □

A diagram summarizing the results in Theorem 7.8 can be found in Figure 12. 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 7.1 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.

7.3 Tractability Results

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

Theorem 7.9 [6] **PG** for SAS⁺-US is polynomial.

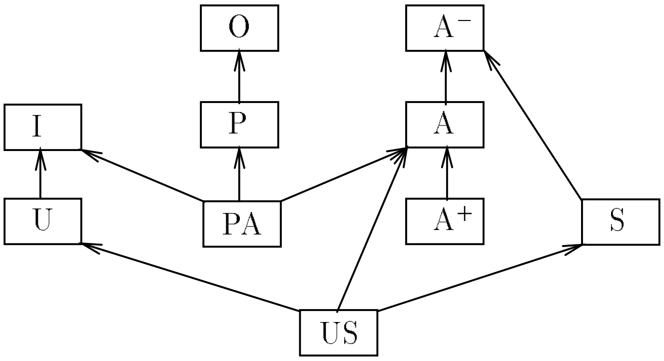


Figure 12: Summary of Theorem 7.8.

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 7.10 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. Let s' and t' be two new domain values. Then, $New_o = \{o'\} \cup \{o_v^{\mathsf{pre}}, o_v^{\mathsf{post}} \mid \mathsf{post}(o)[v] \neq \mathsf{u}\}$ where

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

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

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

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

By Lemma 6.2, 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 13. The upper part of Figure 13 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 13 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 7.11 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: Tedious but straightforward. The only-if part is performed by induction over the length of the shortest path from x to y. The if part builds on simple observations regarding irreplaceable operators. □

Lemma 7.12 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 7.13 Bounded plan generation for SAS$^+$-IA$^-$O is polynomial.

Proof: Let Π be an arbitrary SAS$^+$-IA$^-$O instance. By Lemma 7.11, $\Pi' = Atr(\Pi)$ 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 7.12, 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 substitute 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. □

We also need a polynomial-time algorithm that can decide if a SAS$^+$-IA$^-$ instance is prevail-order-preserving or not.

