To appear in W. Burgard, A. B. Cremers, T. Christaller, eds., *Advances in Artificial Intelligence, Proceedings / KI-99, 23th Annual German Conference on Artificial Intelligence*, Bonn, Germany, September 1999. (C)Springer-Verlag.

Compilation Schemes: A Theoretical Tool for Assessing the Expressive Power of Planning Formalisms

Bernhard Nebel

Institut für Informatik, Albert-Ludwigs-Universität
Freiburg, Germany

Abstract. The recent approaches of extending the GRAPHPLAN algorithm to handle more expressive planning formalisms raise the question of what the formal meaning of "expressive power" is. We formalize the intuition that expressive power is a measure of how concisely planning domains and plans can be expressed in a particular formalism by introducing the notion of "compilation schemes" between planning formalisms. Using this notion, we analyze the expressive power of a large family of propositional planning formalisms and show, e.g., that Gazen and Knoblock's approach to compiling conditional effects away is optimal.

1 Introduction

The recent approaches of extending the GRAPHPLAN algorithm [3] to handle more expressive planning formalisms [1,9,10,12] raise the question of what the formal meaning of *expressive power* is. In order to address the problem of measuring the relative expressive power of planning formalisms, we start with the intuition that a formalism $\mathcal{X}$ is *at least as expressive* as another formalism $\mathcal{Y}$ if *planning domains* and the corresponding *plans* in formalism $\mathcal{Y}$ can be *concisely expressed* in the formalism $\mathcal{X}$.

Bäckström [2] proposed to measure the expressiveness of planning formalisms using *ESP-reductions*, which are, roughly speaking, polynomial many-one reductions[1] on planning instances that *do not change the plan length*. Using this notion, he showed that all of the propositional variants of basic STRIPS not containing conditional effects or arbitrary logical formulae can be considered as expressively equivalent. However, taking our point of view, on one hand ESP-reductions are too restrictive. Firstly, plans must have identical size, while we might want to allow a moderate growth. Secondly, requiring that the transformation can be computed in polynomial time is overly restrictive. In fact, one

[1] We assume that the reader has a basic knowledge of complexity theory [8,16], and is familiar with the notion of *polynomial many-one reductions* and the *complexity classes* P, NP, coNP, and PSPACE. All other notions will be introduced in the paper when needed.

formalism might be as expressive as another one, but the mapping between the formalisms might not recursive at all. On the other hand, ESP-reductions are too liberal because they allow for arbitrary transformations between planning instances. We, however, want to consider only transformations between planning domains, which are independent from the concrete initial state and goal specification.

Inspired by recent approaches in the area of *knowledge compilation* [5], we formalize the notion of *concisely expressible* using *compilation schemes*, which are solution-preserving mappings from planning domains in one formalism to planning domains in another formalism restricting the result to be of polynomial size. Furthermore, we distinguish between compilation schemes in whether they *preserve plan size exactly*, *linearly*, or *polynomially*.

Using the notion of *compilability*, we analyze a wide range of propositional planning formalisms, ranging from basic STRIPS to a planning formalism containing *conditional effects*, *arbitrary logical formulae*, and *partial state specifications*. As one of the results, we identify two equivalence classes of planning formalisms with respect to *polynomial-time* compilability preserving plan size exactly. This means that adding a language feature to a formalism without leaving the class does not increase the expressive power and should not affect the principal efficiency of the planning method. However, we also provide results that *separate* planning formalisms. Such separation results indicate that adding a particular language feature adds to the expressive power and to the difficulty of finding a plan. For example, we prove that conditional effects cannot be compiled away and that logical formulae cannot be compiled into conditional effects—provided the plans in the target formalism are allowed to grow only linearly. This answers, e.g., the question of whether Gazen and Knoblock's [9] approach to compile conditional effects away could be improved. If only linear plan growth is allowed, the size of the compiled domain structure is necessarily super-polynomial.

The rest of the paper is structured as follows. In Section 2, we introduce the range of propositional planning formalisms analyzed in this paper together with general terminology and definitions. Based on that, we introduce in Section 3 the notion of compilability between planning formalisms. In Section 4 we present polynomial-time compilation schemes between different formalisms that preserve the plan size, demonstrating that these formalisms are of identical expressiveness. For all of the remaining cases, we prove in Section 5 that there cannot be any compilation scheme preserving plan size linearly, even if there are no bounds on the computational resources of the compilation process. Finally, in Section 6 we summarize and discuss the results.

2 Propositional Planning Formalisms

First, we will define a very general propositional planning formalism, which appears to be as expressive as the propositional variant of ADL [17]. We will then consider various syntacticly restricted variants of this formalism.

