To appear in *Computational Intelligence*, 1995.

Complexity Results for SAS$^+$ Planning*

Christer Bäckström
Department of Computer and Information Science,
Linköping University,
S-581 83 Linköping, Sweden
e-mail: cba@ida.liu.se
phone: +46 13 282429

Bernhard Nebel
Department of Computer Science
University of Ulm
D-89069 Ulm, Germany
e-mail: nebel@informatik.uni-ulm.de
phone: +49 731 5024122

May 7, 1995

Abstract

We have previously reported a number of tractable planning problems defined in the SAS$^+$ formalism. This report complements these results by providing a complete map over the complexity of SAS$^+$ planning under all combinations of the previously considered restrictions. We analyze the complexity both of finding a minimal plan and of finding any plan. In contrast to other complexity surveys of planning we study not only the complexity of the decision problems but also of the generation problems. We prove that the SAS$^+$-PUS problem is the maximal tractable problem under the restrictions we have considered if we want to generate minimal plans. If we are satisfied with any plan, then we can generalize further to the SAS$^+$-US problem, which we prove to be the maximal tractable problem in this case.

*This research was supported by the Swedish National Board for the Engineering Sciences (TFR) under grant Dnr. 92-143, by the German Ministry for Research and Technology (BMFT) under grant ITW 8901 8, and by the European Commission as part of DRUMS-II, the ESPRIT Basic Research Project P6156.

1 Introduction

Planning is the reasoning problem of finding a (totally or partially ordered) set of operators that achieves a specified goal from a given initial state.[1] It is well known that this problem is computationally intractable[2] in general [Chapman, 1987; Bylander, 1991; Bylander, 1994; Erol *et al.*, 1992]. In fact, domain-independent planning systems that are guaranteed to find a solution if there exists one, *i.e.*, *complete* planners, are bogged down by the huge search space.[3] One way to deal with this problem is to use heuristic search techniques and to sacrifice completeness. Another way is to give up on expressiveness of the planning formalism in order to achieve efficiency.

Focusing on the latter alternative, we have previously studied planning problems closely related to problems in the area of *sequential control*.[4] Using the SAS and SAS$^+$ formalisms,[5] which can be viewed as variants of propositional STRIPS [Fikes and Nilsson, 1971] or ground TWEAK [Chapman, 1987], we considered a number of local and global restrictions and analyzed the resulting complexity of the planning problem. The overall goal of this research has been to identify successively more and more expressive, computationally tractable planning problems, with the hope of ultimately finding problems which are relevant to practical applications, especially in the area of sequential control.

The restrictions we considered can be informally characterized as follows:

Post-uniqueness: For each effect there is at most one operator that achieves this effect. In other words, desired effects determine the operators to be used in a plan.

Single-valuedness: Preconditions such that the corresponding state variables are not changed by the operator must all have the same value, when defined. For instance, if a certain operator requires the light to be on, no other operator must use the precondition that the light is off without also changing the state of the light.

Unariness: Each operator affects only one state variable.

Binariness: All state variables have exactly two possible values.

The two former restrictions are global in the sense that they apply to a set of operators. *Post-uniqueness*, is a very severe restriction that simplifies the planning process considerably because for each value of a state variable there is

[1] In other words, by "planning" we mean classical, single-agent, non-parallel planning.

[2] As customary, we call a problem computationally intractable if it is NP-hard [Garey and Johnson, 1979].

[3] This is indicated, for instance, by the empirical data reported by Kambhampati and Hendler [1992, Fig. 12] and by Hanks and Weld [1992, Fig. 5].

[4] Sequential control is a subdisciplin of *automatic control*, concerned mainly with generation and analysis of sequences of control actions for industrial processes *etc.* Hence, the area includes many planning-like problems. A more in-depth discussion of the relationship between planning (especially our work) and sequential control can be found in Klein [1993].

[5] SAS is an acronym for *Simplified Action Structures,* denoting that the formalism is a restriction of the *action structure* formalism [Sandewall and Rönnquist, 1986].

at most one operator that can achieve this value. *Single-valuedness* resembles the restriction that only positive preconditions are allowed [Bylander, 1991; Erol *et al.*, 1992], but is slightly more general in that single-valuedness only applies to the preconditions such that the corresponding state variables are not changed. The two latter restrictions are local to the operators. *Binariness* is just the restriction we find in propositional STRIPS and ground TWEAK, namely that each state variable can be either true or false. *Unariness*, finally, means that we have only one postcondition for each operator, a restriction also considered by other authors [Bylander, 1991].

Starting with the SAS formalism and imposing all of the restrictions sketched above, we get the SAS-PUBS[6] planning problem—the first problem we proved to be solvable in polynomial time [Bäckström and Klein, 1990, 1991b]. Applying a bottom-up strategy, we then generalized this result by removing restrictions, resulting in the more expressive, but still tractable, SAS-PUS [Bäckström and Klein, 1991a] and SAS$^+$-PUS [Bäckström, 1992a, 1992b] problems.

Having started from one tractable problem and generalized twice to new tractable problems it is interesting to ask how much further we can generalize and stay tractable by simply removing restrictions. This report answers that question by providing an exhaustive map over the complexities for generating both optimal and non-optimal plans for all possible combinations of the P, U, B and S restrictions. It turns out that the SAS$^+$-PUS problem is the maximally expressive tractable problem if we insist on generating optimal plans (*i.e.*, plans of minimal length). Whichever of the three restrictions on this problem we drop, the resulting problem is intractable. On the other hand, if we do not require the solutions to be minimal, then we can generalize somewhat further; if we remove the P restriction, resulting in the SAS$^+$-US problem, we can still plan in polynomial time, although we are no longer guaranteed to generate optimal plans. However, if we further remove either of the two restrictions on the SAS$^+$-US problem, it becomes inherently intractable to generate a plan since there may then be minimal plans of exponential length.

The remainder of this report is structured as follows. The next section contains a description of the SAS$^+$ formalism and an example showing how to model planning problems. Section 3 defines the different planning problems we analyze in this report and provides a survey of our complexity results. In Section 4, we prove the hardness results and in Section 5, we specify an algorithm for the SAS$^+$-US plan generation problem. Finally, in Section 6, we summarize and discuss the results of our analysis.

2 The SAS$^+$ Formalism

The SAS$^+$ formalism is in principle a slight variation on the propositional STRIPS formalism [Fikes and Nilsson, 1971]. This section briefly recasts the main differences between the two formalisms and gives a formal definition of the SAS$^+$ formalism. The reader is referred to our previous publications [Bäckström

[6] We use the first letter of the names of a restrictions in order to denote what kind of subproblem we mean.

and Klein, 1991a, 1991b, Bäckström, 1992a, 1992b] for further background, motivation and examples.

2.1 World Modeling in the SAS$^+$ Formalism

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

A multi-valued, discrete state variable can, in principle, be simulated by a number of propositional atoms, but there are at least three good reasons for using multi-valued state variables. First, many applications are more naturally described in this way, especially in *sequential control*. Secondly, we have been able to isolate certain restrictions on planning problems that seem relevant for real problems and which reduce the computational complexity considerably. These restrictions would most likely have been very hard to isolate using the traditional STRIPS formalism. Thirdly, it will be a smaller step to generalize the state variables to structured domains such as subsets of the integers or interval-valued domains.[7] Re-expressing these restrictions in a traditional formalism is, in principle, possible, but the result is very complicated and unintuitive [Bäckström, 1992a; Nebel and Bäckström, 1994]. Furthermore, recent research [Bäckström, 1994b] has shown that there may be a considerable loss of efficiency if solving state-variable planning instances by first re-encoding them as propositional STRIPS instances.

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.

Plans consist of actions (or plan steps), each action being an instantiation of some operator (or action type). Since all operators in the SAS$^+$ formalism are ground and we only consider total-order plans in this report, we need not distinguish between operators and actions. The main difference in contrast to the traditional STRIPS modeling is that the (STRIPS) pre-condition is split into two conditions, depending on whether the variables are changed by the operator or only required to have some specific value. The 'behavior' of an operator is modeled by its *pre-*, *post-* and *prevail-conditions*—all three being partial states. The post-condition of an operator expresses which state variables it changes and what values these variables will have after executing the operator, if executed successfully. The pre-condition specifies which values these changed variables must have before the operator is executed. The prevail-condition specifies which of the unchanged variables must have some specific value before

[7] We have previously presented a variant of action structures with interval-valued state variables [Bäckström, 1988a, 1988b].

the execution of the operator and what these values are.[8] Both the pre- and prevail-condition of the operator must be satisfied for the operator to execute successfully.

Definition 2.1 An instance of the SAS$^+$ planning problem is given by a tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ 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** $\mathcal{S}_\mathcal{V} = \mathcal{D}_{v_1} \times \ldots \times \mathcal{D}_{v_m}$ and the **partial state space** $\mathcal{S}_\mathcal{V}{}^+ = \mathcal{D}_{v_1}^+ \times \ldots \times \mathcal{D}_{v_m}^+$ are implicitly defined. We write $s[v]$ to 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 $\mathsf{pre}, \mathsf{post}, \mathsf{prv} \in \mathcal{S}_\mathcal{V}{}^+$ denote the **pre-, post-** and **prevail-condition** respectively. $\mathcal{O}$ is subject to the following two restrictions: For every operator $\langle \mathsf{pre}, \mathsf{post}, \mathsf{prv} \rangle \in \mathcal{O}$,

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

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

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

□

Restriction R1 essentially says that a state variable can never be made undefined, once made defined by some operator. Restriction R2 says that the prevail-condition of an operator must never define a variable which is affected by the operator.

A variable v is **defined** in a state s iff $s[v] \neq \mathsf{u}$. We write $s \sqsubseteq t$ if the state s is subsumed (or satisfied) by state t, *i.e.* if $s[v] = \mathsf{u}$ or $s[v] = t[v]$. We extend this notion to whole states, defining

$$s \sqsubseteq t \text{ iff } \quad \text{for all } v \in \mathcal{V}, s[v] = \mathsf{u} \text{ or } s[v] = t[v].$$

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.

Given a variable v and two values $x, y \in \mathcal{D}_v$, s.t. either $x = \mathsf{u}$ or $y = \mathsf{u}$ the operation $\sqcup$ is defined as

$$x \sqcup y \quad = \quad \begin{cases} x & \text{if } y = \mathsf{u}, \\ y & \text{if } x = \mathsf{u}, \end{cases}$$

(note that $\mathsf{u} \sqcup \mathsf{u} = \mathsf{u}$). The $\sqcup$ operator is extended to states s.t. given two states s and t, $(s \sqcup t)[v] = s[v] \sqcup t[v]$ for all $v \in \mathcal{V}$.

[8] If it is possible to execute actions in parallel, the prevail-conditions are additionally required to hold during the execution of an action [Bäckström, 1988a]. However, this is not relevant in our case.

Furthermore, an operator o **affects** a variable $v \in \mathcal{V}$ iff $\mathsf{post}(o)[v] \neq \mathsf{u}$. Given a set of operators $\mathcal{O}$ the **restriction of $\mathcal{O}$ to** v is written $\mathcal{O}[v]$ and is defined as the set of all operators in $\mathcal{O}$ that affect v.

It should be noted that the SAS$^+$ planning model allows not only the goal state and the pre-conditions to be partial but also the initial state. This allows for a rudimentary modeling of uncertain initial states. Most standard variants of STRIPS do not allow for this.

2.2 Plans

According to usual definitions in planning a *total-order (or linear) plan* is a sequence of actions and a *partial-order (or non-linear) plan* is a partially ordered set of actions. The actions in a partial-order plan can be executed in any total order consistent with the partial order of the plan, that is, a partial-order plan can be viewed as a compact representation for a set of total-order plans. In this report we will only consider total-order plans. Since several instances of the same operator are uniquely determined by their positions in a total-order plan, we do not distinguish between actions and operators in the following.

The restriction to total-order plans is reasonable since any planning problem is solved by some total-order plan iff it is solved by some partial-order plan.[9] This means that all our hardness results for total-order planning apply to partial-order planning as well.

Definition 2.2 Given a set of variables $\mathcal{V}$ and a set of operators $\mathcal{O}$ over $\mathcal{V}$, a **plan** over $\mathcal{O}$ is a sequence $\bar{\alpha} = \langle o_1, \ldots, o_n \rangle$ of operators s.t. $o_k \in \mathcal{O}$ for $1 \leq k \leq n$. □

The **length** of $\bar{\alpha}$ is denoted $|\bar{\alpha}|$ and is defined as the number of operators in $\bar{\alpha}$, *i.e.* $|\langle o_1, \ldots, o_n \rangle| = n$. Given a plan $\bar{\alpha} = \langle o_1, \ldots, o_n \rangle$ the notation $\bar{\alpha}/k$ denotes the plan $\langle o_1, \ldots, o_k \rangle$. Furthermore, each of the sequences $\langle\rangle$ and $\bar{\alpha}/k$ for all $1 \leq k \leq n$ is called a **prefix** of $\bar{\alpha}$.

Two plans $\bar{\alpha} = \langle o_1, \ldots, o_m \rangle$ and $\bar{\beta} = \langle o'_1, \ldots, o'_n \rangle$ can be **concatenated**, which is written $(\bar{\alpha};\bar{\beta})$ and defined as $(\bar{\alpha};\bar{\beta}) = \langle o_1, \ldots, o_m, o'_1, \ldots, o'_n \rangle$. Given a plan $\bar{\alpha}$ and an operator o, we write $(\bar{\alpha};o)$ and $(o;\bar{\alpha})$ as abbreviations for $(\bar{\alpha};\langle o \rangle)$ and $(\langle o \rangle;\bar{\alpha})$ respectively. Furthermore, since the concatenation operator is obviously associative we will usually omit grouping parentheses.