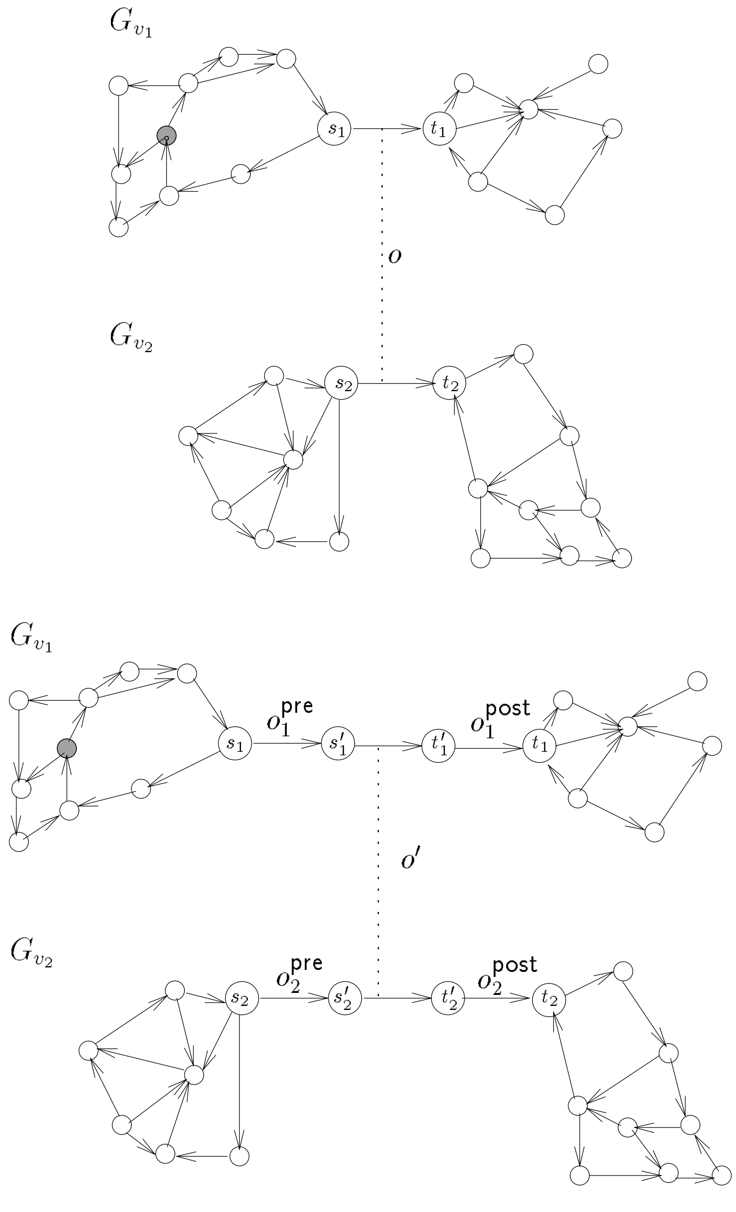


Figure 13: Domain-transition graphs for v_1 and v_2 before and after the A-transform.

Theorem 7.14 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 section 3 to decide whether $Atr(\Pi)$ is prevail-order-preserving or not. By Lemma 7.11, $Atr(\Pi)$ is a SAS$^+$-IAO instance iff Π is a SAS$^+$-IA$^-$O instance and the theorem follows. □

7.4 Intractability Results

For the intractability results we have to distinguish between those problems that are inherently intractable, *i.e.* can be proven to take exponential time, and those which are NP-equivalent (*i.e.* intractable unless P=NP)[5]. See Johnson [18] or Garey and Johnson [15] for formal details. We begin by showing some results, concering NP-easiness.

Lemma 7.15 The length of minimal solutions for SAS*-A instances are bounded by $|\mathcal{V}| \cdot (1 + \max_{v \in \mathcal{V}} \mathcal{D}_v)^2$.

Proof sketch: There can be at most $\mathcal{D}_v$ requestable values and the shortest paths between them can be of length $\mathcal{D}_v$ at most. □

Lemma 7.16 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 7.15. □

Lemma 7.17 Minimal solutions are 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

[5] 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.

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 7.16, 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 7.18 **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 [15], as the basis for our reductions. They are defined as follows:

Definition 7.19 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 7.20 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 7.21 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 7.22 **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 $v \in \mathcal{V}$;

- $\mathcal{O} = \{o_k^+ \mid 1 \le k \le n\} \cup \{o_{k,l} \mid 1 \le k \le n \text{ and } x_l \in C_k\}$, where for $1 \le k \le n$ 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 $1 \le k \le n$, $x_l \in C_k$ 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 $1 \le k \le n$, $s_0[x_l] = 0$ for $1 \le l \le m$,
- $s_*[c_k] = \mathsf{u}$ for $1 \le k \le n$, $s_*[x_l] = 1$ for $1 \le l \le m$.

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

Theorem 7.23 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

- $\mathcal{V} = \{x_k \mid 1 \le k \le 3m\}$;
- $\mathcal{D}_v = \{0, 1\}$ for all $v \in \mathcal{V}$;
- $\mathcal{O} = \{o_k^+ \mid 1 \le l \le n\}$ where for $1 \le k \le 3m$ and $1 \le l \le n$,

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

- $s_0[x_k] = 0$ for $1 \le k \le 3m$,

- $s_*[x_k] = 1$ for $1 \le k \le 3m$.

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. □

Theorem 7.24 **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

- $\mathcal{V} = \{u_k \mid 1 \le k \le m\} \cup \{c_l \mid 1 \le l \le n\}$.
- $\mathcal{D}_{u_k} = \{F, T\}$ for $1 \le k \le m$ and $\mathcal{D}_{c_l} = \{F, T\}$ for $1 \le l \le n$.
- $\mathcal{O} = \{o_k^+ \mid 1 \le k \le m\} \cup \{o_{l,p} \mid 1 \le l \le n \text{ and } u_p \in C_l\}$ where for $1 \le i \le m$, $1 \le j \le n$ 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 $1 \le k \le m$. $s_0[c_l] = F$ for $1 \le l \le n$.
- $s_*[u_k] = \mathsf{u}$ for $1 \le k \le m$. $s_*[c_l] = T$ for $1 \le l \le n$.