Let Σ be a finite set of propositional **atoms**. Then, $\widehat{\Sigma}$ is defined to be the set consisting of the constants $\top$ (denoting truth) and $\bot$ (denoting falsity) as well as the **literals**, i.e., atoms and negated atoms, over Σ. The **language of propositional logic** over Σ is denoted by $\mathcal{L}_\Sigma$. Given a set of literals L, we define $\neg L$ to be the **element-wise negation** of L, i.e., $\neg L = \{p | \neg p \in L\} \cup \{\neg p | p \in L\}$.

A **state** s is a *truth-assignment* for the atoms in Σ, which is represented by the set of atoms that are true in this state. A **state specification** S is a subset of $\widehat{\Sigma}$, i.e., it is a *logical theory* consisting of literals only. It is called **consistent** iff it does not contain complementary literals or $\bot$. In general, a state specification describes many states, namely all those that satisfy S. Only in case that S is **complete**, i.e., for each $p \in \Sigma$ we have either $p \in S$ or $\neg p \in S$, S describes precisely one state. By abusing notation, we will refer to the **inconsistent** state specification by $\bot$, which is the **"illegal" state specification**.

Operators are pairs $o = \langle pre, post \rangle$. We use the notation $pre(o)$ and $post(o)$ to refer to the first and second part of an operator o, respectively. The **precondition** pre is a set of propositional formulae. The set $post$, which is the set of **postconditions**, consists of **conditional effects**, each having the form $\Gamma \Rightarrow L$, where the elements of $\Gamma \subseteq \mathcal{L}_\Sigma$ are called **effect conditions** and the elements of $L \subseteq \widehat{\Sigma}$ are called **effects**. Given a state specification S and an operator o, we define the set of **active effects** $A(S, post(o))$ and the set of **potentially active effects** $P(S, post(o))$ as follows:

$$A(S, post(o)) = \bigcup \{L \mid (\Gamma \Rightarrow L) \in post(o), S \models \Gamma\},$$
$$P(S, post(o)) = \bigcup \{L \mid (\Gamma \Rightarrow L) \in post(o), S \cup \Gamma \not\models \bot\}.$$

If for a given state specification S and an operator o, we have $A(S, post(o)) \neq P(S, post(o))$, this implies that the activated effects differ for different states described by the state specification. For this reason, we consider the application of the operator o as illegal in this situation.[2] Similarly to the rule that $A(S, post(o)) \neq P(S, post(o))$ leads to an illegal state specification, we require that if the precondition is not entailed by S or if the state specification is already inconsistent, the result of applying o to S results in $\bot$. This leads to the definition of the operator-result function R for operator o on the state specification S:

$$R(S, o) = \begin{cases} S - \neg A(S, post(o)) \cup A(S, post(o)) & \text{if } S \not\models \bot \text{ and} \\ & S \models pre(o) \text{ and} \\ & A(S, post(o)) \not\models \bot \text{ and} \\ & A(S, post(o)) = P(S, post(o)) \\ \bot & \text{otherwise} \end{cases}$$

A **planning instance** is now a tuple $\Pi = \langle \Xi, \mathbf{I}, \mathbf{G} \rangle$, where

- $\Xi = \langle \Sigma, \mathbf{O} \rangle$ is the **domain structure** consisting of a finite set of propositional atoms Σ and a finite set of operators $\mathbf{O}$,

[2] Of course, alternative semantics are possible.

– $\mathbf{I} \subseteq \widehat{\Sigma}$ is the **initial state specification**, and
– $\mathbf{G} \subseteq \widehat{\Sigma}$ is the **goal specification**.

When we talk about the **size of an instance**, symbolically $||\Pi||$, in the following, we mean the size of a (reasonable) encoding of the instance.

Let Δ be an element of $\mathbf{O}^\star$, i.e., a finite sequence of operators, which is called **plan**. Then $||\Delta||$ denotes the size of the plan, i.e., the number of operators in Δ. We say that Δ is a c**-step plan** if $||\Delta|| \leq c$. The result of applying Δ to a state specification S is recursively defined as follows:

$$Res(S, \langle\rangle) = S,$$
$$Res(S, \langle o_1, o_2, \ldots, o_n\rangle) = Res(R(S, o_1), \langle o_2, \ldots, o_n\rangle).$$

A sequence of operators Δ is said to be a **plan for** Π or a **solution of** Π iff (1) $Res(\mathbf{I}, \Delta) \not\models \bot$ and (2) $Res(\mathbf{I}, \Delta) \models \mathbf{G}$. Note that it can be easily verified [14] that any plan Δ for an instance Π is a *sound plan* in Lifschitz' [13] sense.