Definition 2.3 Given two states $s, t \in \mathcal{S}_{\mathcal{V}}{}^+$, $(s \oplus t)$ denotes that s is *updated by* t which is defined s.t. for all $v \in \mathcal{V}$,

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

[9] Any partial-order plan can be trivially converted into a total-order plan by using topological sorting, which takes low-order polynomial time. However, contrary to some claims in the literature, converting total-order plans into partial-order plans subject to some useful minimization criterion is not as easy [Bäckström, 1993].

The function *result* gives the state resulting from executing a plan and is defined recursively as

$$\begin{aligned} result(s, \langle\rangle) &= s, \\ result(s, (\bar{\alpha};o)) &= \begin{cases} result(s, \bar{\alpha}) \oplus \mathsf{post}(o) & \text{if } (\mathsf{pre}(o) \sqcup \mathsf{prv}(o)) \sqsubseteq result(s, \bar{\alpha}), \\ s & \text{otherwise .}^{10} \end{cases} \end{aligned}$$

□

This definition of the *result* function 'solves' the *frame problem* by employing the *STRIPS assumption*, which is sufficient in this restricted formalism.

An operator o is **admissible** in a state s iff $(\mathsf{pre}(o) \sqcup \mathsf{prv}(o)) \sqsubseteq s$. A plan $\bar{\alpha} = \langle o_1, \ldots, o_n \rangle$ is admissible in a state s iff either $\bar{\alpha}$ is empty or o_k is admissible in $result(s, \bar{\alpha}/(k-1))$ for all $1 \leq k \leq n$.

Given a set $\mathcal{V}$ of variables, a set $\mathcal{O}$ of variables over $\mathcal{V}$ and two states $s, t \in {\mathcal{S}_{\mathcal{V}}}^+$, we say that a plan $\bar{\alpha}$ over $\mathcal{O}$ is a **plan from** s **to** t iff

1. $\bar{\alpha}$ is admissible in s and
2. $t \sqsubseteq result(s, \bar{\alpha})$.

Sometimes, we also say that such a plan **solves** the SAS$^+$ instance $\langle \mathcal{V}, \mathcal{O}, s, t \rangle$. Further, a plan $\bar{\alpha}$ from s to t is **optimal** if there exists no plan from s to t containing fewer operators. That is, we make the assumption that all operators have the same cost, which is reasonable for hardness results; a hardness result will automatically carry over to the case where operators may have different costs. In the only case where we prove an upper bound for optimal plan generation, the SAS$^+$-PUS problem, the assumption does not matter either; there is no choice of actions in this case.

We say that a plan $\bar{\alpha}$ from some state s to some state t **passes** a state s' iff $result(s, \bar{\beta}) = s'$ for some possibly empty prefix of $\bar{\alpha}$. Further given some index $v \in \mathcal{V}$ and some value $x \in \mathcal{D}_v^+$ we say that $\bar{\alpha}$ **passes** x iff $result(s, \bar{\beta})[v] = x$ for some possibly empty prefix of $\bar{\alpha}$.

2.3 An Example: The Cappucino Brewer

Below we will illustrate the SAS$^+$ formalism using an example based on a simple idea of how an automatic cappucino brewer could work. This example is chosen for illustrative purpose only. The brewer (see Figure 1) consists of a steam container, a coffee supply, a coffee filter and a milk steamer. In order to brew a cup of espresso there must be fresh coffee in the filter and steam in the steam container. If the milk in the milk steamer has been steamed to foam we might also top the espresso with milk foam, thus getting a cappucino (ignoring sugar and cocoa powder). We assume there is always coffee in the coffee supply and always milk or milk foam in the milk steamer. The state variables and their domains are described in Table 1.

[10] This is an arbitrary choice; since we are only interested in plans where all operators have their pre- and prevail-conditions satisfied, it does not matter which result we specify in this case.

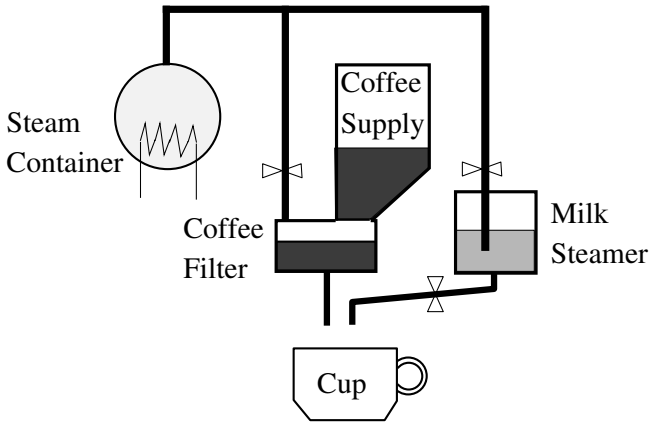

Figure 1: A simplified cappucino brewer

Index	Description	Domain
1	Steam Pressure	$\{\mathsf{H}igh, \mathsf{L}ow\}$
2	Coffee in filter	$\{\mathsf{F}resh, \mathsf{U}sed, \mathsf{N}one\}$
3	Cup content	$\{\mathsf{N}othing, \mathsf{E}spresso, \mathsf{C}appucino\}$
4	Milk steamed	$\{\mathsf{T}rue, \mathsf{F}alse\}$

Table 1: State variables for the cappucino brewer

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

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

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

We further assume that the set of operators in Table 2 is available. We have assumed that the operator HEAT-STEAM heats the steam (or water) in the steam container until reaching operational pressure, whatever pressure (or water temperature) it starts from. Similarly, the operator STEAM-MILK can be executed even if there is already foam in the milk steamer. The operator FILL-FILTER dispenses fresh coffee into the coffee filter and can be executed only if the filter is empty. To the contrary, the operator EMPTY-FILTER will empty the coffee filter even if it was previously empty. The operator BREW-ESPRESSO brews espresso and dispenses it into the cup subject to the condition that the steam pressure is sufficient and there is fresh coffee in the filter. Finally, the operator TOP-WITH-FOAM fills the cup with milk foam.

We assume the initial state is

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

action type	pre	post	prv
HEAT-STEAM	$\langle \mathsf{u}, \mathsf{u}, \mathsf{u}, \mathsf{u} \rangle$	$\langle \mathsf{H}, \mathsf{u}, \mathsf{u}, \mathsf{u} \rangle$	$\langle \mathsf{u}, \mathsf{u}, \mathsf{u}, \mathsf{u} \rangle$
FILL-FILTER	$\langle \mathsf{u}, \mathsf{N}, \mathsf{u}, \mathsf{u} \rangle$	$\langle \mathsf{u}, \mathsf{F}, \mathsf{u}, \mathsf{u} \rangle$	$\langle \mathsf{u}, \mathsf{u}, \mathsf{u}, \mathsf{u} \rangle$
EMPTY-FILTER	$\langle \mathsf{u}, \mathsf{u}, \mathsf{u}, \mathsf{u} \rangle$	$\langle \mathsf{u}, \mathsf{N}, \mathsf{u}, \mathsf{u} \rangle$	$\langle \mathsf{u}, \mathsf{u}, \mathsf{u}, \mathsf{u} \rangle$
BREW-COFFEE	$\langle \mathsf{u}, \mathsf{F}, \mathsf{N}, \mathsf{u} \rangle$	$\langle \mathsf{u}, \mathsf{U}, \mathsf{E}, \mathsf{u} \rangle$	$\langle \mathsf{H}, \mathsf{u}, \mathsf{u}, \mathsf{u} \rangle$
STEAM-MILK	$\langle \mathsf{u}, \mathsf{u}, \mathsf{u}, \mathsf{u} \rangle$	$\langle \mathsf{u}, \mathsf{u}, \mathsf{u}, \mathsf{T} \rangle$	$\langle \mathsf{H}, \mathsf{u}, \mathsf{u}, \mathsf{u} \rangle$
TOP-WITH-FOAM	$\langle \mathsf{u}, \mathsf{u}, \mathsf{E}, \mathsf{u} \rangle$	$\langle \mathsf{u}, \mathsf{u}, \mathsf{C}, \mathsf{u} \rangle$	$\langle \mathsf{u}, \mathsf{u}, \mathsf{u}, \mathsf{T} \rangle$

Table 2: Operators for the cappucino brewer

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

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

and to make a cappucino we set the goal

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

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

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

⟨FILL-FILTER, HEAT-STEAM, BREW-ESPRESSO, STEAM-MILK,
TOP-WITH-FOAM⟩

and

⟨HEAT-STEAM, FILL-FILTER, EMPTY-FILTER, FILL-FILTER,
BREW-ESPRESSO, EMPTY-FILTER,STEAM-MILK,
EMPTY-FILTER, TOP-WITH-FOAM⟩

are valid plans for making a cup of cappucino (according to our modeling[11]), while the following plan is not valid,

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

⟨FILL-FILTER, HEAT-STEAM, BREW-ESPRESSO, EMPTY-FILTER,
TOP-WITH-FOAM⟩

For further examples of problem modeling in the SAS$^+$ formalism appear in other publications [Bäckström, 1992a; Jonsson and Bäckström, 1994b].

3 SAS$^+$ Planning Problems

When analyzing the computational properties of planning, one is usually interested in the problem of *generating* a plan from the initial to the goal state. More specifically, one may be interested in generating either an *arbitrary* or a *minimal* plan. In the following, these problems are called **plan generation** and **optimal plan generation**, respectively. Corresponding to these *search problems*, one can define the *decision problems* of determining whether there exists a plan at all, the **plan existence** problem, and the problem of deciding whether there exists a plan of prespecified length, the **bounded plan existence** problem. Although decision problems are usually tightly related to their corresponding search problems [Garey and Johnson, 1979], there are some subtleties to be considered. While it is clear that a search problem is always as hard as the corresponding decision problem, it is not guaranteed that the search problem is always as easy as the corresponding decision problem. In particular, if the length of the solution cannot be guaranteed to be polynomial, tractability of the decision problem does not imply that the search problem can be solved in polynomial time, simply because writing down the solution may take too much time. For this reason, we consider the search as well as the decision problems in the following.

3.1 Subproblems of the SAS$^+$ Planning Problems

In some of our previous publications [Bäckström and Klein, 1991a, 1991b] we used a more restricted variant of the SAS$^+$ formalism—the SAS formalism. There are two differences between SAS and SAS$^+$. The first one is that both the initial and goal states must be total for instances of the SAS planning problem. The second is that an operator cannot change a state variable from undefined to some defined value, only from one defined value to another defined value.

Definition 3.1 An instance $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ of the SAS$^+$ planning problem is an instance of the **SAS planning problem** iff it satisfies the following two restrictions:

(R3) $s_0, s_\star \in \mathcal{S}_\mathcal{V}$.

(R4) for all $o \in \mathcal{O}$, $\mathsf{pre}(o) = \mathsf{u}$ iff $\mathsf{post}(o) = \mathsf{u}$.

□

We will also be interested in four other restrictions that we have previously studied [Bäckström and Klein, 1991a, 1991b]. An instance of the SAS$^+$ (bounded) plan existence or (optimal) plan generation problem is *post-unique* (P)

iff no two distinct operators can change the same state variable to the same value and the instance is *unary* (U) iff each operator changes exactly one state variable. It is, further, *binary* (B) iff all state variable domains are two-valued. 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.

Definition 3.2 Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ be an instance of a SAS$^+$ planning problem. For all $v \in \mathcal{V}$, the domain $\mathcal{D}_v$ is

(B) **binary** iff $|\mathcal{D}_v| = 2$.

The set $\mathcal{O}$ is

(P) **post-unique** iff for each $v \in \mathcal{V}$ and $x \in \mathcal{D}_v$, there is at most one $o \in \mathcal{O}$ s.t. $\mathsf{post}(o)[v] = x$;

(U) **unary** iff for all $o \in \mathcal{O}$, $\mathsf{post}(o)[v] \neq \mathsf{u}$ for exactly one $v \in \mathcal{V}$;

(S) **single-valued** iff there exist no two operators $o, o' \in \mathcal{O}$ and $v \in \mathcal{V}$ s.t. $\mathsf{prv}(o)[v] \neq \mathsf{u}$, $\mathsf{prv}(o')[v] \neq \mathsf{u}$ and $\mathsf{prv}(o)[v] \neq \mathsf{prv}(o')[v]$.

These definitions are extended to Π s.t. Π is binary iff $\mathcal{D}_v$ is binary for all $v \in \mathcal{V}$, and Π is post-unique, unary and single-valued iff $\mathcal{O}$ is post-unique, unary and single-valued respectively. □

These four restrictions were identified by studying a test example in automatic control (the tunnel problem [Bäckström, 1992a, pages 16–17]), thus complementing the usual problems from the AI world. For a somewhat more elaborate discussion of the restrictions, see Bäckström and Klein [1991a] or Bäckström [1992a].

The above restrictions, R3, R4, P, U, B and S, induce a space of subproblems of the SAS$^+$ problems. We name these subproblems systematically as follows. The SAS$^+$ problems restricted to instances satisfying R3 and R4 are called the SAS problems. The SAS or SAS$^+$ problems restricted to instances that satisfy some subset of the restrictions P, U, B and S is named by appending the corresponding letters to the name. For example, the SAS$^+$-PS problems are restricted to instances that are post-unique and single-valued while the SAS-UBS problems are restricted to instances that satisfy the restrictions R3 and R4 and which are also unary, binary and single-valued. For example, the cappucino brewer in the previous section is an instance of the SAS$^+$-PS problem, while not satisfying any further restrictions (*i.e.* it is not an instance of either the SAS-PS, SAS$^+$-PBS or SAS$^+$-PUS problem[12]).