Obviously there exists a satisfying truth assignment for C iff there exists a plan solving Π. □

Theorem 7.25 **PE** is NP-hard for SAS*-BSIA$^+$.

Proof: Proof by reduction from SAT. The construction is similar to the one used in the previous theorem. □

Finally, we state a result about inherently intractable problems, previously found in the literature as indicated.

Theorem 7.26 [6] Both SAS*-PUB and SAS*-PBS have instances with exponentially sized minimal solutions (and are thus inherently intractable).

The original proof [5] of Theorem 7.26 is somewhat stronger and shows that SAS-PUB and SAS-PBS have instances with exponentially sized minimal solutions.

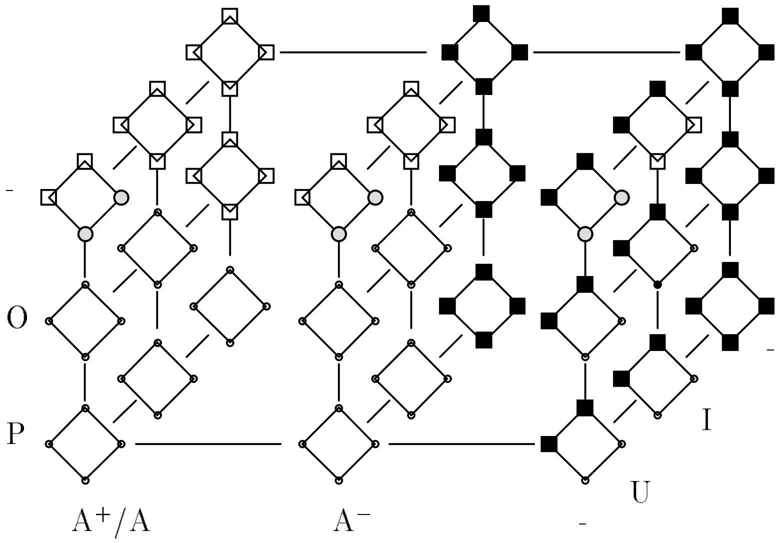
7.5 Summary

By combining the hardness and easiness results of the previous sections, we attain an exhaustive map over the complexity results. The resulting lattice is shown in Figure 14.

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 8. 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 here. If it satisfies IA$^-$ but not restriction O, then it must also 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 satisfies IA$^-$ and is prevail-order-preserving, then we can find a minimal solution in polynomial time by using the A-transform and the SAS$^+$-IAO algorithm.

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

Table 8: Simplified map of the complexity of SAS* plan generation.



- Polynomial for **PG** and **BPG**
- Polynomial for **PG**, NP-equivalent for **BPG**
- NP-equivalent for **PG** and **BPG**
- Inherently intractable.

Figure 14: Complexity of SAS* plan generation.

8 Concluding Remarks

8.1 Conclusion

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 article. 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.

8.2 Discussion

Having read this far, the reader may rightfully ask whether we believe it possible to eventually find tractable classes covering most application problems. The answer, however, is not as simple as the question. While we believe that many application problems in structured environments, such as industry, are inherently tractable, this fact may not be easily exploited. Finding and exploiting the underlying structure causing tractability may be non-trivial. In other cases, tractability may only hold for certain values of parameters which cannot be easily bounded, but can be assumed from experience to have reasonable values.[6] While this latter case does not guarantee formal tractability, it may guarantee tractability under a certain hypothesis, which may be quite reasonable in many cases. It is also worth pointing out that in many cases of NP-complete planning problems, NP-completeness is likely to stem from a scheduling and/or resource allocation subproblem, while planning, *i.e.* finding the actions, is simple. In this case, a deliberate separation of the problems may enable the use of an efficient scheduler/resource allocator and a simple planner in combination. A concrete example of how to use the SAS$^+$-IAO planner in a setting having many similarities with industrial processes can be found in Klein *et al.* [25].

