To be presented at the workshop on *Comparative Analysis of AI Planning Systems* at the *12th (US) National Conference on Artificial Intelligence (AAAI'94)*, Seattle, WA, USA, Jul.–Aug., 1994.

Planning using Transformation between Equivalent Formalisms: A Case Study of Efficiency

Christer Bäckström
Department of Computer and Information Science
Linköping University, S-581 83 Linköping, Sweden
email: cba@ida.liu.se

Abstract

We have considered two planning formalisms which are known to be expressively equivalent—the CPS formalism, using propositional atoms, and the SAS^+ formalism, using multi-valued state variables. As a case study, we have modified a well-known partial-order planner for CPS into an 'identical' planner for SAS^+ and we have considered two encodings of SAS^+ into CPS. It turns out that it is more efficient to solve SAS^+ instances by using the SAS^+ planner directly than to first encode them as CPS instances and use the CPS planner. For one encoding of SAS^+ into CPS, which is a polynomial reduction, the CPS planner has a polynomially or exponentially larger search space, depending on goal selection strategy. For the other encoding, which is not a polynomial reduction, the CPS search is of the same size as the SAS^+ search space, but the cost per node can be exponentially higher in this case. On the other hand, solving CPS instances by encoding them as SAS^+ instances and using the SAS^+ version of the planners does not suffer from such problems.

1 Introduction

We have previously [1, 2, 3] analyzed four formalisms for propositional planning: two variants of propositional STRIPS (CPS and PSN), ground TWEAK and the SAS^+ formalism—the three latter formalisms adding negative goals, uncertain initial states and multi-valued state variables respectively to CPS. All these features are summarized in Table 1.

These four formalisms seem to form a sequence of successively increasing expressive power, in the order presented. However, we have shown [1, 2, 3] that in fact, all four formalisms are equally expressive under polynomial reduction[1] which is theoretically appealing since this makes a number of theoretical results, especially complexity results, carry over between the formalisms. For instance,most standard complexity classes are closed under polynomial reduction.

The result also tells us that we can model an application as a planning instance in one of the formalisms and solve this by using the polynomial

[1] See [6] or [8] for details on polynomial reductions.

	CPS	PSN	GT	SAS^+
Partial goals	•	•	•	•
Negative pre-conditions		•	•	•
Negative goals		•	•	•
Partial initial states			•	•
Multi-valued state variables				•

Table 1: A comparison of the CPS, PSN, GT and SAS^+ formalisms.

reduction to find an equivalent instance in another formalism and then use a planner for this formalism. Unfortunately, this does not guarantee that we do not face a harder problem than if solving the instance directly in the first formalism—polynomial reducibility is too coarse for for such guarantees. In theory, it may be exponentially harder to solve an equivalent instance in another formalism. However, whether this exponential blow-up arises in practice is not obvious—it depends on which planners are used *etc.*

This paper presents a case study, using reduction from SAS^+ to CPS and two essentially identical planners for the two formalisms. We show that for these planners it may, in fact, be exponentially harder to solve an equivalent CPS instance than to solve the original SAS^+ instance, depending on certain parameters and which reduction is used.

We know of no similar analysis comparing different formalisms, but some comparisons of various planners and search techniques for one fixed formalism can be found in the literature [9, 11].

The remainder of this paper is organized as follows. Section 2 recapitulates the CPS and SAS^+ formalisms. Section 3 recapitulates McAllester and Rosenblitt's [10] planner for CPS and shows how this can be modified to plan for SAS^+. Section 4 defines two reductions from SAS^+ to CPS, one polynomial and one exponential and in Section 5 the complexity of planning in the SAS^+ formalism directly is compared with planning for equivalent CPS instances under both reductions.

2 Two Planning Formalisms

The two formalism considered in this paper are Classical Propositional STRIPS (CPS) and SAS^+. The CPS formalism should be well known to most readers and for further detail on the SAS^+ formalism, the reader is referred to previous publications [2, 4].

Definition 2.1 *An instance of the* **CPS planning problem** *is given by a tuple* $\Pi = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$*, where*

- $\mathcal{P}$ *is a finite set of* **atoms***;*
- $\mathcal{O}$ *is a finite set of* **operators** *of the form* $\langle \varphi, \alpha, \delta \rangle$*, subject to the restriction that* $\alpha \cap \delta = \emptyset$*, where* $\varphi, \alpha, \delta \subseteq \mathcal{P}$ *denote the* **precondition***,*

addlist *and* **deletelist** *respectively;*

- $\mathcal{I} \subseteq \mathcal{P}$ *and* $\mathcal{G} \subseteq \mathcal{P}$ *denote the* **initial state** *and* **goal state** *respectively.*

If $o = \langle \varphi, \alpha, \delta \rangle \subseteq \mathcal{O}$*, we write* $\varphi(o)$*,* $\alpha(o)$ *and* $\delta(o)$ *to denote* φ*,* α *and* δ *respectively. Let* $\mathcal{O}^*$ *denote the set of all sequences of operators from* $\mathcal{O}$*. The members of* $\mathcal{O}^*$ *are called* **plans** *over* Π*. The ternary relation* $valid \subseteq \mathcal{O}^* \times \mathcal{I} \times \mathcal{G}$ *is defined recursively s.t. for arbitrary operator sequence* $\langle o_1, \ldots, o_n \rangle \in \mathcal{O}^*$ *and arbitrary states* $S, T \subseteq \mathcal{P}$*,* $valid(\langle o_1, \ldots, o_n \rangle, S, T)$ *iff either*