For single-valued operator sets, there is obviously only one defined value in each variable domain that can appear in the prevail-conditions of operators. In order to be able to refer to these values collectively we introduce the term *global prevail condition.*

[12] A slightly simplified version does fit in the SAS$^+$-PUS problem, though, [Bäckström, 1992a; Bäckström, 1992b].

Definition 3.3 Given a single-valued set $\mathcal{O}$ of operators, we define the **global prevail condition** of $\mathcal{O}$ as $\widehat{\mathsf{prv}}(\mathcal{O}) = \sqcup_{o \in \mathcal{O}} \mathsf{prv}(o)$. Similarly, given a plan $\bar{\alpha}$ over $\mathcal{O}$ we define $\widehat{\mathsf{prv}}(\bar{\alpha}) = \sqcup_{o \in \bar{\alpha}} \mathsf{prv}(o)$. □

For instance, the global prevail-condition of the operator set for the cappucino brewer is the partial state $\langle \mathsf{H}, \mathsf{u}, \mathsf{u}, \mathsf{T} \rangle$.

3.2 Overview of Complexity Results

We already know [Bäckström, 1992a, 1992b] that we can generate optimal (in the sense of minimal length) plans for SAS$^+$-PUS instances in polynomial time. Since this problem is a generalization of the previously studied, tractable SAS-PUBS [Bäckström and Klein, 1991b] and SAS-PUS [Bäckström and Klein, 1991a] optimal plan generation problems, it is interesting to ask whether we can generalize even further, staying tractable. Unfortunately, it turns out that we cannot remove any of the three restrictions (P, U and S) and still generate optimal plans tractably.

Figure 2 summarizes the complexity results for optimal plan generation using a lattice defined by the possible combinations of the restrictions in Definition 3.2. It turns out that all complexity results presented in this report hold irrespectively of whether we are studying SAS instances or SAS$^+$ instances. Hence, we do not distinguish the SAS cases from the SAS$^+$ cases in the problem lattices.

When removing the P restriction from the SAS$^+$-PUS problem, optimal plan generation becomes NP-equivalent[13], as follows by Theorem 4.13. Removing any other restriction leads to the situation that the solution cannot be bounded polynomially in the size of the problem instance, *i.e.*, the plan to be generated can have a length exponential in the size of the input (see Theorems 4.4 and 4.5). Here we encounter what Garey and Johnson called the "second source of intractability" [Garey and Johnson, 1979, p. 11], namely, that a problem cannot be solved in polynomial time simply because writing down the solution can take exponential time. Instead of giving up at this point, one may hope to achieve a weak tractability result of the form that plans of "reasonable" size (if they exist) can be generated in time polynomial in the size of the solution. More formally, one wants to generate optimal plans of a prespecified length k in time polynomial in k, *i.e.*, one aims for pseudo-polynomial algorithms [Garey and Johnson, 1979]. Unfortunately, however, even such a weak form of tractability cannot be achieved, as follows from our results concerning the complexity of the bounded plan existence problem, which are summarized in Figure 3.

[13] For the intractability results we 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 $P = NP$. We cannot use the term NP-complete in this context since we consider the search problem (generating a solution) and not the decision problem (whether a solution exists). 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 Turing 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. See Garey and Johnson [1979] or Johnson [1990] for formal details.

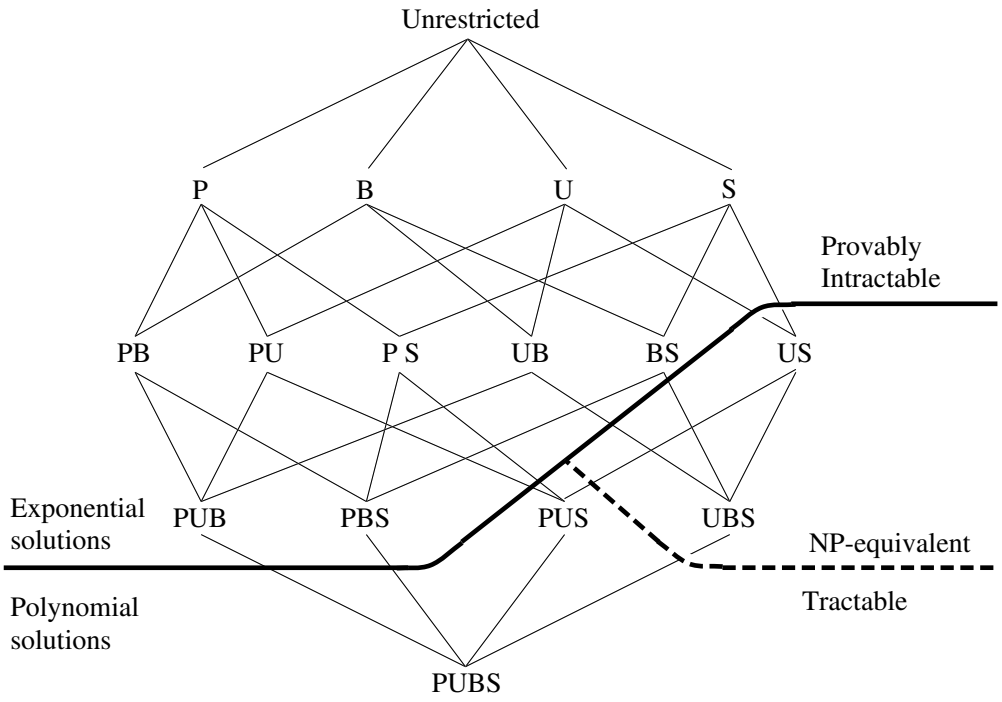

Figure 2: Complexity of optimal plan generation for SAS/SAS$^+$ and restrictions.

Tractability of the SAS$^+$-PUS bounded plan existence problem follows from the above cited results [Bäckström, 1992a, 1992b]. NP-completeness of the SAS-UBS up to SAS$^+$-US is demonstrated by Theorem 4.13, and PSPACE-completeness of SAS-UB and SAS-BS (and all problems above in the lattice) is demonstrated by Corollary 4.10. Finally, Corollary 4.17 demonstrates that SAS-PUB and SAS-PBS bounded plan existence is NP-hard *in the strong sense* [Garey and Johnson, 1979], *i.e.*, even if the runtime is measured in the magnitude of the length parameter (and not in its representation), the problem is NP-hard. This implies that we cannot hope to find a pseudo-polynomial algorithm for the bounded existence problem for the SAS-PUB and SAS-PBS restrictions.

While the SAS$^+$-PUS problem was found to be the maximal tractable problem for optimal plan generation (wrt. the restrictions in Definition 3.2), this is no longer the case if we consider also non-optimal solutions. It turns out that we can find a solution in polynomial time even if we remove the P restriction, *i.e.*, if we have alternative ways of achieving an effect (see Section 5). On the other hand, the intractability results concerning the length of the solution for SAS-PUB and SAS-PBS (Corollary 4.6) hold, of course, also for non-optimal plan generation, so SAS$^+$-US is the unique maximal tractable plan generation problem wrt. the P, U, B and S restrictions. Figure 4 summarizes the complexity results for the plan generation problems.

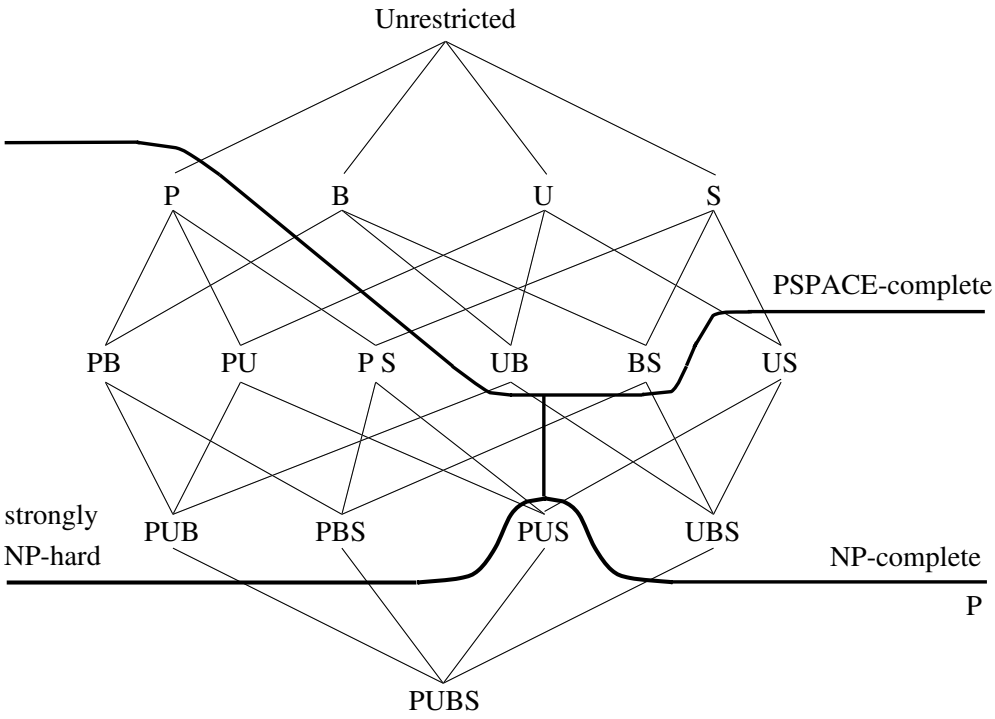

Figure 3: Complexity of bounded plan existence for SAS/SAS$^+$ and restrictions.

Considering the corresponding decision problem, the plan existence problem, the tractability results carry over, of course. Further all the PSPACE-completeness results for bounded plan existence carry over to plan existence. However, we were not able to determine the complexity for the restrictions between SAS-PBS/SAS-PUB and SAS$^+$-P. On one hand, the post-uniqueness restriction is very strong and seems to preclude any attempt to reduce an NP-hard problem to these problems. On the other hand, the fact that the plans can have exponential length seems to preclude a polynomial algorithm or even a nondeterministic polynomial algorithm right from the beginning. It may be the case, however, that it is possible to find proofs for the existence of a plan that are considerably shorter than the plan itself, proofs that may even be generated in polynomial time. However, even if we would be able to decide these problems in polynomial time, this would not help very much in practice because we could not guarantee that a plan is actually generated in polynomial time since the generated plan may have exponential length. Nevertheless, the determination of the complexity of the SAS$^+$-PUS and SAS$^+$-PBS existence problems is an interesting open problem, and an answer might point out the direction for finding new restrictions, which may allow for polynomial time generation algorithms.

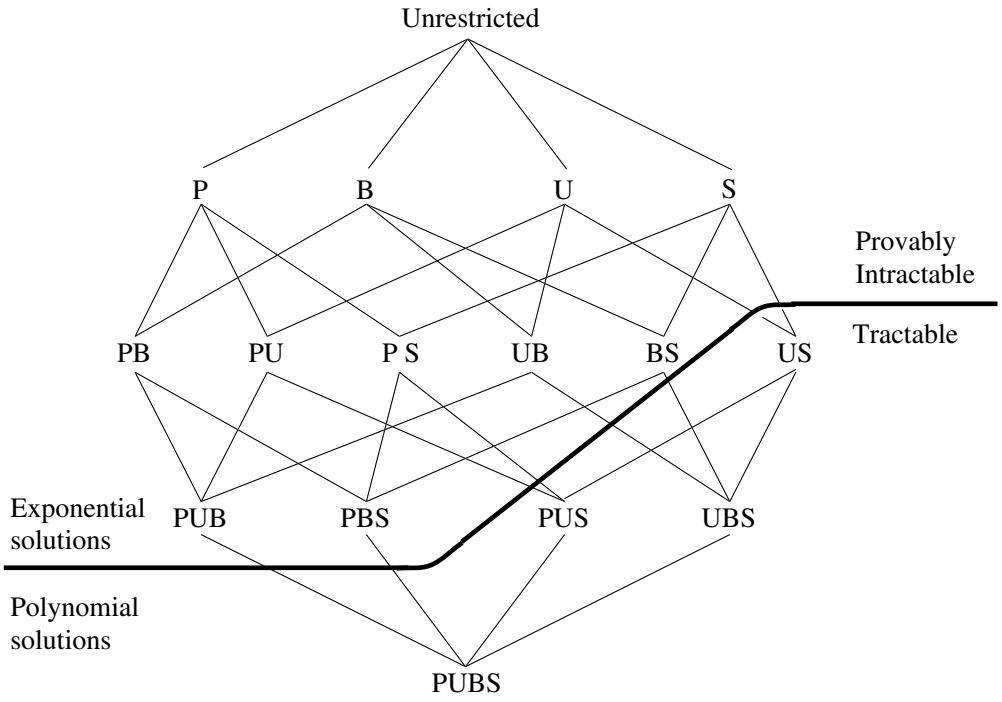

Figure 4: Complexity of plan generation for SAS/SAS$^+$ and restrictions.

4 Computational Complexity of Restricted SAS$^+$ Planning Problems

4.1 Plan Length

First, we show that minimal plans for SAS$^+$-US instances have only polynomial length. For this purpose, we introduce the notion of a *v-chain*, which is a special type of operator sequence we will need in some of the proofs below. Intuitively, a v-chain for some variable $v \in \mathcal{V}$ is a sequence of operators all affecting v and having the property that they form an admissible plan wrt. variable v—but not necessarily if also other variables are taken into account.