The propositional variant of basic STRIPS [6], which we will also call $\mathcal{S}$ in what follows, is a planning formalism that requires *complete state specifications*, *unconditional effects*, and *propositional atoms* as formulae. Less restrictive planning formalisms can have the following additional features:

Incomplete state specifications ($\mathcal{I}$): The state specifications may not be complete.
Conditional Effects ($\mathcal{C}$): Effects can be conditional.
Literals as formulae ($\mathcal{L}$): The formulae in preconditions and effect conditions can be literals.
Boolean formulae ($\mathcal{B}$): The formulae in preconditions and effect conditions can be arbitrary boolean formulae.

These extensions can also be combined. We will use combinations of letters to refer to such multiple extensions. For instance, $\mathcal{S}_\mathcal{L}$ refers to the formalism $\mathcal{S}$ extended by literals in the precondition lists, $\mathcal{S}_{\mathcal{IC}}$ refers to the formalism allowing for incomplete state specifications and conditional effects, but all formulae have to be atoms, and $\mathcal{S}_{\mathcal{BIC}}$, finally, refers to the general planning formalism introduced above. Figure 1 displays the partial order on propositional planning formalisms defined in this way using a Hasse diagram. In the sequel we say that $\mathcal{X}$ **is a specialization** of $\mathcal{Y}$, written $\mathcal{X} \sqsubseteq \mathcal{Y}$, iff $\mathcal{X}$ is identical to $\mathcal{Y}$ or below $\mathcal{Y}$ in the Hasse diagram.

Comparing this set of planning formalisms with the one Bäckström [2] analyzed, one notices that despite small differences in the presentation of the planning formalisms $\mathcal{S}$, $\mathcal{S}_\mathcal{L}$, and $\mathcal{S}_{\mathcal{LI}}$ are identical to CPS, PSN, and GT, respectively.

While one would expect that planning in $\mathcal{S}$ is much easier than planning in $\mathcal{S}_{\mathcal{BIC}}$, it turns out that—taking a computational complexity perspective—this is not the case. When analyzing the computational complexity of planning in different formalisms, we consider as usual the problem of deciding whether there *exists a plan* for a given instance—the **plan existence problem** (PLANEX). We will use a prefix referring to the planning formalism if we consider the existence problem in a particular planning formalism.

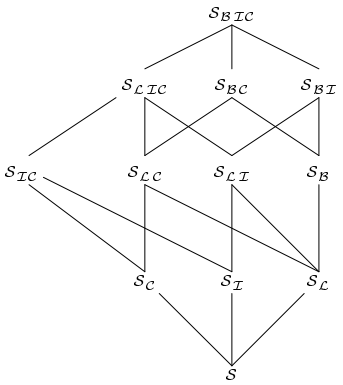

Fig. 1. Planning formalisms partially ordered by syntactic features

Theorem 1. *$\mathcal{X}$-PLANEX is PSPACE-complete for all $\mathcal{X}$ with $\mathcal{S} \sqsubseteq \mathcal{X} \sqsubseteq \mathcal{S}_{\mathcal{BIC}}$.*

Proof Sketch. PSPACE-hardness of $\mathcal{S}$-PLANEX follows from [4, Corollary 3.2]. Membership of $\mathcal{S}_{\mathcal{BIC}}$-PLANEX in PSPACE follows because each plan step can be verified by an NP-oracle, i.e., in PSPACE.[3] ∎

While this result may be interpreted as showing that we can easily solve the $\mathcal{S}_{\mathcal{BIC}}$ planning problem by transforming it to $\mathcal{S}$, it does not say anything about how easily we can transform *domain structures* and how long plans will be after the transformation.

3 Compilation Schemes

Basically, *compilation schemes* are solution-preserving mappings (which need not to be recursive) on domain structures.[4] Since in these mappings auxiliary atoms might be introduced, the compilation scheme may also influence the initial state and goal specifications. Furthermore, the initial state and goal specifications may also need some massaging, e.g., when compiling from partial to complete state specifications. The necessary *state-translation functions* should be very limited in computational power, however, so that they do not influence the principal expressive power.

A **compilation scheme from $\mathcal{X}$ to $\mathcal{Y}$** is a tuple of functions $\mathbf{f} = \langle f_\xi, f_i, f_g, t_i, t_g \rangle$ that induces a function F from $\mathcal{X}$-instances $\Pi = \langle \Xi, \mathbf{I}, \mathbf{G} \rangle$ to $\mathcal{Y}$-instances $F(\Pi)$ as follows:

$$F(\Pi) = \big\langle f_\xi(\Xi), f_i(\Xi) \cup t_i(\Sigma, \mathbf{I}), f_g(\Xi) \cup t_g(\Sigma, \mathbf{G}) \big\rangle,$$

and satisfies the following conditions:

[3] Full proofs for all theorems can be found in the long version of the paper [14].