1. $n = 0$ *and* $T \subseteq S$ *or*
2. $n > 0$*,* $\varphi(o_1) \subseteq S$ *and* $valid(\langle o_2, \ldots, o_n \rangle, (S \cup \alpha(o_1) - \delta(o_1)), T)$*.*

A plan $\langle o_1, \ldots, o_n \rangle \in \mathcal{O}^*$ **solves** Π *iff* $valid(\langle o_1, \ldots, o_n \rangle, \mathcal{I}, \mathcal{G})$*.*

Instead of propositional atoms, the SAS$^+$ formalism uses multi-valued state variables, which may additionally have the value undefined. A value assignment to the state variables is a (partial) state. Further, an operator is modelled using a pre-, a post- and a prevail-condition. Any variable being defined in the postcondition will get this defined value after executing the operator and the others will remain unchanged. A variable which is required to have some specific value before executing the operator should be specified to have this value either in the precondition, if the variable is changed by the operator, or in the prevailcondition, if it is not changed. That is, the pre- and prevailconditions together correspond to the precondition of an CPS operator..[2]

Definition 2.2 *An instance of the SAS*$^+$ *planning problem is given by a tuple* $\Pi = \langle \mathcal{V}, \mathcal{O}, s_\mathcal{I}, s_\mathcal{G} \rangle$ *with components defined as follows:*

- $\mathcal{V} = \{v_1, \ldots, v_m\}$ *is a set of* **state variables**. *Each variable* $v \in \mathcal{V}$ *has an associated* **domain** $\mathcal{D}_v$*, which implicitly defines an* **extended domain** $\mathcal{D}_v^+ = \mathcal{D}_v \cup \{\mathsf{u}\}$*, where* u *denotes the* **undefined value**. *Further, the* **total state space** $\mathcal{S} = \mathcal{D}_{v_1} \times \ldots \times \mathcal{D}_{v_m}$ *and the* **partial state space** $\mathcal{S}^+ = \mathcal{D}_{v_1}^+ \times \ldots \times \mathcal{D}_{v_m}^+$ *are implicitly defined. 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* $\langle \mathsf{b}, \mathsf{e}, \mathsf{f} \rangle$*, where* $\mathsf{b}, \mathsf{e}, \mathsf{f} \in \mathcal{S}^+$ *denote the* **pre-, post-** *and* **prevail-condition** *respectively.* $\mathcal{O}$ *is subject to the following two restrictions: For every operator* $\langle \mathsf{b}, \mathsf{e}, \mathsf{f} \rangle \in \mathcal{O}$*,*

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

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

[2] Although, not technically necessary, this distinction between the pre-condition and the prevail-condition has shown to have conceptual advantages in some cases. For instance, it has been possible to identify certain restrictions that result in computationally tractable subcases of the SAS$^+$ planning problem (see Bäckström and Nebel [4] for a recent summary).

- $s_{\mathcal{I}} \in \mathcal{S}^+$ *and* $s_{\mathcal{G}} \in \mathcal{S}^+$ *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 pre- and prevail-conditions of an operator must never define the same variable.

We write $s[v]$ *to denote the value of the variable* v *in a state* s*. We also 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 $o = \langle \mathsf{b}, \mathsf{e}, \mathsf{f} \rangle$ *is a SAS*$^+$ *operator, we write* $\mathsf{b}(o)$, $\mathsf{e}(o)$ *and* $\mathsf{f}(o)$ *to denote* b, e *and* f *respectively. As previously,* $\mathcal{O}^*$ *denotes the set of operators sequences and the members of* $\mathcal{O}^*$ *are called plans. Given two states* $s, t \in \mathcal{S}^+$*, we define for all* $v \in \mathcal{V}$*,*

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

The ternary relation valid $\subseteq \mathcal{O}^* \times s_{\mathcal{I}} \times s_{\mathcal{G}}$ *is defined recursively s.t. for arbitrary operator sequence* $\langle o_1, \ldots, o_n \rangle \in \mathcal{O}^*$ *and arbitrary states* $s, t \in \mathcal{S}^+$*,* $valid(\langle o_1, \ldots, o_n \rangle, s, t)$ *iff either*

1. $n = 0$ *and* $t \sqsubseteq s$ *or*
2. $n > 0$, $\mathsf{b}(o_1) \sqsubseteq s$, $\mathsf{f}(o_1) \sqsubseteq s$ *and* $valid(\langle o_2, \ldots, o_n \rangle, (s \oplus \mathsf{e}(o_1)), t)$.

A plan $\langle o_1, \ldots, o_n \rangle \in \mathcal{O}^*$ **solves** Π *iff* $valid(\langle o_1, \ldots, o_n \rangle, s_{\mathcal{I}}, s_{\mathcal{G}})$.