Definition 4.1 Given a set $\mathcal{V}$ of variables, a set $\mathcal{O}$ of operators over $\mathcal{V}$, two states $s, t \in {\mathcal{S}_\mathcal{V}}^+$ and a variable $v \in \mathcal{V}$, a plan $\bar{\alpha} = \langle o_1, \ldots, o_n \rangle$ over $\mathcal{O}$ is a **v-chain over $\mathcal{O}$ from s to t** iff either

1. the following conditions hold:
   (a) o_k affects v for $1 \leq k \leq n$,
   (b) $\mathsf{pre}(o_1)[v] \sqsubseteq s[v]$,
   (c) $t[v] \sqsubseteq \mathsf{post}(o_n)[v]$,
   (d) $\mathsf{pre}(o_k)[v] \sqsubseteq \mathsf{post}(o_{k-1})$ for $1 < k \leq n$

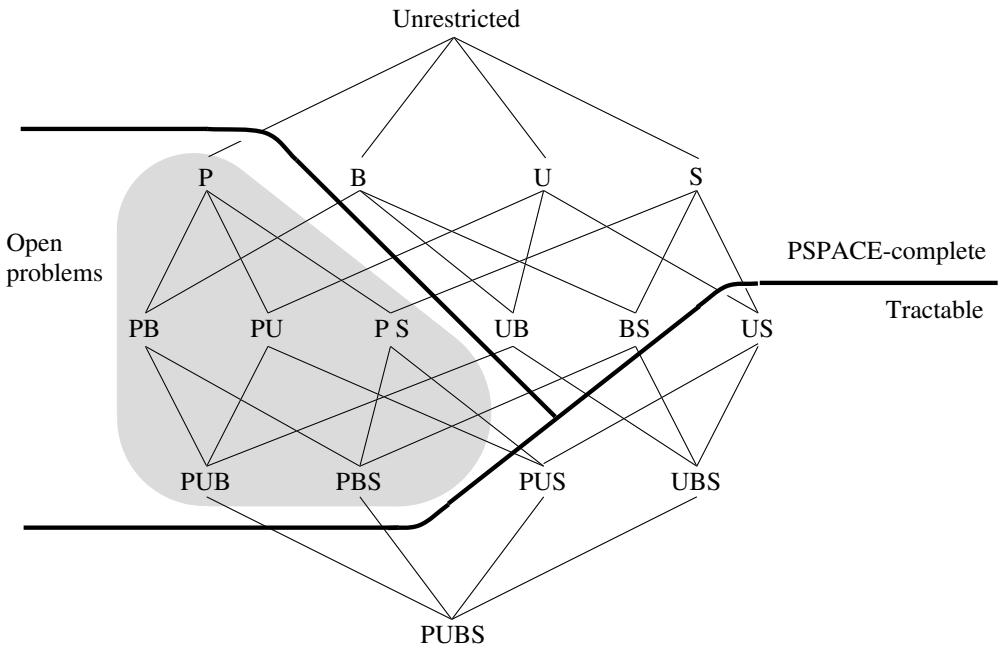

Figure 5: Complexity of plan existence for SAS/SAS$^+$ and restrictions. Open problems are marked by a grey area.

or

2. $\bar{\alpha}$ is the **empty** v**-chain** $\langle\rangle$ and $t[v] \sqsubseteq s[v]$.

A v-chain $\bar{\alpha} = \langle o_1, \ldots, o_n \rangle$ is **loop-free** iff $\mathsf{post}(o_k)[v] \neq \mathsf{post}(o_l)[v]$ for $k \neq l$.
□

Based on this definition, we can prove that minimal SAS$^+$-US plans are of a certain form. For each variable v, the corresponding operators in the minimal plan form a v-chainfrom the initial value to the final value of v. This v-chainis further either loop-free or the concatenation of two loop-free v-chains, passing the global prevail-condition for v in the latter case.

Lemma 4.2 Given an instance $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ of the SAS$^+$-US (optimal) plan generation problem, if $\bar{\alpha}$ is a minimal solution to Π, then for each $v \in \mathcal{V}$,

1. if $\widehat{\mathsf{prv}}(\bar{\alpha})[v] = \mathsf{u}$, then $\bar{\alpha}[v]$ is a loop-free v-chain from s_0 to $s_\star$,
2. if $\widehat{\mathsf{prv}}(\bar{\alpha})[v] \neq \mathsf{u}$, then $\bar{\alpha}[v]$ is of the form $(\bar{\beta};\bar{\gamma})$ where $\bar{\beta}$ is a loop-free v-chain from s_0 to $\widehat{\mathsf{prv}}(\bar{\alpha})[v]$ and $\bar{\gamma}$ is a loop-free v-chain from $\widehat{\mathsf{prv}}(\bar{\alpha})[v]$ to $s_\star$.

□

Proof: Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ be an instance of the SAS$^+$-US (optimal) plan generation problem and suppose $\bar{\alpha} = \langle o_1, \ldots, o_m \rangle$ is a minimal solution to Π.

Let $\bar{\alpha}[v] = \langle o_{i_1}, \ldots, o_{i_n} \rangle$. For each $v \in \mathcal{V}$, there are three cases, which we prove separately.

Case 1: $\widehat{\mathsf{prv}}(\bar{\alpha})[v] = \mathsf{u}$. Suppose $\bar{\alpha}[v]$ is not loop-free, *i.e.* $\mathsf{post}(o_{i_k})[v] = \mathsf{post}(o_{i_l})[v]$ for some $1 \leq k < l \leq n$. Define $\bar{\beta}$ as $\bar{\alpha}$, but lacking the operators $\{o_{i_{k+1}}, \ldots, o_{i_l}\}$ of $\bar{\alpha}[v]$. Obviously, $\bar{\beta}$ is also a plan from s_0 to $s_\star$, since $\widehat{\mathsf{prv}}(\bar{\alpha})[v] = \mathsf{u}$ and $\mathcal{O}$ is unary and, thus, $\mathsf{prv}(o)[v] \sqsubseteq s[v]$ for all $o \in \bar{\alpha}$ and all $s \in {\mathcal{S}_\mathcal{V}}^+$. Furthermore, $|\bar{\beta}| < |\bar{\alpha}|$, which contradicts that $\bar{\alpha}$ is minimal. Hence, $\bar{\alpha}[v]$ must be loop-free.

Case 2: $\widehat{\mathsf{prv}}(\bar{\alpha})[v] \neq \mathsf{u}$ and $s_0[v] \neq \widehat{\mathsf{prv}}(\bar{\alpha})[v]$. Suppose $\mathsf{post}(o_{i_k})[v] = \mathsf{post}(o_{i_l})[v] = \widehat{\mathsf{prv}}(\bar{\alpha})[v]$ for some $1 \leq k < l \leq n$. Wlg. assume $\mathsf{post}(o_j)[v] \neq \widehat{\mathsf{prv}}(\bar{\alpha})[v]$ for $i_k < j < i_l$. Obviously $\mathsf{prv}(o_j)[v] = \mathsf{u}$ for all o_j in $\bar{\alpha}$ s.t. $i_{k+1} < j < i_l$ because of single-valuedness. Hence, $\mathsf{prv}(o_j)[v] \sqsubseteq \mathit{result}(s_0, \bar{\alpha}/i_k)[v]$ for all o_j s.t. $i_k < j < i_{l+1}$. Define $\bar{\beta}$ as $\bar{\alpha}$, but lacking the operators $\{o_{i_{k+1}}, \ldots, o_{i_l}\}$ of $\bar{\alpha}[v]$. Obviously, $\bar{\beta}$ is also a plan from s_0 to $s_\star$. Furthermore, $|\bar{\beta}| < |\bar{\alpha}|$, which contradicts that $\bar{\alpha}$ is minimal and, hence, $\mathsf{post}(o_{i_k})[v] = \mathsf{post}(o_{i_l})[v] = \widehat{\mathsf{prv}}(\bar{\alpha})[v]$ implies $k = l$. Let k be the unique l s.t. $\mathsf{post}(o_{i_l})[v] = \widehat{\mathsf{prv}}(\bar{\alpha})[v]$. The proofs that $\langle o_{i_1}, \ldots, o_{i_k} \rangle$ and $\langle o_{i_{k+1}}, \ldots, o_{i_n} \rangle$ are loop-free are analogous to the proof of case 1 above.

Case 3: $\widehat{\mathsf{prv}}(\bar{\alpha})[v] \neq \mathsf{u}$ and $s_0[v] = \widehat{\mathsf{prv}}(\bar{\alpha})[v]$. Obviously, the empty sequence $\langle\rangle$ is a loop-free v-chain from s_0 to $\widehat{\mathsf{prv}}(\bar{\alpha})[v]$. The proof that $\bar{\alpha}[v]$ is a loop-free v-chain from $\widehat{\mathsf{prv}}(\bar{\alpha})[v]$ to $s_\star$ is analogous to the proof of case 2 above.

Since either of the cases above holds for each $v \in \mathcal{V}$, we conclude that any minimal solution to Π satisfies the theorem. □

It is an immediate consequence of this theorem that the minimal solutions to instances of the SAS$^+$-US (optimal) plan generation problem are of polynomial size.

Theorem 4.3 Any minimal solution to an instance $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ of the SAS$^+$-US (optimal) plan generation problem is of length $O(\sum_{v \in \mathcal{V}} |\mathcal{D}_v|)$. □

Proof: Immediate from Lemma 4.2 since for all $v \in \mathcal{V}$, any loop-free v-chain is of size $O(|\mathcal{D}_v|)$. □

This result together with the two following theorems, which show that SAS-PUB and SAS-PBS instances can lead to exponential plans, establishes the validness of the dividing line between polynomially and exponentially sized solutions in Figures 2 and 4.

Theorem 4.4 For each $m > 0$ there is some solvable instance Π of the SAS-PUB (optimal) plan generation problem s.t. Π is of size $|\Pi| = O(p(m))$ for some polynomial p and all minimal solutions to Π are of length $\Theta(2^m)$. □

Proof: Let $m > 0$. Define an instance $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ of the SAS-PUB (optimal) plan generation problem s.t.

- $\mathcal{V} = \{v_1, \ldots, v_m\}$;
- $\mathcal{D}_v = \{0, 1\}$ for all $v \in \mathcal{V}$;

- $\mathcal{O} = \{o_1^+, o_1^-, \ldots, o_m^+, o_m^-\}$ where for $1 \leq k \leq m$ and $1 \leq i \leq m$,

$$\begin{aligned}
\mathsf{pre}(o_k^+)[v_i] = \mathsf{post}(o_k^-)[v_i] &= \begin{cases} 0 & \text{if } i = k, \\ u & \text{otherwise}, \end{cases} \\
\mathsf{post}(o_k^+)[v_i] = \mathsf{pre}(o_k^-)[v_i] &= \begin{cases} 1 & \text{if } i = k, \\ u & \text{otherwise}, \end{cases} \\
\mathsf{prv}(o_k^+)[v_i] = \mathsf{prv}(o_k^-)[v_i] &= \begin{cases} 0 & \text{if } i < k-1, \\ 1 & \text{if } i = k-1, \\ u & \text{otherwise}; \end{cases}
\end{aligned}$$

- $s_0[v_i] = 0$ for $1 \leq i \leq m$;
- $s_\star[v_i] = \begin{cases} 1 & \text{if } i = m, \\ 0 & \text{otherwise}. \end{cases}$

Obviously Π is of size $|\Pi| = O(m^2)$, which satisfies the first claim in the theorem.

Define the operator sequences $\bar{\alpha}_k^+$ and $\bar{\alpha}_k^-$ recursively s.t. for $1 < k \leq m$,

$$\begin{aligned}
\bar{\alpha}_1^+ &= \langle o_1^+ \rangle, \\
\bar{\alpha}_1^- &= \langle o_1^- \rangle, \\
\bar{\alpha}_k^+ &= \bar{\alpha}_{k-1}^+ ; o_k^+ ; \bar{\alpha}_{k-1}^-, \\
\bar{\alpha}_k^- &= \bar{\alpha}_{k-1}^+ ; o_k^- ; \bar{\alpha}_{k-1}^-.
\end{aligned}$$

It is obvious that the plan $\bar{\alpha}_m^+$ is the unique, minimal solution to Π. Furthermore, $\bar{\alpha}_m^+$ is a Hamilton path in the state space from s_0 to $s_\star$ so $|\bar{\alpha}_m^+| = 2^m - 1$ and the theorem follows. □

If we interpret the states as binary numbers in the proof above, the states are 'visited' in the same order as they would be enumerated by a *Gray code* counter.

Theorem 4.5 For each $m > 0$ there is some solvable instance Π of the SAS-PBS (optimal) plan generation problem s.t. Π is of size $|\Pi| = O(p(m))$ for some polynomial p and all minimal solutions to Π are of length $\Theta(2^m)$. □

Proof: Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ be an instance of the SAS-PUB (optimal) plan generation problem s.t. $|\mathcal{V}| = m$. Wlg. assume $\mathcal{D}_v = \{0, 1\}$ for all $v \in \mathcal{V}$. First define the *inverse* $\overline{x}$ for a value x s.t. for all $v \in \mathcal{V}$ and $x \in \mathcal{D}_v$,

$$\overline{x} = \begin{cases} 0 & \text{if } x = 1, \\ 1 & \text{if } x = 0, \\ \mathsf{u} & \text{otherwise}. \end{cases}$$

Then define the instance $\Pi' = \langle \mathcal{V}', \mathcal{O}', t_0, t_\star \rangle$ of the SAS-PBS plan generation problem s.t.