[4] This means that compilation schemes between planning formalisms are similar to knowledge compilation schemes [5], where the fixed part of a computational problem is the domain structure and the variable part consists of the initial state and goal specifications.

1. there exists a plan for Π iff there exists a plan for $F(\Pi)$;
2. the **state-translation functions** t_i and t_g are **modular**, i.e., for $\Sigma = \Sigma_1 \cup \Sigma_2$, $S \subseteq \widehat{\Sigma}$, and $S \not\models \bot$, the functions t_x (for $x = i, g$) satisfy

$$t_x(\Sigma, S) = t_x(\Sigma_1, S \cap \widehat{\Sigma_1}) \cup t_x(\Sigma_2, S \cap \widehat{\Sigma_2}),$$

   and they are polynomial-time computable;
3. the size of the results of f_ξ, f_i, and f_g is polynomial in the size of the arguments.

Although there are no resource bounds on f_ξ, f_i, and f_g in the general case, we are also interested in *efficient compilation schemes*. We say that $\mathbf{f}$ is a **polynomial-time compilation scheme** if f_ξ, f_i, and f_g are polynomial-time computable functions.

In addition to that we measure the size of the corresponding plans in the target formalism. If a compilation scheme $\mathbf{f}$ has the property that for every plan Δ solving an instance Π there exists a plan Δ' solving $F(\Pi)$ such that $||\Delta'|| \leq ||\Delta|| + k$ for some positive integer constant k, $\mathbf{f}$ is a **compilation scheme preserving plan size exactly** (modulo some additive constant). If $||\Delta'|| \leq c \times ||\Delta|| + k$ for positive integer constants c and k, then $\mathbf{f}$ is a **compilation scheme preserving plan size linearly**, and if $||\Delta'|| \leq p(||\Delta||, ||\Pi||)$ for some polynomial p, then $\mathbf{f}$ is a **compilation scheme preserving plan size polynomially**. More generally, we say that a planning formalism $\mathcal{X}$ is **compilable** to formalism $\mathcal{Y}$ (in poly. time, preserving plan size exactly, linearly, or polynomially), if there exists a compilation scheme with the appropriate properties. We write $\mathcal{X} \preceq^x \mathcal{Y}$ in case $\mathcal{X}$ is compilable to $\mathcal{Y}$ or $\mathcal{X} \preceq^x_p \mathcal{Y}$ if the compilation can be done in polynomial time. The super-script x can be 1, c, or p depending on whether the scheme is exactly, linearly, or polynomially plan size preserving, respectively. As is easy to see, all the notions of compilability introduced above are reflexive and transitive.

Proposition 2. *The relations $\preceq^x$ and $\preceq^x_p$ are transitive and reflexive.*

Furthermore, it is obvious that when moving upwards in the diagram displayed in Figure 1, there is always a polynomial-time compilation scheme preserving plan size exactly. If π_i denotes the projection to the i-th argument and $\emptyset$ denotes the function that returns always the empty set, the generic compilation scheme for moving upwards in the partial order is $\mathbf{f} = \langle \pi_1, \emptyset, \emptyset, \pi_2, \pi_2 \rangle$.

Proposition 3. *If $\mathcal{X} \sqsubseteq \mathcal{Y}$, then $\mathcal{X} \preceq^1_p \mathcal{Y}$.*

4 Compilability Preserving Plan Size Exactly

Proposition 3 leads to the question of whether there exist other compilation schemes than those implied by the specialization relation.

First, we will establish that $\mathcal{S}_{\mathcal{LI}}$, $\mathcal{S}_{\mathcal{L}}$, $\mathcal{S}_{\mathcal{I}}$, and $\mathcal{S}$ are polynomial-time compilable into each other preserving plan size exactly. Having a closer look at

Bäckström's [2] equivalence proof for $\mathcal{S}_{\mathcal{LI}}$ and $\mathcal{S}$ using an ESP-reduction, it turns out that the ESP-reduction he specified can be reformulated as a compilation scheme. From that the next theorem is immediate.

Theorem 4. *$\mathcal{S}_{\mathcal{LI}}$, $\mathcal{S}_{\mathcal{I}}$, $\mathcal{S}_{\mathcal{L}}$, and $\mathcal{S}$ are polynomial-time compilable to each other preserving plan size exactly.*

One view on this result is that it does not matter from an expressivity point of view whether we allow for atoms only or permit also negative atoms and it does not matter whether we have complete or partial state specification—provided propositional formulae and conditional effects are not allowed. Interestingly, this view generalizes to the case where conditional effects are allowed.