3 The MAR Algorithm

McAllester and Rosenblitt [10] have presented a systematic partial-order planner for the CPS formalism. We will refer to this algorithm as the CPS-MAR, or simply MAR, algorithm[3] This section will briefly recapitulate the MAR algorithm and the definitions needed for it; the reader is referred to the original paper [10] for further intuition and explanation.

MAR is a refinement planner, simulating the initial and goal states by two operators, so we need the following definition.

Definition 3.1 *Given an CPS problem instance* $\Pi = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$*, there are two implicitly defined, special operators* $o_{\mathcal{I}} = \langle \emptyset, \mathcal{I}, \emptyset \rangle$ *and* $o_{\mathcal{G}} = \langle \mathcal{G}, \emptyset, \emptyset \rangle$*, which need not be members of* $\mathcal{O}$*.*

A **self-contained plan** *over* Π *is an operator sequence* $\langle o_1, \ldots, o_n \rangle$ *s.t.* $o_1 = o_{\mathcal{I}}$*,* $o_n = o_{\mathcal{G}}$ *and* $\langle o_2, \ldots, o_{n-1} \rangle \subseteq \mathcal{O}^*$*. A self-contained plan* ω *is* **valid** *iff* $valid(\omega, \emptyset, \emptyset)$*.*

[3] The lifted version of the algorithm is usually referred to as SNLP after a popular implementation of it.

Theorem 3.2 *Given an CPS problem instance* $\Pi = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$*, a self-contained plan* $\langle o_1, \ldots, o_n \rangle$ *over* Π *is valid iff valid*$(\langle o_2, \ldots, o_{n-1} \rangle, \mathcal{I}, \mathcal{G})$.

The following definitions are further needed.

Definition 3.3 *An* **action** *(or plan step) is an instantiation of an operator. A* **condition** π *is a propositional atom. A* **causal link** *is a triple* $\langle a, \pi, b \rangle$*, usually written* $a \xrightarrow{\pi} b$*, where* a *and* b *are actions and* π *is a condition s.t.* $\pi \in \alpha(a)$ *and* $\pi \in \varphi(b)$.

A **plan structure** *for an CPS instance* $\Pi = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ *is a triple* $\Theta = \langle \mathcal{A}, \Omega, \Lambda \rangle$ *s.t.* $\mathcal{A}$ *is a set of actions,* Ω *is a binary relation on* $\mathcal{A}$*, the action order, and* Λ *is a set of causal links over the action in* $\mathcal{A}$*. The members of* Ω *are usually written as* $a \prec b$*. Furthermore,* $\mathcal{A}$ *must contain two designated actions* a_{min} *and* a_{max}*, being instantiations of* $o_{\mathcal{I}}$ *and* $o_{\mathcal{G}}$ *respectively, and* Ω *must contain* $a_{min} \prec a_{max}$*. The minimal plan structure* $\langle \{a_{min}, a_{max}\}, \{a_{min} \prec a_{max}\}, \emptyset \rangle$ *is referred to as the* **empty plan structure** *for* Π *A plan structure* $\langle \mathcal{A}, \Omega, \Lambda \rangle$ *is* **order consistent** *iff the transitive closure,* Ω^+*, of* Ω *is irreflexive and* a_{min} *and* a_{max} *are the unique minimal and maximal elements respectively.*

An **open goal** *in a plan* $\langle \mathcal{A}, \Omega, \Lambda \rangle$ *is a tuple* $\langle a, \pi \rangle$ *s.t.* $a \in \mathcal{A}$*,* $\pi \in \varphi(a)$ *and there is no action* b *s.t.* $b \xrightarrow{\pi} a \in \Lambda$*. An action* $a \in \mathcal{A}$ *is a* **threat** *to a causal link* $b \xrightarrow{\pi} c \in \Lambda$ *iff* $a \neq b$*,* $\pi \in \alpha(a)$ *or* $\pi \in \delta(a)$ *and neither* $a \prec b \in \Omega^+$ *nor* $c \prec a \in \Omega^+$.

A plan structure $\Theta = \langle \mathcal{A}, \Omega, \Lambda \rangle$ *is* **complete** *iff it is order consistent and it contains no open goals and no threats.*

The MAR planner is shown in Figure 1.[4] It is used by passing it the empty plan structure for an CPS instance as argument and it then calls itself recursively, extending this plan structure. The algorithm is presented as a non-deterministic algorithm, that is it only defines its search space—a search strategy has to be added to the algorithm according to taste.

The following two theorems (due to McAllester and Rosenblitt [10]) together say that MAR is a sound and complete algorithm for CPS planning.

Theorem 3.4 *For every complete plan structure* $\langle \mathcal{A}, \Omega, \Lambda \rangle$ *for some CPS instance* $\Pi = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ *and for every topological sorting* $\langle a_1, \ldots, a_n \rangle$ *of* $\langle \mathcal{A}, \Omega^+ \rangle$*, the sequence of operators* $\langle o_2, \ldots, o_{n-1} \rangle$ *corresponding to the actions* $\langle a_2, \ldots, a_{n-1} \rangle$ *solves* Π.