- $\mathcal{V}' = \{v_0, v_1 \mid v \in \mathcal{V}\}$;

- $\mathcal{D}_{v'} = \{0, 1\}$ for all $v \in \mathcal{V}$;
- $\mathcal{O}'$ contains one operator o' for each $o \in \mathcal{O}$ s.t. for all $v \in \mathcal{V}$,

$$\begin{aligned}
\mathsf{pre}(o')[v_0] &= \overline{\mathsf{pre}(o)[v]}, \\
\mathsf{pre}(o')[v_1] &= \mathsf{pre}(o)[v], \\
\mathsf{post}(o')[v_0] &= \overline{\mathsf{post}(o)[v]}, \\
\mathsf{post}(o')[v_1] &= \mathsf{post}(o)[v], \\
\mathsf{prv}(o')[v_0] &= \begin{cases} 1 & \text{if } \mathsf{prv}(o)[v] = 0, \\ \mathsf{u} & \text{otherwise}, \end{cases} \\
\mathsf{prv}(o')[v_1] &= \begin{cases} 1 & \text{if } \mathsf{prv}(o)[v] = 1, \\ \mathsf{u} & \text{otherwise}; \end{cases}
\end{aligned}$$

- for all $v \in \mathcal{V}$,

$$\begin{aligned}
t_0[v_0] &= \overline{s_0[v]}, \\
t_0[v_1] &= s_0[v], \\
t_\star[v_0] &= \overline{s_\star[v]}, \\
t_\star[v_1] &= s_\star[v].
\end{aligned}$$

Π' is obviously an instance of the SAS-PBS problem, so it remains to prove that Π' is solvable and that its minimal solutions are of exponential size in the size of Π'. We prove this by proving that there is a length-preserving bijection between the solutions for Π' and the solutions for Π.

We first note that for each $v \in \mathcal{V}$ either

$$t_0[v_0] = 1 \text{ and } t_0[v_1] = 0$$

or

$$t_0[v_0] = 0 \text{ and } t_0[v_1] = 1.$$

Furthermore, all operators in $\mathcal{O}'$ are defined such that this holds also for any state resulting from executing a plan over $\mathcal{O}'$ in t_0. We will prove that there is a plan $\bar{\alpha}$ over $\mathcal{O}$ that solves Π iff there is a plan $\bar{\beta}$ over $\mathcal{O}'$ that solves Π'.

$\Rightarrow$: Suppose the plan $\bar{\alpha} = \langle o_1, \ldots, o_n \rangle$ over $\mathcal{O}$ solves Π, *i.e.*, $\bar{\alpha}$ is admissible in s_0 and $\mathit{result}(s_0, \bar{\alpha}) = s_\star$. Let $\bar{\beta} = \langle o'_1, \ldots, o'_n \rangle$ be a plan over $\mathcal{O}'$ s.t. for $1 \leq k \leq n$, o_k and o'_k are corresponding operators, according to the reduction. It is obvious that for $1 \leq k \leq n$ and $v \in \mathcal{V}$,

$$\mathit{result}(t_0, \bar{\beta}/(k-1))[v_0] = \overline{\mathit{result}(s_0, \bar{\alpha}/(k-1))[v]}$$

and

$$\mathit{result}(t_0, \bar{\beta}/(k-1))[v_1] = \mathit{result}(s_0, \bar{\alpha}/(k-1))[v],$$

so

$$\mathsf{prv}(o'_k) \sqsubseteq \mathit{result}(t_0, \bar{\beta}/(k-1)).$$

Hence, $\bar{\beta}$ is admissible in t_0 and $\mathit{result}(t_0, \bar{\beta}) = t_\star$, *i.e.*, $\bar{\beta}$ solves Π'.

$\Rightarrow$: Suppose the plan $\bar{\beta} = \langle o'_1, \ldots, o'_n \rangle$ over $\mathcal{O}'$ solves Π', *i.e.*, $\bar{\beta}$ is admissible in t_0 and $result(t_0, \bar{\beta}) = t_\star$. Obviously, for $1 \leq k \leq n$ and $v \in \mathcal{V}$,

$$result(s_0, \bar{\alpha}/(k-1))[v] = result(t_0, \bar{\beta}/(k-1))[v_1] = \overline{result(t_0, \bar{\beta}/(k-1))[v_0]}$$

so

$$\mathsf{prv}(o_k) \sqsubseteq result(s_0, \bar{\alpha}/(k-1)) \text{ iff } \mathsf{prv}(o'_k) \sqsubseteq result(t_0, \bar{\beta}/(k-1)).$$

Hence, $\bar{\alpha}$ is admissible in s_0 and $result(s_0, \bar{\alpha}) = s_\star$, *i.e.*, $\bar{\alpha}$ solves Π.

It follows that there is a length preserving bijection between the solutions to Π' and the solutions to Π so it further follows from Theorem 4.4 that the minimal plans solving Π' are of length $2^m - 1$. □

The reduction in the above proof actually satisfies the stronger requirements for three-stage polynomial reductions [Bäckström, 1992a, Chapter 5] and exact structure-preserving reductions [Bäckström, 1994a].

An immediate consequence of the previous two theorems is that both optimal and non-optimal SAS$^+$-PUB and SAS$^+$-PBS plan generation is provably intractable since we might have to output a solution which is exponentially larger than the problem instance itself.

Corollary 4.6 The time complexity of SAS-PUB and SAS-PBS (optimal) plan generation has a lower bound of $\Omega(2^{|\mathcal{V}|})$. □

Some care should be taken in interpreting these results, though. What we see here is the second cause for intractability as defined by Garey and Johnson [1979, p. 11], namely, that minimal solutions may have exponential size. Since it seems to be unrealistic to execute such plans in reasonable time, they are of little interest from a practical point of view [Bäckström, 1992a, pp. 142–147].

It may very well be the case that it is possible to find a pseudo-polynomial algorithm or to devise polynomial decision algorithms, *i.e.*, it may be possible to find a proof of existence in polynomial time. Nevertheless, the results indicate that it is provably impossible to generate plans in polynomial time for instances of SAS-PUS and SAS-PBS instances.

4.2 PSPACE-complete Problems

First of all, we prove an upper bound for SAS$^+$ plan existence.

Theorem 4.7 SAS$^+$ (bounded) plan existence is in PSPACE. □

Proof: We can guess a plan one operator at a time and verify it step by step using only polynomial space. Hence, the problem is in NPSPACE and thus also in PSPACE. This also holds if an upper bound on the plan length is specified. □

Bylander [1991] analyzed the complexity of plan existence in a variant of propositional STRIPS we will refer to as the PSN (Propositional STRIPS with

Negative Goals) formalism. Translated to our terminology, PSN is a notational variant of SAS$^+$-B with a *total initial state*. We have earlier shown [Bäckström, 1992a, 1992b, 1994a] that SAS$^+$ (bounded) plan existence is equivalent under polynomial reduction to the (bounded) plan existence problem for PSN and two other variants of the propositional STRIPS and propositional TWEAK formalisms. Since Bylander has proven plan existence PSPACE-complete for unrestricted PSN [Bylander, 1991], PSPACE-completeness for unrestricted SAS$^+$ (bounded) plan existence follows.

Here we will prove the stronger result that even SAS-BS plan existence is PSPACE-hard. This result is very similar to Bylander's Corollary 3.2 [1994, p. 180], which states that plan existence for PSN, where all operators have at most two positive pre-conditions and two arbitrary post-conditions, is PSPACE-complete. However, SAS-BS is not directly comparable with PSN. SAS-BS is on one hand more liberal since the single-valuedness applies only to unchanged state variables. On the hand, SAS-BS is more restrictive because we require that the goal state is *total* and that any state variable appearing in a post-condition has a defined value prior to the execution, while a PSN goal state can be partial and values may be changed from undefined to defined.

Theorem 4.8 SAS-BS (bounded) plan existence is PSPACE-hard. □

Proof: Analyzing the proof of Bylander's Theorem 3.1 [1994, p. 179–180] stating that PSN plan existence is PSPACE-hard, it turns out that the claim can be sharpened to the statement that *SAS$^+$-BS plan existence with each operator having at most 2 pre-conditions, a total initial state and a goal state consisting of only one variable/value pair that is not used in any precondition* is PSPACE-hard.

However, it is straightforward to reduce this problem to a SAS-BS plan existence problem. Assume the state variables p_i and the special goal state variable g, which all have the domain $\{0, 1\}$. The partial goal state requires g to be 1.

Now we construct a SAS-BS instance by using the same initial state, a total goal state with $g = 1$, $p_i = 0$, and the set of operators of the original instance extended by operators that set all variables $p_i = 0$ and have $g = 1$ as their prevail-condition. The resulting operator set does not necessarily satisfy the SAS restriction R4 because some operators may change an undefined value to a defined one. However, since the number of pre-conditions is bounded by a constant, it is straightforward to transform the operator set to one that satisfies R4.

It is obvious that there exists a plan for the new SAS-BS instance if, and only if, there exists a plan for the original instance. Furthermore, the reduction is obviously polynomial. Hence, SAS-BS plan existence is PSPACE-hard. Further, because bounded existence is at least as hard as existence, also the SAS-BS bounded existence problem is PSPACE-hard. □

We can further prove that also bounded and unbounded SAS-UB plan existence is PSPACE-hard, a result similar to Bylander's Theorem 3.3 [1994, p. 180]. Again, the only difference to our result is that we require a total goal state.

Theorem 4.9 Bounded and unbounded SAS-UB plan existence is PSPACE-hard. □

Proof sketch: Analyzing the proof of Bylander's Theorem 3.3, it becomes obvious that an analogous reduction can be applied to the proof of our Theorem 4.8. □

The following corollary is immediate from the above theorems and definitions of the various problems.

Corollary 4.10 Both bounded and unbounded plan existence are PSPACE-complete for SAS^+, SAS-UB, SAS-BS and all problems ordered between these in the lattice. □

4.3 NP-hard Bounded Existence Problems

Bylander [1994, Theorem 4.2] showed that PSN bounded existence is NP-complete if all operators have only positive pre-conditions and one post-condition. From that it follows immediately that SAS^+-UBS bounded plan existence is NP-hard. In order to prove the more restricted bounded existence problem SAS-UBS to be NP-hard, we will use a reduction from the *minimum cover* problem [Garey and Johnson, 1979, page 222], which is defined as follows.

Definition 4.11 An instance of the **minimum cover** problem is given by a set $X = \{x_1, \ldots, x_m\}$, a set $C = \{C_1, \ldots, C_n\}$ of subsets of X and an integer K. The question is whether there exist a cover for X, *i.e.*, a subcollection $C' \subseteq C$ s.t. $\bigcup_{C_k \in C'} C_k = X$ and $|C'| \leq K$. □

Lemma 4.12 Bounded plan existence for SAS-UBS is NP-hard. □

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-UBS instance $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ s.t.

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

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

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

$$\mathsf{prv}(o_k^+)[v] = \mathsf{prv}(o_k^-)[v] = \mathsf{u}$$

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

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

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

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

- $s_0[c_k] = s_\star[c_k] = 0$ for $1 \leq k \leq n$,
- $s_0[x_k] = 0$ and $s_\star[x_k] = 1$ for $1 \leq k \leq m$.

It is obvious that X has a cover C' such that $|C'| \leq K$ iff there is a plan of size $|X| + 2K$ or less solving Π. □

We can then prove that bounded plan existence is NP-complete for SAS-UBS and SAS$^+$-US and all problems 'between' these in the lattice.

Theorem 4.13 Bounded plan existence is NP-complete for SAS-UBS, SAS-US, SAS$^+$-UBS and SAS$^+$-US. □

Proof: NP-hardness for all four problems follows from Lemma 4.12, so it remains to prove that bounded plan existence for SAS$^+$-US is in NP. We know from Theorem 4.3 that minimal solutions are of polynomial size for all four problems. Furthermore, since all operators are unconditional we can validate a proposed solution in polynomial time. It follows that all four problems are NP-complete. □

We further prove *strong* NP-hardness for SAS-PUB bounded existence by reduction from the NP-complete *clique* problem [Garey and Johnson, 1979, page 194], which is defined as follows.

Definition 4.14 An instance of the **clique** problem is given by a graph $G = (V, E)$ and a positive integer $k \leq |V|$. The question is: Does G contain a clique (*i.e.*, a complete subgraph) of size k. □

Theorem 4.15 Bounded SAS-PUB plan existence is NP-hard in the strong sense. □

Proof: Proof by reduction from the clique problem. Given an instance of the clique problem consisting of a graph $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$ and a positive integer $k \leq n$, we construct an instance $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star, p \rangle$ of the bounded SAS-PUB existence problem, where $p = 8n - 2k$.

Let $\mathcal{V} = \{a_i, b_i, c_i, d_i, v_i \mid 1 \leq i \leq n\}$ be the state variables, each with domain $\{0, 1\}$. The initial state s_0 and the goal state $s_\star$ are defined s.t. for each $1 \leq$

op.	Pre-cond.	Post-cond.	Prevail-cond.
o_{a_i}	$\mathsf{pre}(a_i) = 0$	$\mathsf{post}(a_i) = 1$	$\mathsf{prv}(c_i) = 1$ and $\mathsf{prv}(d_i) = 0$
o_{b_i}	$\mathsf{pre}(b_i) = 0$	$\mathsf{post}(b_i) = 1$	$\mathsf{prv}(c_i) = 0$ and $\mathsf{prv}(d_i) = 1$
o_{c_i}	$\mathsf{pre}(c_i) = 0$	$\mathsf{post}(c_i) = 1$	
o'_{c_i}	$\mathsf{pre}(c_i) = 1$	$\mathsf{post}(c_i) = 0$	$\mathsf{prv}(v_i) = 1$
o_{d_i}	$\mathsf{pre}(d_i) = 0$	$\mathsf{post}(d_i) = 1$	
o'_{d_i}	$\mathsf{pre}(d_i) = 1$	$\mathsf{post}(d_i) = 0$	$\mathsf{prv}(w_j) = 1$ for all $(v_i, w_j) \notin E$
o_{v_i}	$\mathsf{pre}(v_i) = 0$	$\mathsf{post}(v_i) = 1$	
o'_{v_i}	$\mathsf{pre}(v_i) = 1$	$\mathsf{post}(v_i) = 0$	

Table 3: Operators used in reduction from *clique* to *bounded SAS-PUB plan existence*. All variables not explicitly stated to have a specific value in a condition are assumed undefined there.