In proving that, there are two additional complications. Firstly, one must compile conditional effects over partial state specifications to conditional effects over complete state specifications. This is a problem because the condition $A(S, post(o)) = P(S, post(o))$ in the definition of the function R must be tested. Secondly, when compiling a formalism with literals into a formalism that allows for atoms only, the condition $A(S, post(o)) \not\models \bot$ in the definition of R must be taken care of. Nevertheless, it is possible to solve these problems using a polynomial-time compilation preserving plan size exactly.

Theorem 5. *$\mathcal{S}_{\mathcal{LIC}}$, $\mathcal{S}_{\mathcal{LC}}$, $\mathcal{S}_{\mathcal{IC}}$, and $\mathcal{S}_{\mathcal{C}}$ are polynomial-time compilable to each other preserving plan size exactly.*

A summary of the results in this section is given in Figure 2. The two equivalence classes with respect to polynomial-time compilability preserving plan size exactly will be called $\mathcal{S}_{\mathcal{LI}}$- and $\mathcal{S}_{\mathcal{LIC}}$-class, in symbols $[\mathcal{S}_{\mathcal{LI}}]$ and $[\mathcal{S}_{\mathcal{LIC}}]$, naming them after their respective largest elements.

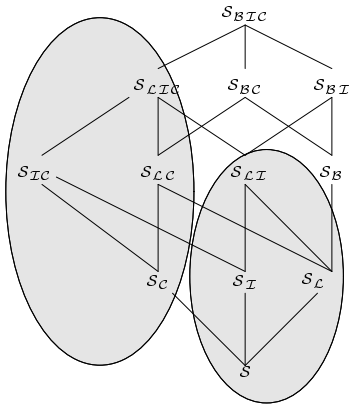

Fig. 2. Equivalence classes of planning formalisms created by polynomial-time compilation schemes preserving plan size exactly

5 The Limits of Compilation when Preserving Plan Size Linearly

In this section, we will show that there are no compilation schemes preserving plan size linearly other than those identified above and those implied by the specialization relation. First of all, we will prove that *conditional effects cannot be compiled away*. The deeper reason for this is that with conditional effects one can independently do a number of things in parallel, which is impossible in formalisms without conditional effects.

In order to illustrate this point, let us consider an example. We start with a set of n propositional atoms $\Sigma_n = \{p_1, \ldots, p_n\}$ and a disjoint copy of this set: $\Sigma_n^\# = \{p_i^\# \mid p_i \in \Sigma_n\}$. Furthermore, if $S \subseteq \widehat{\Sigma_n}$, then $S^\#$ shall denote the corresponding set of literals over $\widehat{\Sigma_n^\#}$. Consider now the following $\mathcal{S}_{\mathcal{LIC}}$ domain structure:

$$\begin{aligned}
\Sigma_{2n} &= \Sigma_n \cup \Sigma_n^\#, \\
\mathbf{O}_{2n} &= \left\{ \langle \top, \{p_i \Rightarrow p_i^\#, \neg p_i \Rightarrow \neg p_i^\# \mid p_i \in \Sigma_n\} \rangle \right\}, \\
\Xi_{2n} &= \langle \Sigma_{2n},\ \mathbf{O}_{2n} \rangle.
\end{aligned}$$

From the construction it follows that for all pairs $(\mathbf{I}, \mathbf{G})$ such that $\mathbf{I}$ is a consistent and complete set over $\widehat{\Sigma_n}$ and $\mathbf{G} \subseteq \mathbf{I}^\#$, the instance $\Pi = \langle \Xi_{2n}, \mathbf{I}, \mathbf{G} \rangle$ has a one-step plan. Conversely, for all pairs $(\mathbf{I}, \mathbf{G})$ with $\mathbf{G} \cap \widehat{\Sigma^\#} \not\subseteq \mathbf{I}^\#$, there does not exist a solution. Constructing polynomially-sized $\mathcal{S}_{\mathcal{BI}}$ domain structures with the same property turns out to be impossible, even if we allow for c-step plans.

Theorem 6. $\mathcal{S}_{\mathcal{LIC}} \not\preceq^c \mathcal{S}_{\mathcal{BI}}$.

Proof Sketch. Assume for contradiction that there exists a compilation scheme that compiles Ξ_{2n} to a $\mathcal{S}_{\mathcal{BI}}$ structure Ξ'_{2n}. Since there are only polynomially many c-step plans, for a large enough n one plan Δ must be used for two different initial states in the target formalism. Since Δ is an unconditional plan, it adds and deletes the same atoms regardless of the initial state. One can then show that all goal atoms not added by Δ must be already present in both initial states, from which one concludes that identical goal states are produced, which means that $\mathbf{f}$ is not solution-preserving and, hence, not a compilation scheme, which is the desired contradiction.