Furthermore, we should neither be too obsessed by formal tractability only. The search for tractable subclasses and the complexity analysis of restrictions can provide useful information about how to find efficient, although not polynomial, algorithms for other restricted classes (*c.f.* results in temporal reasoning [34]). For instance, in some of our algorithms it may be possible to relax some restriction and introduce a limited form of search. Another possibility is to build up a library of algorithms for tractable subclasses and then use these as subroutines in a more general search-based planner, or to use a classifier and invoke one of these algorithms whenever possible and otherwise use a general search-based planner.

It may be worth pointing out that the restricted-problem approach is somewhat similar to knowledge-based planning as used in HTN planners such as O-Plan [11]. In an HTN planner, expert knowledge is encoded as task reduction schemata having the effect of allowing only a small portion of the whole search-space of plans to be explored. Tate [pers. comm. 1995] argues that search should be avoided entirely whenever possible. In principle, the

[6] An example of a *constant-by-experience* parameter limiting the plan length appears in manufacturing planning [27].

main difference between this approach and ours is whether the information used to avoid or reducing search comes from expert knowledge or from a formal analysis of the problem. The HTN approach is more general, but not formally verifiable, while our approach requires special algorithms tailored to subclasses, but provides formal guarantees.

Similarly, using a general search-based planner like TWEAK or SNLP equipped with heuristics to prune or reorder the search tree is also a complementary approach. In principle, it may be possible to tailor such a planner to a tractable class, but the heuristic needed for this may be quite complex and non-trivial to find, and a tailored algorithm is most likely more efficient.

Another issue is whether our approach scales up to handle more realistic applications requiring reasoning about uncertainty, resources and metric time. This question remains to be satisfactorily answered for the other approaches we have just discussed, too. As far as our approach is concerned, the truth is that we have hope but no guarantee. However, even if our approach does not scale up, our research is hardly wasted; we strongly believe that we have to thoroughly understand these simpler formalisms before we can formally attack and understand more expressive ones, and that many lessons learned will carry over. An example supporting this is the paper on temporal projection by Lin and Dean [17]. Dean [pers. comm. 1993] says that the problem they are trying to solve is temporal projection with uncertainty, but after having spent some years trying this they found the problem so difficult they had to switch back, analysing the basic case first.

Acknowledgements

We would like to thank Marco Cadoli for helpful comments and Jonas Kvarnström for implementing the *Plan* algorithm. This research has been sponsored by the Swedish Research Council for the Engineering Sciences (TFR) under grants Dnr. 92-143 and Dnr. 93-00291.

References

[1] C. Bäckström, Computational complexity of reasoning about plans, Ph.D. Thesis, Linköping University, Linköping, Sweden (1992).

[2] C. Bäckström, Equivalence and tractability results for SAS^+ planning, in: Proc. 3rd Int'l Conf. on Principles of Knowledge Repr. and Reasoning (KR-92), Cambridge, MA, USA (1992) 126–137.

[3] C. Bäckström, Expressive equivalence of planning formalisms, Artif. Intell. 76(1–2) (1995) 17–34.

[4] C. Bäckström, Five years of tractable planning, in: Ghallab and Milani [16], pages 19–33.

[5] C. Bäckström and I. Klein, Planning in polynomial time: The SAS-PUBS class, Comput. Intell. 7(3) (1991) 181–197.

[6] C. Bäckström and B. Nebel, Complexity results for SAS^+ planning, Comput. Intell. 11(4) (1995) 625–655.

[7] A. Barrett and D. S. Weld, Partial-order planning: Evaluating possible efficiency gains, Artif. Intell. 67(1) (1994) 71–112.

[8] A. L. Blum and M. L. Furst, Fast planning through planning graph analysis, Artif. Intell. 90 (1997) 281–300.

[9] T. Bylander, The computational complexity of propositional STRIPS planning, Artif. Intell. 69 (1994) 165–204.

[10] D. Chapman, Planning for conjunctive goals, Artif. Intell. 32 (1987) 333–377.

[11] K. Currie and A. Tate, O-Plan: The open planning architecture, Artif. Intell. 52 (1991) 49–86.