$i \leq n$,

$$\begin{array}{rclrcl} s_0[a_i] & = & 0 & s_\star[a_i] & = & 1 \\ s_0[b_i] & = & 0 & s_\star[b_i] & = & 1 \\ s_0[c_i] & = & 0 & s_\star[c_i] & = & 1 \\ s_0[d_i] & = & 0 & s_\star[d_i] & = & 1 \\ s_0[v_i] & = & 0 & s_\star[v_i] & = & 0 \end{array}$$

Further define

$$\mathcal{O} = \{o_{a_i}, o_{b_i}, o_{c_i}, o'_{c_i}, o_{d_i}, o'_{d_i}, o_{v_i}, o'_{v_i} \mid 1 \leq i \leq n\},$$

with pre-, post- and prevail-conditions as specified in Table 3.

Now we claim that G contains a clique of size k iff Π has a solution of size p or shorter.

"$\Rightarrow$" Assume G contains a clique $C \subseteq V$ of size k. Construct a plan with four phases P_1, P_2, P_3 and P_4. For every node $v_i \in V - C$ include the following subsequence in P_1:

$$\ldots, o_{c_i}, o_{a_i} \ldots$$

For every node $v_i \in C$ include

$$\ldots, o_{d_i}, o_{b_i}, \ldots$$

in P_1. Note that regardless of how these subsequences are put together, the resulting sequence is always executable and leads to some state s s.t. $s[a_i] = 1$ for all $v_i \in V - C$ and $s[b_i] = 1$ for all $v_i \in C$.

Phase P_2 contains the operators o_{v_i} for all nodes $v_i \in V - C$ in any order.

Phase P_3 contains for each node $v_i \in V - C$ the subsequence

$$\ldots o'_{c_i}, o_{d_i}, o_{b_i}, o_{c_i}, \ldots$$

and for each node $v_i \in C$

$$\ldots o'_{d_i}, o_{c_i}, o_{a_i}, o_{d_i}, \ldots$$

These subsequences are put together in any order.

The operator of type o'_{c_i} in the first kind of subsequence is executable, since the pre-condition $\mathsf{prv}[c_i] = 1$ is produced in phase P_1 and the prevail-condition $\mathsf{prv}[v_i] = 1$ is produced in phase P_2. The remaining operators are all obviously executable.

The operator of type o'_{d_i} in the second kind of sequences is executable since the variables w_j that are mentioned in the prevail-condition cannot belong to the set C because of the definition of the operator. Assuming otherwise would lead to the conclusion that C is not a clique, contrary to our assumption. Hence, all defined w_j variables in the prevail-condition of o'_{d_i} must belong to the set $V - C$. The remaining sequence is executable because of the settings in P_1 and P_3.

Summarizing, after phase P_3 we have reached a state s s.t.

$$s[a_i] = s[b_i] = s[c_i] = s[d_i] = 1.$$

Finally, in phase P_4, for every node $v_i \in V - C$, an operator of type o'_{v_i} is executed in order to reset v_i.

Analyzing the constructed plan shows that it indeed leads from the initial state s_0 to the goal state $s_\star$. Further, P_1 is of length $2n$, P_2 is of length $n - k$, P_3 is of length $4n$, and phase P_4 is of length $n - k$. Hence, the overall length p is $8n - 2k$.

"$\Leftarrow$" Assume there exists a plan of length $8n - 2k$ or less. Let $q \leq 8n - 2k$ denote the length of the plan.

Note that for every node $v_i \in V$ one of the following subsequences must be present in the plan:

$$\ldots, o_{c_i}, \ldots, o_{a_i}, \ldots, o'_{c_i}, \ldots, o_{d_i}, \ldots, o_{b_i}, \ldots, o_{c_i}, \ldots$$

or

$$\ldots, o_{d_i}, \ldots, o_{b_i}, \ldots, o'_{d_i}, \ldots, o_{c_i}, \ldots, o_{a_i}, \ldots, o_{d_i}, \ldots$$

It may be the case that o'_{c_i}, o_{c_i} pairs or o'_{d_i}, o_{d_i} pairs appear more than once. In this case, however, it is easy to see that there exists a plan of length $q' \leq q$ that contains only one such pair for every node v_i and that also solves Π.

Furthermore, in order to guarantee the executability of the o'_{c_i} and o'_{d_i} operators and in order to guarantee that finally $v_j = 0$ for all $v_j \in V$, the plan must contain subsequences of the form

$$\ldots, o_{v_j}, \ldots, o'_{v_j}, \ldots$$

for all $v_j \in W$, where $W \subseteq V$. Again, it may be the case that for one node v_j there is more than one such pair. However, in this case there exists a plan of length $q'' \leq q'$ that contains only one such pair for each $v_j \in W$.

Assume that $v_k \in V - W$. This implies that we have the subsequence

$$\ldots, o_{d_k}, \ldots, o_{b_k}, \ldots, o'_{d_k}, \ldots, o_{c_k}, \ldots, o_{a_k}, \ldots, o_{d_k}, \ldots$$

in the plan. The prevail-condition of o'_{d_k} requires that for some $U \subseteq V$ all variables $v_l \in U$ have the value 1. By construction of the operators, U contains

only nodes that are not connected to v_k. Hence, since o'_{d_k} is executable, v_k must be connected to all nodes $V - W$. Since v_k was an arbitrary node of $V - W$, this holds for all nodes in $V - W$, so $V - W$ must be a clique.

Note that the plan we derived from the original plan has a length of $q'' = 6n + 2|W| = 8n - 2|V - W|$. Hence, we have

$$8n - 2|V - W| = q'' \leq q' \leq q \leq 8n - 2k,$$

or

$$|V - W| \geq k.$$

Hence, the existence of a plan of length $q \leq 8n - 2k$ implies the existence of a clique in G of size k.

Finally, since the length parameter q is polynomially bounded by the problem size, the transformation implies strong NP-hardness. □

We can, finally, prove that bounded SAS-PBS plan existence is NP-hard by reduction from the SAS-PUB problem, which we have just proven NP-complete.

Theorem 4.16 Bounded SAS-PBS plan existence is NP-hard in the strong sense. □

Proof: Proof by reduction from bounded SAS-PUB plan existence. Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star, K \rangle$ be an instance of the bounded SAS-PUB plan existence problem. Define the corresponding instance $\Pi' = \langle \mathcal{V}', \mathcal{O}', t_0, t_\star, K \rangle$ of the SAS-PBS bounded plan existence problem as follows. Let

- $\mathcal{V}' = \{v_x \mid v \in \mathcal{V} \text{ and } x \in \mathcal{D}_v\}$,
- $\mathcal{D}_{v'} = \{0, 1\}$ for all $v' \in \mathcal{V}'$.

Further define $\mathcal{O}'$ s.t. for each operator $o \in \mathcal{O}$, there is a corresponding operator $o' \in \mathcal{O}'$ s.t. for all $v_x \in \mathcal{V}'$,

$$\mathsf{pre}(o')[v_x] = \begin{cases} 1 & \text{if } \mathsf{pre}(o)[v] = x, \\ 0 & \text{if } \mathsf{u} \neq \mathsf{pre}(o)[v] \neq x, \\ \mathsf{u} & \text{otherwise}; \end{cases}$$

$$\mathsf{post}(o')[v_x] = \begin{cases} 1 & \text{if } \mathsf{post}(o)[v] = x, \\ 0 & \text{if } \mathsf{u} \neq \mathsf{post}(o)[v] \neq x, \\ \mathsf{u} & \text{otherwise}; \end{cases}$$

$$\mathsf{prv}(o')[v_x] = \begin{cases} 1 & \text{if } \mathsf{prv}(o)[v] = x, \\ \mathsf{u} & \text{otherwise}. \end{cases}$$

Finally, define t_0 and $t_\star$ s.t. for all $v_x \in \mathcal{V}'$,

$$t_0[v_x] = \begin{cases} 1 & \text{if } s_0[v] = x, \\ 0 & \text{otherwise}; \end{cases}$$

$$t_\star[v_x] = \begin{cases} 1 & \text{if } s_\star[v] = x, \\ 0 & \text{otherwise}; \end{cases}$$

It is obvious that a plan $\langle o_1, \ldots, o_n \rangle$ over $\mathcal{O}$ solves Π iff the plan $\langle o'_1, \ldots, o'_n \rangle$ over $\mathcal{O}'$ solves Π'. □

The following corollary is immediate from the above theorem.

Corollary 4.17 Bounded plan existence is NP-hard in the strong sense for SAS-PUB, SAS-PBS and all problems above these in the lattice. □

5 A Polynomial Algorithm for the SAS$^+$-US Generation Problem

In this section we will prove that SAS$^+$-US plan generation is tractable by presenting a polynomial-time algorithm for this problem and proving it correct. The algorithm is a generalization of Bylander's [1994, p. 183–185] algorithm for deciding PSN plan existence with all operators restricted to positive preconditions and only one post-condition.

5.1 Algorithm

We will prove below that the following algorithm solves the non-optimal SAS$^+$-US plan generation problem in polynomial time.

Algorithm 5.1

```
1  AchievePrevail(s, N)
2  ᾱ ← ⟨⟩
3  t ← s
4  loop
5    for v ∈ V − N do
6      Create the directed graph G = ⟨D_v^+, ∅⟩
7      Mark s[v] and u
8      while there is some o ∈ O s.t. pre(o)[v] is marked, post(o)[v] unmarked
       and prv(o) ⊑ t do
9        Mark post(o)[v] in G
10       Insert arc labeled o from pre(o)[v] to post(o)[v] in G
11     if there is a path from s[v] to prv̂[v] in G then
12       t[v] ← prv̂[v]
13       β̄ ← ⟨ the labels along the path ⟩
14       ᾱ ← (ᾱ;β̄)
15 until t not changed
16 return ⟨t, ᾱ⟩
```

```
1  AchieveGoal(s, t)
2  ᾱ ← ⟨⟩
3  s' ← t
4  loop
5    for v ∈ V do
```

```
6       if t[v] ≠ u then
7         Create the directed graph G = ⟨D_v^+, ∅⟩
8         Mark t[v] in G
9         while there is some o ∈ O s.t. pre(o)[v] is unmarked, post(o)[v]
    marked and prv(o) ⊑ s' do
10          Mark pre(o)[v] in G
11          Insert arc labeled o from pre(o)[v] to post(o)[v] in G
12      if t[v] = u then
13        s'[v] ← s[v]
14      elsif there is a path from s[v] to t[v] then
15        s'[v] ← s[v]
16        β̄ ← ⟨ the labels along the path ⟩
17        ᾱ ← (β̄;ᾱ)
18  until s' not changed
19  return ⟨s', ᾱ⟩
```

```
1   Plan(s_0, s_⋆)
2   N ← ∅
3   loop
4     ⟨s, ᾱ⟩ ← AchievePrevail(s_0, N)
5     ⟨t, β̄⟩ ← AchieveGoal(s, s_⋆)
6     if t ⊑ s then
7       return (ᾱ;β̄)
8     elsif there is some v ∈ N s.t. t[v] ⋢ s[v] then
9       reject
10    else
11      N ← N ∪ {v ∈ V | t[v] ⋢ s[v]}
12  end loop
```

□

On an abstract level, the algorithm can be explained as follows. *Plan* repeatedly performs a combined forward and backward search, implemented by *AchievePrevail* and *AchieveGoal* respectively. If these searches meet in some common state, then a solution is found and returned. Otherwise, the difference between the states reached by the forward and backward searches respectively is analyzed. If it can be determined from this analysis that no solution exists, then *Plan* rejects, otherwise the analysis is used to guide and 'focus' further searches. Before presenting the formal correctness proofs for the algorithm, we offer a somewhat more detailed explanation of how it works.

The function *AchievePrevail* takes as arguments a state s and a variable set N. It returns a state t and a plan $\bar{\alpha}$ from s to t such that for all v either $t[v] = s[v]$ or $s[v] = \widehat{\mathsf{prv}}(\mathcal{O})[v]$. More precisely, *AchievePrevail* maximizes the number of variables v s.t. $s[v] = \widehat{\mathsf{prv}}(\mathcal{O})[v]$, subject to the constraints that $t[v'] = s[v']$ for all $v' \in N$ and that there must be a plan from s to t (the plan $\bar{\alpha}$ returned). *AchievePrevail* uses an iterated greedy strategy to achieve $t[v] = \widehat{\mathsf{prv}}(\mathcal{O})[v]$ for successively more and more variables $v \notin N$. The outer loop is repeated until t is not changed anymore and the body of this loop works

as follows. For each $v \notin N$, *AchievePrevail* builds a DAG with the domain $\mathcal{D}_v^+$ as vertex set and with an, initially empty, set of labeled arcs. *AchievePrevail* then marks the vertices $s[v]$ and u and adds arcs incrementally in the following way. If there is an operator o such that $\mathsf{pre}(o)[v]$ is marked but $\mathsf{post}(o)[v]$ is not marked and, further, $\mathsf{prv}(o) \sqsubseteq t$, then an arc labeled o is inserted from $\mathsf{pre}(o)[v]$ to $\mathsf{post}(o)[v]$ and $\mathsf{post}(o)[v]$ is also marked as visited. The graph will eventually become a forest consisting of two trees, rooted in $s[v]$ and u respectively, plus possibly a number of isolated one-vertex trees. It is thus trivial to test if there is a path from either $s[v]$ or u to $\widehat{\mathsf{prv}}(\mathcal{O})[v]$. If such a path exists, then the labels along the path form a v-chain from $s[v]$ to $\widehat{\mathsf{prv}}(\mathcal{O})[v]$ and t and $\bar{\alpha}$ are changed accordingly. Otherwise, there are two cases. Either there is no v-chain at all from $s[v]$ to $\widehat{\mathsf{prv}}(\mathcal{O})[v]$, in which case *Plan* will eventually return with $t[v] = s[v]$. The other case is that there is no v-chain from $s[v]$ to $\widehat{\mathsf{prv}}(\mathcal{O})[v]$ which is admissible in the current t. This means that any such v-chain has some operator requiring a prevail-condition which is not yet achieved in t. However, all operators are unary so no operator affecting v can make any such v-chain admissible since this would require achieving $\widehat{\mathsf{prv}}(\mathcal{O})[v']$ for some $v' \neq v$. Hence, the attempt to achieve $t[v] = \widehat{\mathsf{prv}}(\mathcal{O})[v]$ is postponed until $t[v'] = \widehat{\mathsf{prv}}(\mathcal{O})[v']$ has been achieved for at least one further $v' \neq v$.