Theorem 3.5 *If MAR is called with the empty plan for some CPS instance* $\Pi = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$*, then it returns a complete plan structure for* Π *if there exists some plan over* Π *solving* Π *and otherwise it fails.*

[4] There are some minor differences compared to the original MAR [10]: the cost cut-off limit is here not included in the algorithm *per se*, but considered to be part of the search strategy, and all action orderings implicit in the causal links are also duplicated in the order Ω. Further, we use a different notation.

```
1  procedure MAR(Θ);   where Θ = ⟨A, Ω, Λ⟩
2  begin
3    if ω is not order consistent  then fail
4    elsif ω is complete  then return ω
5    elsif there exists some c ∈ A which is a threat to some a --π--> b ∈ Λ
6     then
7       nondeterministically do either of the following
8       1. return Plan(⟨A, Ω ∪ {c ≺ a}, Λ⟩)
9       2. return Plan(⟨A, Ω ∪ {b ≺ c}, Λ⟩)
10   else there must exist some open goal ⟨a, π⟩ in ω
11      Arbitrarily choose such an open goal ⟨a, π⟩ and
        nondeterministically do either of the following
12      1. Nondeterministically choose some action b ∈ A that
13         contributes π;
14         return Plan(⟨A, Ω ∪ {b ≺ a}, Λ ∪ {b --π--> a}⟩);
15      2. Nondeterministically choose some operator o ∈ O that
        contributes π
16         Let b be a new instantiation of o;
17         return Plan(⟨A ∪ {b}, Ω ∪ {b ≺ a}, Λ ∪ {b --π--> a}⟩)
18   end if
19 end
```

Figure 1: The MAR Algorithm

Everything in the MAR algorithm that is specific to the CPS formalism is abstracted away from the actual algorithm and hidden in the underlying definitions. Hence, we can get a planning algorithm for SAS$^+$ by making some changes to these underlying definitions only. No changes are required in the actual algorithm—the different definitions would result in different implementations of the algorithm, though. The SAS$^+$ version of the algorithm, referred to as SAS$^+$-MAR, results if we substitute SAS$^+$ instances for CPS instances in the definitions in the previous section and make the following redefinitions.

Definition 3.6 *Given an SAS$^+$ problem instance $\Pi = \langle \mathcal{V}, \mathcal{O}, s_\mathcal{I}, s_\mathcal{G} \rangle$, there are two implicitly defined, special operators $o_\mathcal{I} = \langle \mathsf{u}, s_\mathcal{I}, \mathsf{u} \rangle$ and $o_\mathcal{G} = \langle \mathsf{u}, \mathsf{u}, s_\mathcal{G} \rangle$, which need not be members of $\mathcal{O}$.*

A **self-contained plan** *over Π is an operator sequence $\langle o_1, \ldots, o_n \rangle$ s.t. $o_1 = o_{s_\mathcal{I}}$, $o_n = o_{s_\mathcal{G}}$ and $\langle o_2, \ldots, o_{n-1} \rangle \subseteq \mathcal{O}^*$. A self-contained plan ω is* **valid** *iff valid$(\omega, \langle \mathsf{u}, \ldots, \mathsf{u} \rangle, \langle \mathsf{u}, \ldots, \mathsf{u} \rangle)$.*

Theorem 3.7 *Given an SAS$^+$ problem instance $\Pi = \langle \mathcal{V}, \mathcal{O}, s_\mathcal{I}, s_\mathcal{G} \rangle$, a self-contained plan $\langle o_1, \ldots, o_n \rangle$ over Π is valid iff valid$(\langle o_2, \ldots, o_{n-1} \rangle, s_\mathcal{I}, s_\mathcal{G})$.*

Definition 3.8 *A* **condition** *π is a tuple $\langle v, x \rangle$ s.t. $v \in \mathcal{V}$ and $x \in \mathcal{D}_v$.*

A **causal link** *is a triple $\langle a, \pi, b \rangle$, usually written $a \xrightarrow{\pi} b$, where a and b are actions and $\pi = \langle v, x \rangle$ is a condition s.t. $\mathsf{e}(a)[v] = x$ and $\mathsf{b}(b) = x$.*

An **open goal** *in a plan $\langle \mathcal{A}, \Omega, \Lambda \rangle$ is a tuple $\langle a, \pi \rangle$ s.t. $a \in \mathcal{A}$, $\pi = \langle v, x \rangle$, either $\mathsf{b}(a)[v] = x$ or $\mathsf{f}(a)[v] = x$ and there is no action b s.t. $\langle b, \pi, a \rangle \in \Lambda$.*

An action $a \in \mathcal{A}$ is a **threat** *to a causal link $\langle b, \langle v, x \rangle, c \rangle \in \Lambda$ iff $a \neq b$, $\mathsf{e}(a) = y$ for some $y \in \mathcal{D}_v$ and neither $a \prec b \in \Omega^+$ nor $c \prec a \in \Omega^+$.*

A plan $\omega = \langle \mathcal{A}, \Omega, \Lambda \rangle$ is **complete** *iff ω is order consistent, ω contains no open goals and for each causal link $\langle b, \pi, c \rangle \in \Lambda$ which is threatened by some action $a \in \mathcal{A}$, either $a \prec b \in \Omega$ or $c \prec a \in \Omega$.*

Also SAS$^+$-MAR is a sound and complete planning algorithm.