■

Next, we show that *general formulae cannot be compiled to conditional effects* by making use of results from circuit complexity. Assume that the atoms in Σ_n are numbered from 1 to n. Then a word w consisting of n bits could be encoded by the set of literals

$$\begin{aligned}
\mathbf{I}_w = \{p_i \mid &\text{ if the } i\text{th bit of } w \text{ is } 1\} \cup \\
\{\neg p_i \mid &\text{ if the } i\text{th bit of } w \text{ is } 0\}.
\end{aligned}$$

We now say that the n-bit word w is **accepted with a one-step or c-step plan** by $\langle \Sigma_n \cup \{g\}, \mathbf{O}_n \rangle$ iff there exists a one-step or c-step plan, respectively, for the instance $\Pi_w = \langle \langle \Sigma_n \cup \{g\}, \mathbf{O}_n \rangle, \mathbf{I}_w \cup \{\neg g\}, \{g\} \rangle$.

A **uniform polynomially-sized family of domain structures** is an infinite sequence $\boldsymbol{\Xi} = (\Xi_0, \Xi_1, \ldots)$, such that $|\Sigma_i| = i$ and the Ξ_i's can be generated by a $\log i$-space Turing machine. The language accepted by such a family with one-step (or c-step) plans is the set of words accepted using the domain structure Ξ_i for words of length i. Papadimitriou [16, p. 386] has pointed out that the class of languages accepted by *uniform polynomially-sized boolean expressions* is identical to the class of languages accepted by uniform polynomially-sized boolean circuits with logarithmic depth—the class NC^1. From that the next proposition follows immediately.

Proposition 7. *The class of languages accepted by uniform families of $\mathcal{S}_{\mathcal{B}}$ domain structures using one-step plan acceptance is identical to NC^1.*

Theorem 8. $\mathcal{S}_{\mathcal{B}} \not\preceq^c [\mathcal{S}_{\mathcal{LIC}}]$.

Proof Sketch. $\mathcal{S}_{\mathcal{LC}}$-plans with fixed goals and varying initial states can be represented by circuits with unbounded fan-in and constant depth, i.e., by AC^0-circuits. This means that c-step acceptance for families of polynomially-sized $\mathcal{S}_{\mathcal{LC}}$ structures is in AC^0. Assuming now that there exists a compilation scheme leads to the conclusion that non-uniform $\mathsf{AC}^0 \supseteq \mathsf{NC}^1$, which is impossible because of a result by Furst *et al.* [7], who showed that some languages in NC^1 are not in non-uniform AC^0. ∎

Finally, we show that *general formulae together with partial state specifications cannot be compiled away*, even if the plans grow polynomially. Let the **one-step plan existence problem** (1-PLANEX) be the PLANEX problem restricted to plans of size of 1. From the definition of the function R it is evident that $\mathcal{S}_{\mathcal{BIC}}$-1-PLANEX and $\mathcal{S}_{\mathcal{BI}}$-1-PLANEX are coNP-hard. Let p be some fixed polynomial, then the **polynomial step plan-existence problem** (p-PLANEX) is the PLANEX problem restricted to plans that have length bounded by $p(n)$, if n is the size of the planning instance. As is easy to see, this problem is in NP for all formalisms except $\mathcal{S}_{\mathcal{BIC}}$ and $\mathcal{S}_{\mathcal{BI}}$. Based on these observations and using a proof technique introduced by Kautz and Selman [11], we can prove the next result, which depends on the assumption that the *polynomial hierarchy does not collapse*. This assumption is stronger than $\mathsf{P} \neq \mathsf{NP}$, but is nevertheless commonly believed to be true.

Theorem 9. $\mathcal{S}_{\mathcal{BI}} \not\preceq^p \mathcal{S}_{\mathcal{BC}}$ *unless the polynomial hierarchy collapses.*

Proof Sketch. First, we construct a family of $\mathcal{S}_{\mathcal{BI}}$ structures such that for each formula φ of size n an initial state $\mathbf{I}_\varphi$ can be constructed such that $\langle \Xi_n, \mathbf{I}_\varphi, \{g\} \rangle$ has a one-step plan iff φ is unsatisfiable.