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

[13] K. Erol, D. S. Nau and V. S. Subrahmanian, When is planning decidable?, in: Proc. 1st Int'l Conf. on Artif. Intell. Planning Sys. (AIPS-92), College Park, MD, USA (1992) 222–227.

[14] R. E. Fikes and N. J. Nilsson, STRIPS: A new approach to the application of theorem proving to problem solving, Artif. Intell., 2 (1971) 189–208.

[15] M. Garey and D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness (Freeman, New York, 1979).

[16] M. Ghallab and A. Milani, editors, New Directions in AI Planning: Proc. 3rd Eur. WS. Planning (EWSP'95), Assisi, Italy (IOS Press, 1995).

[17] S. Hong Lin and T. Dean, Exploiting locality in temporal reasoning, in: C. Bäckström and E. Sandewall, editors, Current Trends in AI Planning: Proc. 2nd Eur. WS. Planning (EWSP'93), Vadstena, Sweden (1993) 199–212.

[18] D. S. Johnson, A catalog of complexity classes, in: J. van Leeuwen, editor, Handbook of Theoretical Computer Science: Algorithms and Complexity, volume A (Elsevier, Amsterdam, 1990) 67–161.

[19] P. Jonsson, Complexity of state-variable planning under structural restrictions, Licentiate thesis, Linköping University, Linköping, Sweden (1995).

[20] P. Jonsson and C. Bäckström, Complexity results for state-variable planning under mixed syntactical and structural restrictions, in: Proc. 6th Int'l Conf. on Artif. Intell.: Methodology, Systems, Applications (AIMSA-94), Sofia, Bulgaria (1994) 205–213.

[21] P. Jonsson and C. Bäckström, Complexity results for state-variable planning under mixed syntactical and structural restrictions, Research Report R-95-17, Department of Computer and Information Science, Linköping University (1995).

[22] P. Jonsson and C. Bäckström, Tractable planning with state variables by exploiting structural restrictions, in: Proc. 12th (US) Nat'l Conf. on Artif. Intell. (AAAI-94), Seattle, WA, USA (1994) 998–1003.

[23] P. Jonsson and C. Bäckström, Tractable planning with state variables by exploiting structural restrictions, Research Report R-95-16, Department of Computer and Information Science, Linköping University (1995).

[24] I. Klein, Automatic synthesis of sequential control schemes, Ph.D. Thesis, Linköping University, Linköping, Sweden (1993).

[25] I. Klein, P. Jonsson, and C. Bäckström, Tractable planning for an assembly line, in: Ghallab and Milani [16], pages 313–324.

[26] J. Kvarnström, Implementation of a tractable planner, Master thesis report, Department of Computer and Information Science, Linköping University, Linköping, Sweden (1996).

[27] A. Márkus and J. Váncza, Inference and optimization methods for manufacturing process planning, in: A. G. Cohn, editor, Proc. 11th Eur. Conf. on Artif. Intell. (ECAI-94), Amsterdam, Netherlands (1994) 595–599.

[28] K. Mehlhorn, Graph Algorithms and NP-Comleteness, volume 2 (Springer, Berlin, 1984).

[29] R. Paige and F. Henglein, Mechanical translation of set theoretic problem specifications into efficient RAM code—A case study, Journal of Symbolic Computation 4 (1987) 207–232.

[30] E. Sandewall, The pipelining transformation on plans for manufacturing cells with robots, in: Proc. 10th Int'l Joint Conf. on Artif. Intell. (IJCAI-87), Milano, Italy (1987) 1055–1062.

[31] E. Sandewall and R. Rönnquist, A representation of action structures, in: Proc. 5th (US) Nat'l Conf. on Artif. Intell. (AAAI-86), Philadelphia, PA, USA (1986) 89–97.

[32] R. Tarjan, Depth-first search and linear graph algorithms, SIAM J. Comput. 1 (1972) 146–160.

[33] A. Tarski, A lattice-theoretical fixpoint theorem and its applications, Pacific Journal of Mathematics 5 (1955) 285–309.

[34] P. van Beek, Reasoning about qualitative temporal information, in: Proc. 8th (US) Nat'l Conf. on Artif. Intell. (AAAI-90), Boston, MA, USA (1990) 728–734.