Theorem 3.9 *For every complete plan structure $\langle \mathcal{A}, \Omega, \Lambda \rangle$ for some SAS$^+$ instance $\Pi = \langle \mathcal{V}, \mathcal{O}, s_\mathcal{I}, s_\mathcal{G} \rangle$ and for every topological sorting $\langle a_1, \ldots, a_n \rangle$ of $\langle \mathcal{A}, \Omega^+ \rangle$, the sequence of operators $\langle o_2, \ldots, o_{n-1} \rangle$ corresponding to the actions $\langle a_2, \ldots, a_{n-1} \rangle$ solves Π.*

Theorem 3.10 *If MAR is called with the empty plan for some SAS$^+$ instance $\Pi = \langle \mathcal{V}, \mathcal{O}, s_\mathcal{I}, s_\mathcal{G} \rangle$, then it returns a complete plan structure for Π if there exists some plan over Π solving Π and otherwise it fails.*

4 Lin- and Log-encodings

We know from the formalism equivalence proofs [1, 2] that a SAS$^+$ problem instance can be solved by re-encoding it as an equivalent CPS problem instance in the following way, which we call a log-encoding.

Definition 4.1 *A* **log-encoding** *of a SAS$^+$ problem instance $\Pi = \langle \{v_1, \ldots, v_m\}, \mathcal{O}, s_\mathcal{I}, s_\mathcal{G} \rangle$ is an equivalent CPS instance Π' defined as follows. For each $v \in \mathcal{V}$ define a set of atoms $\mathcal{P}_v$ as follows. Assume wlg. that $\mathcal{D}_v = \{0, \ldots, n_v - 1\}$ for some $n_v \geq 0$. Let $k_v = \lceil \log n_v \rceil$. Further, let the function $bit_v : \mathcal{D}_v \times \{1, \ldots, k_v\} \rightarrow \{0, 1\}^{k_v}$ be defined s.t. for $x \in \mathcal{D}_v$ and $1 \leq i \leq k_v$, $bit(x, i)$ is the ith bit in the binary encoding of x. Now define $\mathcal{P}_v = \{p_1^v, \overline{p}_1^v, \ldots, p_{k_v}^v, \overline{p}_{k_v}^v\}$ where all members are distinct atoms. The state variable v can be encoded by the atoms of $\mathcal{P}_v$ in the following way. Define the function $\mu_v : \mathcal{D}_v^+ \rightarrow \mathbf{2}^{\mathcal{P}_v}$ s.t. for $x \in \mathcal{D}_v$ and $1 \leq i \leq k_v$,*

$$p_i^v \in \mu_v(x) \text{ iff } x \neq \mathsf{u} \text{ and } bit(x, i) = 1$$

$$\overline{p}_i^v \in \mu_v(x) \text{ iff } x \neq \mathsf{u} \text{ and } bit(x, i) = 0.$$

Since all $\mathcal{P}_v$ are disjoint we can define $\mathcal{P} = \cup_{v \in \mathcal{V}} \mathcal{P}_v$ and merge all μ_v into one function $\mu : \mathcal{S}^+ \rightarrow \mathbf{2}^{\mathcal{P}}$ defined as s.t. for $s \in \mathcal{S}^+$, $\mu(s) = \cup_{v \in \mathcal{V}} \mu_v(s[v])$. Also define the function $\overline{\mu} : \mathcal{S}^+ \rightarrow \mathbf{2}^{\mathcal{P}}$ defined s.t. for $s \in \mathcal{S}^+$, $\overline{\mu}(s) = \cup_{v \in \mathcal{V}'} (\mathcal{P}_v - \mu_v(s[v]))$, where $\mathcal{V}' = \{v \in \mathcal{V} \mid s[v] \neq \mathsf{u}\}$. Further, define a set $\mathcal{O}'$ of CPS operators s.t. for each operator $\langle \mathsf{b}, \mathsf{e}, \mathsf{f} \rangle \in \mathcal{O}$, $\mathcal{O}'$ contains the operator $\langle \mu(\mathsf{b}) \cup (\mathsf{f}), \mu(\mathsf{e}), \overline{\mu}(\mathsf{e}) \rangle$. Finally, let $\Pi' = \langle \mathcal{P}, \mathcal{O}', \mu(s_\mathcal{I}), \mu(s_\mathcal{G}) \rangle$ be the corresponding CPS instance.

Similarly, we define a lin-encoding.