Given a set of n atoms, denoted by P_n, we define the set of clauses $\mathbf{A}_n$ to be the set containing all clauses with three literals that can be built using these

atoms. The size of $\mathbf{A}_n$ is $O(n^3)$, i.e., polynomial in n. Let $\mathbf{D}_n$ be a set of new atoms $p_{\{l_1,l_2,l_3\}}$ corresponding one-to-one to the clauses in $\mathbf{A}_n$. Furthermore, let

$$\Phi_n = \bigwedge \left\{ (l_1 \vee l_2 \vee l_3 \vee p_{\{l_1,l_2,l_3\}}) \mid \{l_1, l_2, l_3\} \in \mathbf{A}_n \right\}.$$

We now construct the family of $\mathcal{S}_{\mathcal{BI}}$ domain structures as follows:

$$\Sigma_n = P_n \cup \mathbf{D}_n \cup \{g\},$$
$$\mathbf{O}_n = \{\langle \{\neg\Phi_n\}, \{\top \Rightarrow g\}\rangle\}.$$

Let $\mathbf{C}$ be a function that determines for all 3CNF formulae φ, which atoms in $\mathbf{D}_n$ correspond to the clauses in the formula , i.e., $\mathbf{C}(\varphi) = \{p_{\{l_1,l_2,l_3\}} | \{l_1, l_2, l_3\} \in \varphi\}$. Now, the initial state for any particular formula φ of size n can be computed by $\mathbf{I}_\varphi = \neg\mathbf{C}(\varphi) \cup (\mathbf{D}_n - \mathbf{C}(\varphi)) \cup \{\neg g\}$. From the construction, it follows that there exists a one-step plan for $\langle \Sigma_n, \mathbf{O}_n, \mathbf{I}_\varphi, \{g\}\rangle$ iff φ is unsatisfiable.

Let us now assume that there exists a compilation scheme $\mathbf{f}$ from $\mathcal{S}_{\mathcal{BI}}$ to $\mathcal{S}_{\mathcal{BC}}$ preserving plan size polynomially. From that we can construct an *nondeterministic, advice-taking Turing machine* with *polynomial advice* that decides unsatisfiability of a 3CNF formula in polynomial time (because $\mathcal{S}_{\mathcal{BC}}$-$p$-PLANEX is in NP). This means that we can decide a coNP-complete problem on a nondeterministic, polynomial advice-taking Turing machine in polynomial time. From that it follows that coNP $\subseteq$ NP/poly, which implies that the polynomial hierarchy collapses [19]. ∎

A summary of the results of this section is given in Table 1. The symbol $\sqsubseteq$

$\preceq^x$	$\mathcal{S}_{\mathcal{BIC}}$	$[\mathcal{S}_{\mathcal{LIC}}]$	$\mathcal{S}_{\mathcal{BC}}$	$\mathcal{S}_{\mathcal{BI}}$	$\mathcal{S}_{\mathcal{B}}$	$[\mathcal{S}_{\mathcal{LI}}]$
$\mathcal{S}_{\mathcal{BIC}}$	$=$	$\not\preceq^p$ (9)	$\not\preceq^p$ (9)	$\not\preceq^c$ (6)	$\not\preceq^p$ (9)	$\not\preceq^p$ (9)
$[\mathcal{S}_{\mathcal{LIC}}]$	$\sqsubseteq$	$=$	$\sqsubseteq$	$\not\preceq^c$ (**6**)	$\not\preceq^c$ (6)	$\not\preceq^c$ (6)
$\mathcal{S}_{\mathcal{BC}}$	$\sqsubseteq$	$\not\preceq^c$ (8)	$=$	$\not\preceq^c$ (6)	$\not\preceq^c$ (6)	$\not\preceq^c$ (8)
$\mathcal{S}_{\mathcal{BI}}$	$\sqsubseteq$	$\not\preceq^p$ (9)	$\not\preceq^p$ (**9**)	$=$	$\not\preceq^p$ (9)	$\not\preceq^p$ (9)
$\mathcal{S}_{\mathcal{B}}$	$\sqsubseteq$	$\not\preceq^c$ (**8**)	$\sqsubseteq$	$\sqsubseteq$	$=$	$\not\preceq^c$ (8)
$[\mathcal{S}_{\mathcal{LI}}]$	$\sqsubseteq$	$\sqsubseteq$	$\sqsubseteq$	$\sqsubseteq$	$\sqsubseteq$	$=$

Table 1. Separation Results

means that there exists a compilation scheme because the first formalism is a specialization of the second one. In all the other cases, we specify the separation and give the theorem number from which this result can be deduced, perhaps by applying Prop. 2 and 3. As can be seen, we have shown that for all pairs of formalisms for which we have not identified a polynomial-time compilation scheme preserving plan size exactly, a compilation scheme is impossible even if we allow for linear growth of the plan and unbounded computational resources.

There is of course the question whether compilation schemes preserving plan size polynomially are possible. In the long version of the paper [14] we show

that between all pairs of formalisms for which such compilation schemes have not been ruled out, polynomial-time compilation schemes preserving plan size polynomially do exist.