Analogously, *AchieveGoal* takes two states s and t as arguments and returns a state s' and a plan $\bar{\beta}$ from s' to t such that $s'[v] = s[v]$ or $s'[v] = t[v]$ for all v. More precisely, s' is the state 'closest' to s in the sense that it is the *unique maximal* state, wrt. the number of variables v such that $s'[v] = s[v]$, from which there is a plan to t. This claim is subject to the further restriction that we only consider plans where all operators o satisfy that $\mathsf{prv}[v] \sqsubseteq s[v]$ for all v. This may seem strange, but *AchieveGoal* will only be called in cases where the argument s is a state returned by *AchievePrevail*. Hence, we can be certain that any plan from s_0 to $s_\star$ will satisfy this restriction (assuming certain properties of N, as will be explained below). *AchieveGoal* uses an iterated greedy strategy analogous to the one used by *AchievePrevail*.

The main procedure, *Plan*, repeatedly calls *AchievePrevail* and *AchieveGoal* until either a plan from s_0 to $s_\star$ is found or we can be sure that no such plan exists. *Plan* is *sound* since whenever it terminates in line 7 returning a plan, then that plan is a plan from s_0 to $s_\star$ because of the way *AchievePrevail* and *AchieveGoal* works. Furthermore, the set N has the property that at any time during the execution of *Plan*, we can be certain that no plan from s_0 to $s_\star$ can pass $\widehat{\mathsf{prv}}(\mathcal{O})[v]$ for any $v \in N$. This is the reason why it is safe to prevent *AchievePrevail* from achieving $\widehat{\mathsf{prv}}(\mathcal{O})[v]$ for all $v \in N$.

This property is trivially true the first time around the loop, since N is initially empty. To see that the property remains to hold during all subsequent turns requires an induction argument. Assuming that it holds before a particular turn of the loop, in which *Plan* does not terminate, all variables v that are added to N in line 11 satisfy that $t[v] \not\sqsubseteq s[v]$. This can be for either of two cases. In the first case $s[v] = \widehat{\mathsf{prv}}(\mathcal{O})[v]$, but there is no plan that is admissible in s that can change variable v from $\widehat{\mathsf{prv}}(\mathcal{O})[v]$ to $s_\star[v]$. Hence, we must not attempt to achieve $s[v] = \widehat{\mathsf{prv}}(\mathcal{O})[v]$ in the next turn of the loop, which is prevented by inserting v into N. In the other case, $s[v] = s_0[v]$, but there is no plan that

is admissible in $s = s_0$ that can change variable v from $s[v] = s_0[v]$ to $s_\star[v]$. This, however, is required by any plan from s_0 to $s_\star$ so no such plan can exist. Adding v to N then makes *Plan* reject in the next turn of the loop. This condition could have been detected already in line 7 by using a more complicated condition, but is done in this way to make the algorithm simpler.

Finally, N grows strictly for each turn of the loop so *Plan* must eventually terminate in line 7 or 9 since N is bounded above by $\mathcal{V}$. Furthermore, if *Plan* rejects in line 9, then $t[v] \not\sqsubseteq s[v]$ for some $v \in N$. It follows from the property of N that there is no plan from s_0 to $s_\star$ passing $\widehat{\mathsf{prv}}(\mathcal{O})[v]$, so it does not matter that *AchievePrevail* was prevented from achieving $s[v] = \widehat{\mathsf{prv}}(\mathcal{O})[v]$. It further follows from the property of N and from how *AchievePrevail* and *AchieveGoal* work that no plan admissible in s can change variable v from $s[v]$ to $s_\star[v]$. Hence, there can be no plan from s_0 to $s_\star$ since $s[v] = s_0[v]$ which together with the termination result above motivates why *Plan* is *complete*.

5.2 Soundness and Completeness of the Algorithm

The formal proofs that Algorithm 5.1 is a sound and complete, polynomial-time algorithm for the non-optimal SAS$^+$-US plan generation problem follow below.

Definition 5.1 Given two states $s, t \in \mathcal{S}_{\mathcal{V}}{}^+$ and a variable set $V \subseteq \mathcal{V}$, we say that s **agrees with** t **for** V iff $s[v] = t[v]$ for all $v \in V$, and we say that s **agrees with** t **exactly for** V iff $s[v] = t[v]$ for all $v \in V$ and $s[v] \neq t[v]$ for $v \notin V$. We further define the state $s|_t^V$ s.t. for all $v \in \mathcal{V}$,

$$s|_t^V[v] = \begin{cases} t[v] & \text{if } v \in V, \\ s[v] & \text{otherwise}. \end{cases}$$

□

In order to make the following proofs easier to read we will tacitly assume we are solving an instance $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_\star \rangle$ of the SAS$^+$-US plan generation problem. For instance, it will be implicitly assumed that all plans are plans over $\mathcal{O}$. We will further assume that *Plan* is called with the parameters s_0 and $s_\star$. Finally, we will use $\widehat{\mathsf{prv}}$ as a shorthand for $\widehat{\mathsf{prv}}(\mathcal{O})$.

Lemma 5.2 When called with a state $s \in \mathcal{S}_{\mathcal{V}}{}^+$ and a variable set $N \subseteq \mathcal{V}$, *AchievePrevail* returns a tuple $\langle t, \bar{\alpha} \rangle$ s.t.

1. $\bar{\alpha}$ is a plan from s to t,
2. $t = s|^V_{\widehat{\mathsf{prv}}}$ for some $V \subseteq \mathcal{V} - N$ s.t.
3. for all $V' \subseteq \mathcal{V} - N$, if there exists a plan from s to some state s' passing $\widehat{\mathsf{prv}}[v]$ for all $v \in V'$, then $V' \subseteq V$.

□

Proof: Assume *AchievePrevail* is called with the parameters s and N, and that it returns the tuple $\langle t, \bar{\alpha} \rangle$. Claims 1 and 2 are obvious from the algorithm, so let the variable set V be defined s.t. $t = s|^V_{\widehat{\mathsf{prv}}}$. It remains to prove claim 3, *i.e.* for all $V' \subseteq \mathcal{V} - N$, if there exists a plan from s to some state s' passing $\widehat{\mathsf{prv}}[v]$ for all $v \in V'$, then $V' \subseteq V$. The proof is trivial for those $v \in \mathcal{V}$ s.t. $s_0[v] = \widehat{\mathsf{prv}}[v]$, so we assume wlg. that $s_0[v] \neq \widehat{\mathsf{prv}}[v]$ for all $v \in \mathcal{V}$. Proof by induction over the size of V'.

Basis: The claim holds trivially for the empty set.

Induction: Suppose claim 3 holds for all variable sets $S \subseteq \mathcal{V} - N$ of size n, where $0 \leq n < |\mathcal{V}|$. Further suppose $\bar{\beta}$ is a plan from s to some state s' passing $\widehat{\mathsf{prv}}[v]$ exactly for $v \in T$ for some variable set $T \subseteq \mathcal{V} - N$ of size $n+1$. We are only concerned with those v for which $\bar{\beta}$ passes $\widehat{\mathsf{prv}}[v]$; the actual final state s' is not important for claim 3. Hence, we can wlg. assume that $s' = s|^T_{\widehat{\mathsf{prv}}}$ and that $\bar{\beta}$ is a minimal plan from s to s'. Note that $\bar{\beta} \neq \langle\rangle$ since $T \neq \emptyset$.

Furthermore, there must be some variable $v' \in T$ s.t. $\mathsf{prv}(o)[v'] = \mathsf{u}$ for all $o \in \bar{\beta}$, since $\bar{\beta}$ could otherwise not be admissible. Obviously, $\bar{\beta}[v']$ is a v'-chain from s_0 to $\widehat{\mathsf{prv}}[v']$. We define $U = T - \{v'\}$ and let $\bar{\gamma}$ be defined as $\bar{\beta}$ but lacking the operators in $\bar{\beta}[v']$. It is immediate that $\bar{\gamma}$ is a plan from s to $s|^U_{\widehat{\mathsf{prv}}}$ since $\bar{\beta}$ is a plan from s to $s|^T_{\widehat{\mathsf{prv}}}$ and $\mathsf{prv}(o)[v'] = \mathsf{u}$ for all $o \in \bar{\beta}$. It is further obvious that $\bar{\beta}[v']$ is admissible in $s|^U_{\widehat{\mathsf{prv}}}$ since $\mathcal{O}$ is single-valued and $\bar{\beta}$ is admissible in s. Hence, $(\bar{\gamma};\bar{\beta}[v'])$ is a plan from s to $s' = s|^T_{\widehat{\mathsf{prv}}} = s|^{U \cup \{v'\}}_{\widehat{\mathsf{prv}}}$ so it follows from the induction hypothesis that $U \subseteq V$ and it, thus, remains to prove that also $v' \in V$.

Suppose to the contrary that $v' \notin V$. The v'-chain $\bar{\beta}[v']$ must then be admissible in $t = s|^V_{\widehat{\mathsf{prv}}}$ since it is admissible in $s' = s|^U_{\widehat{\mathsf{prv}}}$ and $U \subseteq V$. Hence, *AchievePrevail* could not have terminated returning $\langle t, \bar{\alpha} \rangle$ but would rather have found that $\bar{\beta}[v']$ is admissible in t, thus returning a tuple $\langle t', \bar{\delta} \rangle$ s.t. $t' = s|^{V \cup \{v'\}}_{\widehat{\mathsf{prv}}}$. This contradicts the assumption that *AchievePrevail* returns $\langle t, \bar{\alpha} \rangle$, so it must be the case that $v' \in V$. It follows that $T \subseteq V$, which ends the induction.

We conclude from the induction proof that also claim 3 of the theorem is satisfied. □

Similarly, it can be proven that *AchieveGoal*, when called with two states s and t, returns a state s' and a plan leading from this state to t. Furthermore, s' is the state closest to s from which the goal can be reached, where closest means that no other state from which t can be reached agrees with s on more state variables (and not on any others either). This claim is qualified with the assumption that all operators in the plans considered must have their prevail-conditions satisfied in s.

Lemma 5.3 When called with two states $s, t \in \mathcal{S}_{\mathcal{V}}{}^+$ *AchieveGoal* returns a tuple $\langle s', \bar{\alpha} \rangle$ s.t.

1. $\bar{\alpha}$ is a plan from s' to t,
2. $s' = t|^V_s$ for some $V \subseteq \mathcal{V}$,

3. for all $V' \subseteq \{v \in \mathcal{V} \mid \widehat{\mathsf{prv}}[v] \neq \mathsf{u}\}$ if there exists a plan $\bar{\gamma}$ from some state t' to t s.t. t' agrees with s for V' and $\widehat{\mathsf{prv}}(\bar{\gamma}) \sqsubseteq s$, then $V' \subseteq V$ (*i.e.* for all $v \in \mathcal{V}$, if $t'[v] = s[v]$, then $t[v] = s[v]$).

□

Proof: Analogous to the proof of Lemma 5.2. The main difference to note is that if $t[v] = \mathsf{u}$, then $t[v] \sqsubseteq s[v]$, so $\bar{\beta}[v] = \langle\rangle$ in this case. □

Lemma 5.4 At any time during the execution of *Plan* the set N satisfies that if $v \in N$, then there is no plan from s_0 to $s_\star$ passing $\widehat{\mathsf{prv}}[v]$. □

Proof: Let N_n denote the value of the variable N immediately before the nth turn of the loop. Proof by induction over the number of turns of the loop in *Plan*.

Basis: The claim holds vacuously before the first turn of the loop since $N_1 = \emptyset$.

Induction: Suppose the claim holds for N_k for some $k \geq 1$ and that *Plan* does not terminate in the kth turn of the loop. Prove that then also N_{k+1} satisfies the claim. Assume that in the kth turn of the loop *AchievePrevail* returns $\langle s, \bar{\alpha}\rangle$ and *AchieveGoal* returns $\langle t, \bar{\beta}\rangle$.

We prove for each $v \in \mathcal{V}$ that if $t[v] \not\sqsubseteq s[v]$, then either there exists no plan from s_0 to $s_\star$ passing $\widehat{\mathsf{prv}}[v]$ or there exists no plan at all from s_0 to $s_\star$.