Definition 4.2 *A* lin-encoding *of a SAS$^+$ instance is an equivalent CPS instance defined in the same way as a log-encoding, but with the following two differences. For each $v \in \mathcal{V}$, the set $\mathcal{P}_v$ and the function μ are defined s.t. $\mathcal{P}_v = \{p_1^v, \ldots, p_{n_v}^v\}$ and $p_i^v \in \mu_v(x)$ iff $x = i$.*

In a log-encoding, a state variable v with a domain of size n is simulated by the domain $\mathcal{P}_v$, containing $2\lceil \log n \rceil$ distinct atoms. If the variable is undefined, then it is encoded as the empty set and otherwise it is encoded by a subset S of $\mathcal{P}_v$ s.t. for each pair of atoms $p_i^v, \overline{p}_i^v$ exactly one of these atoms is in S. A lin-encoding is similar, but each possible value x of the variable v is simulated by its own, distinct atom p_x^v. The log-encoding constitutes a polynomial reduction of SAS$^+$ into CPS while the lin-encoding is not a polynomial reduction in the general case [1].

To get som intuition for these encodings, consider the following example. Let $\Pi = \langle \mathcal{V}, \mathcal{O}, s_\mathcal{I}, s_\mathcal{G} \rangle$ be a SAS$^+$ instance where $\mathcal{V} = \{v_1, v_2, v_3\}$ with domains $\mathcal{D}_{v_1} = \mathcal{D}_{v_2} = \mathcal{D}_{v_3} = \{0, \ldots, 3\}$. The values of μ_v for each value of a variable $v \in \mathcal{V}$ are shown in Table 2. Table 3 shows how a sample SAS$^+$ operator in $\mathcal{O}$ is lin- and log-encoded.

	x	u	0	1	2	3
log-reduction	$\mu_v(x)$	$\emptyset$	$\{\overline{p}_1^v, \overline{p}_2^v\}$	$\{\overline{p}_1^v, p_2^v\}$	$\{p_1^v, \overline{p}_2^v\}$	$\{p_1^v, p_2^v\}$
lin-reduction	$\mu_v(x)$	$\emptyset$	$\{p_0^v\}$	$\{p_1^v\}$	$\{p_2^v\}$	$\{p_3^v\}$

Table 2: Log- and lin-encodings of a state variable with domain $\{0, \ldots, 3\}$
.

SAS$^+$ operator o	$\mathsf{b} = \langle \mathsf{u}, \mathsf{u}, 1 \rangle$	$\mathsf{e} = \langle \mathsf{u}, 1, 2 \rangle$	$\mathsf{f} = \langle 3, \mathsf{u}, \mathsf{u} \rangle$
log-reduction of o	$\varphi = \{p_1^1, p_2^1, \overline{p}_1^3, p_2^3\}$	$\alpha = \{\overline{p}_1^2, p_2^2, p_1^3, \overline{p}_2^3\}$	$\delta = \{p_1^2, \overline{p}_2^2, \overline{p}_1^3, p_2^3\}$
lin-reduction of o	$\varphi = \{p_1^3, p_3^1\}$	$\alpha = \{p_1^2, p_2^3\}$	$\delta = \{p_0^2, p_2^2, p_3^2, p_0^3, p_1^3, p_3^3\}$

Table 3: Log- and lin-encodings of a SAS$^+$ operator.

5 Comparison

In this section we will compare solving SAS$^+$ instances directly using the SAS$^+$-MAR algorithm with solving them indirectly by applying the CPS-MAR algorithm to the corresponding lin- or log-encoded instances. CPS-MAR and SAS$^+$-MAR are, in principle, the same algorithm, differing at most polynomially in the cost per search node. That is, the time CPS-MAR takes to solve the hardest CPS instance differs at most polynomially from the time SAS$^+$-MAR takes to solve the hardest SAS$^+$ instance. This, however, does not tell us anything about the time required to solve an arbitrary SAS instance and solving its log-encoding respectively. We will, thus, analyse the

relative complexity of solving an an arbitrary SAS instance and solving its log- and lin-encodings respectively.

We do not commit ourselves to any particular search technique. Rather, we will compare the size of the search spaces. Since the MAR algorithm searches the plan space, which is infinite, we must restrict the analysis to search trees that are cut at a certain depth. That is, we cut the search tree at every node containing a plan structure with more than k actions for some k, which can be captured formally as follows.

Definition 5.1 *A* **depth** n **search tree** *for MAR, is a MAR search tree where every node containing more than* $n+2$ *actions is a leaf.*

As long as we are not considering any domain-specific heuristics, this should give a fair estimate of the time complexity for any search technique using such a limit for cutting branches. This can be formally captured as follows, where the additive constant 2 compensates for the actions a_{min} and a_{max}.

It turns out that if open goals are chosen according to a FIFO strategy, then it may be exponentially more efficient to use SAS$^+$-MAR than to solve a log-encoding using CPS-MAR. If a LIFO strategy is used, then it is at least polynomially more efficient to use SAS$^+$-MAR.[5]

Theorem 5.2 *If a FIFO strategy is used for selecting open goals in line 10 of MAR, then for each* $n > 0$*, there exists a SAS*$^+$ *instance* Π *s.t. SAS*$^+$*-MAR has a depth* n *search space of size* $O(n)$ *for* Π *and CPS-MAR has a depth* n *search space of size* $\Omega(2^n)$ *for the corresponding log-encoded CPS instance.*