6 Summary and Discussion

Motivated by the recent approaches to extend the GRAPHPLAN algorithm [3] to deal with *more expressive planning formalisms*, we asked what the term *expressive power* could mean in this context. One reasonable intuition seems to be that the term *expressive power* refers to how concisely domain structures and the corresponding plans can be expressed. Based on this intuition and inspired by recent work in the area of knowledge compilation [5], we introduced the notion of *compilability* in order to measure the relative expressiveness of planning formalisms.

Using this framework, we analyzed a large range of propositional planning formalisms. The most surprising result of this analysis is that we are able to come up with a complete classification. For each pair of formalisms, we were either able to construct a *polynomial-time compilation scheme* with the required size bound on the resulting plans or we could prove that such a compilation scheme is impossible—even if the computational resources for the compilation process are unbounded. In particular, we showed for the formalisms considered in this paper that incomplete state specifications and literals in preconditions can be compiled to basic STRIPS preserving plan size exactly, and that incomplete state specifications and literals in preconditions and effect conditions can be compiled away preserving plan size exactly, if we have already conditional effects. Moreover, we showed that there are no other compilation schemes preserving plan size linearly except those implied by the specialization relationship and those described above.

These results imply that Gazen and Knoblock's [9] approach to compiling conditional are optimal and demonstrate that adding general formulae and incomplete state specifications on top of that will increase expressive power—and the difficulty of finding a plan—even more.

One question one may ask is what happens if we consider formalisms with propositional formulae that are syntactically restricted, e.g. CNF or DNF formulae. However, we did not include an investigation of such formalisms in our analysis in order to keep it manageable (but see [15]). Another question might be whether *non-modular* state-translation functions could lead to more powerful compilation schemes. While this seems unlikely, it is, nevertheless, an interesting theoretical question.

Acknowledgments

The research reported in this paper was started and partly carried out while the author enjoyed being a visitor at the AI department of the University of New South Wales. Many thanks go to Norman Foo, Maurice Pagnucco, and Abhaya

Nayak and the rest of the AI department for the discussions and cappuccinos. I would also like to thank Birgitt Jenner and Jacobo Toran for some clarifications concerning circuit complexity.

References

1. C. R. Anderson, D. E. Smith, and D. S. Weld. Conditional effects in graphplan. In R. Simmons, M. Veloso, and S. Smith, eds., *Proc. AIPS-98*, p. 44–53. AAAI Press, Menlo Park, 1998.
2. C. Bäckström. Expressive equivalence of planning formalisms. *Artificial Intelligence*, 76(1–2):17–34, 1995.
3. A. L. Blum and M. L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):279–298, 1997.
4. T. Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1–2):165–204, 1994.
5. M. Cadoli and F. M. Donini. A survey on knowledge compilation. *AI Communications*, 10(3,4):137–150, 1997.
6. R. E. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.
7. M. Furst, J. B. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, Apr. 1984.
8. M. R. Garey and D. S. Johnson. *Computers and Intractability—A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA, 1979.
9. B. C. Gazen and C. Knoblock. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In Steel and Alami [18], p. 221–233.
10. S. Kambhampati, E. Parker, and E. Lambrecht. Understanding and extending Graphplan. In Steel and Alami [18], p. 260–272.
11. H. A. Kautz and B. Selman. Forming concepts for fast inference. In *Proc. AAAI-92*, p. 786–793, San Jose, CA, July 1992. MIT Press.
12. J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an ADL subset. In Steel and Alami [18], p. 273–285.
13. V. Lifschitz. On the semantics of STRIPS. In M. P. Georgeff and A. Lansky, eds., *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, p. 1–9, Timberline, OR, June 1986. Morgan Kaufmann.
14. B. Nebel. On the compilability and expressive power of propositional planning formalisms. Technical Report 101, Albert-Ludwigs-Universität, Institut für Informatik, Freiburg, Germany, June 1998.
15. B. Nebel. What is the expressive power of disjunctive preconditions? Technical Report 118, Albert-Ludwigs-Universität, Institut für Informatik, Freiburg, Germany, Mar. 1999.
16. C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, MA, 1994.
17. E. P. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In R. Brachman, H. J. Levesque, and R. Reiter, eds., *Proc. KR-89*, p. 324–331, Toronto, ON, May 1989. Morgan Kaufmann.
18. S. Steel and R. Alami, editors. *Proc. ECP'97*, Toulouse, France, Sept. 1997. Springer-Verlag.
19. C. K. Yap. Some consequences of non-uniform conditions on uniform classes. *Theoretical Computer Science*, 26:287–300, 1983.