We already know from the induction hypothesis that no plan from s_0 to $s_\star$ can pass $\widehat{\mathsf{prv}}[v]$ for any $v \in N_k$ so, by clause 3 of Lemma 5.2, no plan from s_0 to $s_\star$ can pass $\widehat{\mathsf{prv}}[v]$ for any $v \in \mathcal{V}$ s.t. $s[v] \neq \widehat{\mathsf{prv}}[v]$. Let $v \in \mathcal{V}$ be an arbitrary variable s.t. $t[v] \not\sqsubseteq s[v]$. It is then immediate form the algorithm that there are two mutually exclusive cases, either $s[v] = \widehat{\mathsf{prv}}[v] \neq s_0[v]$ or $s[v] = s_0[v]$ (and possibly $\widehat{\mathsf{prv}}[v] = s_0[v]$).

First suppose that $s[v] = \widehat{\mathsf{prv}}[v] \neq s_0[v]$. It follows from the induction hypothesis and clause 3 of Lemma 5.2 that any plan $\bar{\gamma}$ from s_0 to $s_\star$ must satisfy that $\widehat{\mathsf{prv}}(\bar{\gamma}) \sqsubseteq s$. Hence, no plan from s_0 to $s_\star$ can pass a state s' s.t. $s'[v] = \widehat{\mathsf{prv}}[v] = s[v]$, since Lemma 5.3 says that this would imply $t[v] = s[v]$, thus contradicting the assumption that $t[v] \neq s[v]$. Putting v into N_{k+1} is thus safe and will prevent *Plan* from trying to achieve $\widehat{\mathsf{prv}}[v]$ in the next turn.

Instead suppose that $s[v] = s_0[v]$. Further suppose there exists a plan $\bar{\gamma}$ from s_0 to $s_\star$. Obviously, $\bar{\gamma}$ passes some state s' s.t. $s'[v] = s[v]$ since $s[v] = s_0[v]$. Hence, there exists a plan from s' to $s_\star$. However, clause 3 of Lemma 5.3 then implies that $t[v] = s[v]$, which contradicts that $t[v] \neq s[v]$, *i.e.* that *Plan* rejects. Hence, no plan from s_0 to $s_\star$ can exist and it is thus safe to put v into N_{k+1} (which will cause *Plan* to reject in the next turn).

This ends the induction so we are allowed to conclude that the lemma holds.

□

Lemma 5.5 The set N grows strictly for each turn of the loop in *Plan*. □

Proof: Let N_n denote the value of N immediately before the nth turn of the loop in *Plan*. We must prove that for any $k > 1$, if *Plan* does not terminate

in the k first turns, then $N_{k-1} \subset N_k$. This is obvious, however, since the non-termination of *Plan* guarantees that $t[v] \not\sqsubseteq s[v]$ for some $v \notin N_{k-1}$, which will thus be included in N_k. □

Lemma 5.6 If *Plan* rejects, then there is no plan from s_0 to $s_\star$. □

Proof: Assume that in some turn of the loop *AchievePrevail* returns $\langle s, \bar{\alpha} \rangle$, *AchieveGoal* returns $\langle t, \bar{\beta} \rangle$ and *Plan* rejects. There must then be some $v \in N$ s.t. $t[v] \not\sqsubseteq s[v]$ since *Plan* rejects.

Suppose there exists a plan $\bar{\gamma}$ from s_0 to $s_\star$. Since $v \in N$ it is immediate from the algorithm that $s[v] = s_0[v]$. Hence, $\bar{\gamma}$ is a plan from some state s' s.t. $s'[v] = s[v]$ to $s_\star$. It thus follows from clause 3 of Lemma 5.3 that $t[v] = s[v]$ This, however, contradicts the assumption that $t[v] \not\sqsubseteq s[v]$, *i.e.* that *Plan* rejects so there cannot exist any plan from s_0 to $s_\star$. □

Theorem 5.7 If there exists a plan from s_0 to $s_\star$, then *Plan* returns a plan from s_0 to $s_\star$, and otherwise *Plan* rejects. □

Proof: We first observe that *Plan* must eventually terminate, *i.e.* either return a plan or reject since $N \subseteq \mathcal{V}$ and Lemma 5.5 says that N grows strictly.

We first prove that if there is a plan from s_0 to $s_\star$, then *Plan* returns such a plan. The converse of Lemma 5.6 says that if there is a plan from s_0 to $s_\star$, then *Plan* will not reject. Hence, *AchievePrevail* and *AchieveGoal* will eventually return two tuples $\langle s, \bar{\alpha} \rangle$ and $\langle t, \bar{\beta} \rangle$ respectively s.t. $t \sqsubseteq s$. It is obvious from Lemmata 5.2 and 5.3 that the plan $(\bar{\alpha};\bar{\beta})$ returned by *Plan* is a plan from s_0 to $s_\star$.

It remains to prove that *Plan* will reject if there is no plan from s_0 to $s_\star$. Assume there is no such plan. Further suppose *Plan* does not reject. We know from the first half of this proof that *Plan* must then eventually return a plan from s_0 to $s_\star$. However, this contradicts that there is no such plan so *Plan* must eventually reject. □

5.3 Time Complexity of the Algorithm

It is obvious that *Plan* runs in low-order polynomial time. Below we provide an upper bound on the runtime needed by this algorithm.

Theorem 5.8 *Plan* runs in $O(|\mathcal{V}|^4|\mathcal{O}|)$ time. □

Proof: We first analyze *AchievePrevail* and immediately observe that for each $v \in \mathcal{V}$, $t[v]$ is changed at most once, from $s[v]$ to $\widehat{\mathsf{prv}}[v]$, so the outer loop makes at most $|\mathcal{V}|$ turns. The for loop obviously makes $O(|\mathcal{V}|)$ turns. We assume the graphs created in line 6 are represented using adjacency lists, thus taking $O(|\mathcal{D}_v|)$ time to initialize. Further, we observe that each graph created is a forest consisting of two trees rooted in $s[v]$ and u respectively (or one tree plus possibly a number of isolated one-vertex trees if $s[v] = \mathsf{u}$. Hence it takes $O(|\mathcal{D}_v|)$ time to find a path in such a graph. The while loop takes $O(|\mathcal{O}||\mathcal{V}|)$ time. Assuming $|\mathcal{D}_v| \in O(|\mathcal{O}||\mathcal{V}|)$, which any reasonable problem encoding can

be expected to satisfy, the while loop dominates the body of the for loop. Hence, *AchievePrevail* takes $O(|\mathcal{V}|^3|\mathcal{O}|)$ time. Obviously, *AchieveGoal* is of the same complexity as *AchievePrevail*.

Then analyzing *Plan*, we observe that the loop makes $O(|\mathcal{V}|)$ turns, since $N \subseteq \mathcal{V}$ and Lemma 5.4 tells us that N grows strictly. The loop body is dominated by the calls to *AchievePrevail* and *AchieveGoal* so *Plan* runs in $O(|\mathcal{V}|^4|\mathcal{O}|)$ time. □

The algorithm *Plan* was developed only to prove tractability of SAS$^+$-US plan generation, so no attempts has been made at optimizing it. An obvious modification, however, would be to let the test in the while loops of *AchievePrevail* and *AchieveGoal* search only $\mathcal{O}[v]$ instead of the whole $\mathcal{O}$. This could easily be achieved by storing each $\mathcal{O}[v]$ separately, using negligible $O(\mathcal{O})$ preprocessing time.

6 Discussion and Conclusions

Recently a number of results have been published on the computational complexity of propositional STRIPS planning under various restrictions [Bylander, 1991, 1992a, Erol *et al.*, 1992]. In addition to these we have previously presented a number of tractable planning problems using the SAS$^+$ formalism [Bäckström and Klein, 1991a, 1991b, Bäckström, 1992a, 1992b]. All of these results concern the complexity of planning in various restricted versions of certain formalisms. One might also investigate the complexity of planning for specific problems instead of specific formalisms. This has been done for some variants of the blocks-world problem [Gupta and Nau, 1992; Bäckström, 1992a; Bäckström, 1992b]. Furthermore, the complexity of temporal projection and its relationship to planning has been investigated by Dean and Boddy [1988] and by ourselves [Nebel and Bäckström, 1994].

Our previous publications on SAS$^+$ planning have concentrated on finding tractable subproblems of the optimal plan generation problem and trying to extend these while retaining tractability. This report complements these results by presenting a complete investigation of the complexity for each of the possible combinations of the previously considered restrictions. We already know [Bäckström, 1992a, 1992b] that the SAS$^+$ formalism is expressively equivalent to a number of 'standard' propositional STRIPS formalisms, including those analyzed by Bylander [1991] and Erol *et al.* [1992]. One might wonder, then, why we have bothered doing such a complexity analysis for the SAS$^+$ formalism since the equivalence implies that the previously known complexity results carry over. There are, however, at least two important differences between our analysis and the previous ones.

Firstly, Bylander and Erol *et al.* have only studied local restrictions on operators. In particular, they have studied restrictions that appear obvious to try from the way STRIPS operators are defined, like restricting the number of pre-conditions or disallowing delete-conditions. Binariness and unariness are such a local restrictions, while post-uniqueness and single-valuedness are global restrictions on the whole set of operators. Korf [1987] has also studied some

computationally interesting global properties, like independent and serializable subgoals. Unfortunately, finding out whether a problem instance has serializable subgoals is PSPACE-complete [Bylander, 1992b]. In contrast to this, all our restrictions can be tested in low-order polynomial time [Bäckström, 1992a, Theorem 4.8]. Furthermore, we have not derived our restrictions from the formalism *per se* but from studying a test example in the area of automatic control. Using the SAS^+ formalism instead of the STRIPS formalism has facilitated in finding these restrictions, which are interesting from a computational point of view by resulting in tractability. Furthermore, the equivalence result applies to the unrestricted versions of the formalisms and does not hold under arbitrary restrictions.

The second difference is that the previous analyses of planning complexity have only considered the *plan existence problem* [Bylander, 1991], and sometimes also the *bounded plan existence problem* [Bylander, 1994, Erol *et al.*, 1992]. In addition to this, we also analyze the complexity of actually *generating* a (possibly optimal) plan, which is what we are ultimately interested in.

The result of our analysis shows that we have reached the *tractability borderline* and we cannot continue to remove restrictions and still have tractability. The most surprising result for us was that post-uniqueness of operators, which appears to be a very strong restriction, does not guarantee tractability if considered in isolation. In case of non-optimal planning, the solution size is prohibitive. In case of optimal planning, even pseudo-polynomial solutions are ruled out. This should not discourage us, however. It means that we have will have to start considering alternative restrictions or combinations of less restricted variants of the P, U, B and S restrictions. The proposed research methodology is further described in Bäckström [1992a] where also some suggestions for such alternative restrictions are given. There are several possible ways to continue. For instance, we have recently started investigating more complex restrictions, based on the structure of the state-transition graph rather than simple syntactic properties of the set of operators [Jonsson and Bäckström, 1994b, 1994a]. Another way to go is to extend the tractability borderline by sacrificing completeness, an approach taken by Klein [1993].

Acknowledgments

Bart Selman and the anonymous referees have provided helpful comments on earlier versions of this paper.

References

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

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

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

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

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

[Bäckström and Nebel, 1992] Christer Bäckström and Bernhard Nebel. On the computational complexity of planning and story understanding. In Bernd Neumann, editor, *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI-92)*, pages 349–353, Vienna, Austria, August 1992. Wiley.

[Bäckström, 1988a] Christer Bäckström. Reasoning about interdependent actions. Licentiate Thesis 139, Department of Computer and Information Science, Linköping University, Linköping, Sweden, June 1988.

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

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

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

[Bäckström, 1993] Christer Bäckström. Finding least constrained plans and optimal parallel executions is harder than we thought. In Christer Bäckström and Erik Sandewall, editors, *Current Trends in AI Planning: EWSP'93—2nd European Workshop on Planning*, pages 46–59, Vadstena, Sweden, December 1993. IOS Press.

[Bäckström, 1994a] Christer Bäckström. Expressive equivalence of planning formalisms. To appear in *Artificial Intelligence*, special issue on planning and scheduling.

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

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

[Bylander, 1992a] Tom Bylander. Complexity results for extended planning. In Hendler [1992], pages 20–27.

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

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

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

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

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

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

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

[Gupta and Nau, 1992] Naresh Gupta and Dana S Nau. On the complexity of blocks-world planning. *Artificial Intelligence*, 56:223–254, 1992.

[Hanks and Weld, 1992] Steven Hanks and Daniel S. Weld. Systematic Adaptation for Case-Based Planning. In Hendler [1992], pages 96–105.

[Hendler, 1992] James Hendler, editor. *Artificial Intelligence Planning Systems: Proceedings of the 1st International Conference*, College Park, MD, USA, June 1992. Morgan Kaufmann.

[Johnson, 1990] David S Johnson. A catalog of complexity classes. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science: Algorithms and Complexity*, volume A, chapter 2, pages 67–161. Elsevier, Amsterdam, 1990.

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

[Jonsson and Bäckström, 1994b] Peter Jonsson and Christer Bäckström. Tractable planning with state variables by exploiting structural restrictions. In AAAI-94 [1994].

[Kambhampati and Hendler, 1992] Subbarao Kambhampati and James A. Hendler. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55:193–258, 1992.

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

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

[Nebel and Bäckström, 1994] Bernhard Nebel and Christer Bäckström. On the computational complexity of temporal projection, planning and plan validation. *Artificial Intelligence*, 66(1):125–160, 1994. ARTINT 1063.

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

[Sandewall and Rönnquist, 1986] Erik Sandewall and Ralph Rönnquist. A representation of action structures. In *Proceedings of the 5th (US) National Conference on Artificial Intelligence (AAAI-86)*, pages 89–97, Philadelphia, PA, USA, August 1986. American Association for Artificial Intelligence, Morgan Kaufmann.