Proof sketch: For arbitrary $n > 0$, let $\mathcal{V} = \{v\}$ and $\mathcal{D}_v = \{0, \ldots, n-1\}$. Wlg. assume $n = 2^k$ for some $k \geq 0$. Let $\mathcal{O} = \{o_1, \ldots, o_n\}$ where for each $1 \leq i \leq n$, $\mathsf{b}(o_i)[v] = i - 1$, $\mathsf{e}(o_i)[v] = i \bmod n$ and $\mathsf{f}(o_i)[v] = \mathsf{u}$. Define the SAS$^+$ instance $\Pi = \langle \mathcal{V}, \mathcal{O}, \langle 0 \rangle, \langle n-1 \rangle \rangle$ and let Π' be the corresponding log-encoded CPS instance. It is obvious that SAS$^+$-MAR has a non-branching search tree of depth $O(n)$ which is of size $O(n)$, since it is non-branching. Now consider the CPS-MAR search space for Π'. Call a node in the search tree *live* if (1) it is either a leaf containing a plan with $\geq n+2$ actions or (2) it has at least two children which are live. Every threat-free live non-leaf has some open goal and there are at least two different operators that can be instantiated to close this goal. It follows by induction over n that the root of the search tree is live, which immediately implies that the search tree is of size $\Omega(2^n)$. □

Theorem 5.3 *If a LIFO strategy is used for selecting open goals in line 10 of MAR, then for each* $n > 0$*, there exists a SAS*$^+$ *instance* Π *s.t. SAS*$^+$*-MAR has a search space of size* $O(n)$ *for* Π *and CPS-MAR has a search space of size* $\Omega(n^2 \log n)$ *for the corresponding log-encoded CPS instance.*

[5] Analysing only these two strategies does not allow us to conclude that there can be a state-space blow-up for all goal selection strategies, but FIFO and LIFO are at least the extreme cases for such strategies.

Proof sketch: Consider the same SAS$^+$ instance Π as in the proof of Theorem 5.2 and the, same, log-encoding Π'. Consider the CPS-MAR search space for Π'. We introduce the following terminology. Any consistent node without threats is *live* if it is either a solution node or some of its children is live; *full-open* if the next open goal to select belongs to an action, the selected action, having all its preconditions open and *semi-open* if the next open goal to select belongs to an action, the selected action, having at least some preconditions closed by a causal link. We also note that there exists only one solution leaf in the search tree, containing a plan of length $n + 2$ (including the initial and goal actions). Hence, all live nodes are on the path to this, unique solution leaf. Each operator has $\lceil \log n \rceil / 2$ preconditions, so there must be $\Theta(n \log n)$ live nodes, of which $\Theta(n)$ are full-open and $\Theta(n \log n)$ are semi-open.

Each full-open, live non-leaf has at least $n/2$ semi-open children, corresponding to the $n/2$ different operators that can be instantiated to produce the next open goal. Only one of these nodes instantiates an operator that can produce all the preconditions of the selected action, however, so there is exactly one live child among these $n/2$ nodes. Hence, each live, full-open node has at $\Omega(n)$ children, containing some live semi-open node. A similar argument shows that every live, semi-open node has $\Omega(n)$ children, containing some node which is live. Since there are $\Theta(n \log n)$ live nodes, it follows that the search space is of size $\Omega(n^2 \log n)$. □

This latter theorem provides only a lower bound, so we cannot rule out the possibility that there is an exponential difference also for the LIFO strategy. It seems unlikely that there should exist a strategy allowing us to to solve a log-encoding of a SAS$^+$ instance using CPS-MAR as efficiently as if using SAS$^+$-MAR directly. solving However, we could introduce an operator-selection filter in line 14 of the MAR algorithm in the following way. Whenever there is some action of type o having an open goal $p \in \mathcal{P}_v$, for some variable v, then only operators o' s.t. $\alpha(o') \cap \mathcal{P}_v = \varphi(o') \cap \mathcal{P}_v$ are considered for instantiation. Such a filter would make the algorithm avoid branching by selecting the wrong operator to instantiate, so the exponential search-space blow-up disappears. The CPS-MAR search space will still be slightly larger, though, but only within some constant. The reason for this is as follows. Suppose there is some action of type o having an open goal $p \in \mathcal{P}_v$, for some variable v, and that some other precondition $q \in \mathcal{P}_v$ is produced by some action a of type o' in the plan structure. The right choice for producing p is to re-use the action a, but the algorithm would also consider instantiating a new action of type o', leading to a double-cross conflict (three actions constituting two symmetric right-forks [7]), which cannot lead to a solution. However, also this problem could be filtered away. It should be noted, however, that when using such filters, we are allowing the algorithm to use information which is not available in the CPS instance Π', so we are, strictly speaking, no longer doing domain-independent planning.

If using lin-encodings, on the other hand, there is no search-space blow-up at all.

Definition 5.4 *Let* Π *be a SAS*$^+$ *instance and* Π' *its corresponding lin-encoded CPS instance. Let* T *be the search tree of SAS*$^+$*-MAR for* Π *and* T' *the search tree of CPS-MAR for* Π'*. T and* T' *are isomorphic if they are isomorphic trees s.t. for every* N *in* T *and its corresponding node* N' *in* T'*, the plan structures of* N *and* N' *are isomorphic and for each action* a *in* N *and its corresponding action* b *in* N'*,* b *instantiates the lin-encoding of the operator* a *instantiates.*

Theorem 5.5 *Given A SAS*$^+$ *instance* Π *and its lin-encoded equivalent* Π'*, the CPS-MAR search tree for* Π' *is isomorphic to the SAS*$^+$*-MAR search tree for* Π *s.t. the plan structures in corresponding nodes in the two search trees are isomorphic..*

It is immediate that two search spaces are of the same size since they are isomorphic. Unfortunately, this does not imply that the cost of solving a lin-encoded instance is comparable to the cost of solving the original SAS$^+$ instance. The reason is that, as mentioned in the previous section, lin-encodings are not polynomial reductions. More precisely, the number of propositions encoding a state variable may be exponential in the size of the original representation of the state variable. Hence, the cost per node in the search tree for the lin-encoding may be exponentially larger than the cost per node in the search tree for the original SAS$^+$ instance, at least if the size of the state variable domains dominate over the number of state variables and the number of actions in the node.

Often, domains modelled by state variables are solved using a first-order formalism,*eg.* STRIPS or TWEAK, encoding the state variables as unary predicates. However, encoding each state variable as a unary relation and using the lifted version of MAR [10] would be equivalent to using a lin-encoding, so also in this case do we risk an exponentially higher cost per node.

It is further easy to see from the definitions and proofs, that even if allowing negative preconditions, negative goals and partial initial states, *ie.* using either the PSN or GT formalims, the theorems in this section would still hold. The CPS formalism was used only because that is the formalism MAR was designed and defined for originally.

Finally, it is immediate from the formalism equivalence proofs [1, 3] that CPS, PSN and GT instances can be reencoded as SAS$^+$ instances and be solved as efficiently using SAS$^+$-MAR as if using the original CPS/PSN/GT-MAR.

6 Conclusions

We have considered two planning formalisms which are known to be expressively equivalent—the CPS formalism, using propositional atoms, and the SAS$^+$ formalism, using multi-valued state variables. As a case study, we have modified a well-known partial-order planner for CPS into an 'identical' planner for SAS$^+$and we have considered two encodings of SAS$^+$ into CPS.

It turns out that it is more efficient to solve SAS^+ instances by using the SAS^+ planner directly than to first encode them as CPS instances and use the CPS planner. For one encoding of SAS^+ into CPS, which is a polynomial reduction, the CPS planner has a polynomially or exponentially larger search space, depending on goal selection strategy. For the other encoding, which is not a polynomial reduction, the CPS search is of the same size as the SAS^+ search space, but the cost per node can be exponentially higher in this case. To the contrary, however, no similar problems arise when re-encoding CPS instances as SAS^+ instances. Some of the problems can be avoided if we make the comparison unfair, allowing the CPS version of the planner to use domain-specific heuristics or knowledge about the variable encodings. However, in the first case the comparison is then between a domain-independent planner and a domain-dependent one and in the other case, we are no longer using the pure CPS formalism.

Furhter note that encoding each state variable as a unary relation and using the lifted version of MAR [10] would be equivalent to using a lin-encoding, so there may be an exponentially higher node cost also in this case.

The main conclusion is that even if state variables can be easily re-encoded as sets of atoms or as predicates, we may risk losing efficiency if solving such a re-encoded instance rather than the original one. Re-encodings may work well in many practical cases, but must then be used with care.

Acknowledgements

This research was sponsored by *the Swedish Research Council for the Engineering Sciences (TFR)* under grant Dnr. 92-143.

References

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

[2] Christer Bäckström. Equivalence and tractability results for SAS^+ planning. In Swartout and Nebel [12], pages 126–137.

[3] Christer Bäckström. Expressive equivalence of planning formalisms. Submitted manuscript.

[4] Christer Bäckström and Bernhard Nebel. Complexity results for SAS^+ planning. In Bajcsy [5].

[5] Ruzena Bajcsy, editor. *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, Chambéry, France, August–September 1993. Morgan Kaufmann.

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

[7] Joachim Hertzberg and Alexander Horz. Towards a theory of conflict detection and resolution in nonlinear plans. In N. S. Sridharan, editor, *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 937–942, Detroit, MI, USA, August 1989. Morgan Kaufmann.

[8] 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.

[9] Subbarao Kambhampati. On the utility of systematicity: Understanding tradeoffs between redundancy and commitment in partial-order planning. In Bajcsy [5].

[10] David McAllester and David Rosenblitt. Systematic nonlinear planning. In *Proceedings of the 9th (US) National Conference on Artificial Intelligence (AAAI-91)*, pages 634–639, Anaheim, CA, USA, July 1991. American Association for Artificial Intelligence, AAAI Press/MIT Press.

[11] Steven Minton, Mark Drummond, John L. Bresina, and Andrew B. Philips. Total order *vs.* partial order planning: Factors influencing performance. In Swartout and Nebel [12], pages 83–92.

[12] Bill Swartout and Bernhard Nebel, editors. *Proceedings of the 3rd International Conference on Principles on Knowledge Representation and Reasoning (KR-92)*, Cambridge, MA, USA, October 1992. Morgan Kaufmann.