Chapter 1

ON THE EXPRESSIVE POWER OF PLANNING FORMALISMS

Conditional Effects and Boolean Preconditions in the STRIPS Formalism

Bernhard Nebel
Institut für Informatik, Albert-Ludwigs-Universität
Freiburg, Germany
nebel@informatik.uni-freiburg.de

Abstract The notion of "expressive power" is often used in the literature on planning. However, it is usually only used in an informal way. In this paper, we will formalize this notion using the "compilability framework" and analyze the expressive power of some variants of STRIPS allowing for conditional effects and arbitrary Boolean formulae in preconditions. One interesting consequence of this analysis is that we are able to confirm a conjecture by Bäckström that preconditions in conjunctive normal form add to the expressive power of propositional STRIPS. Further, we will show that STRIPS with conditional effects is incomparable to STRIPS with Boolean formulae as preconditions. Finally, we show that preconditions in conjunctive normal form do not add any expressive power once we have conditional effects.

Keywords: Action planning, STRIPS, conditional effects, Boolean preconditions, expressiveness, computational complexity

1. INTRODUCTION

The term *expressive power* is often used when planning formalisms are compared.[1] For instance, Pednault (1989) states that the expressiveness of the formalism he proposed is between STRIPS (Fikes and Nilsson, 1971) and the situation calculus (McCarthy and Hayes, 1969), and many people seem to believe that conditional effects increase the expressiveness of STRIPS significantly (Anderson et al., 1998; Kambhampati et al., 1997; Koehler et al., 1997). Further, Anderson et al. (1998) seem to conjecture that STRIPS with conditional ef-

fects is expressively equivalent to STRIPS with conditional effects and Boolean preconditions.[2] However, there are only few approaches that try to address the problem of capturing this term formally.

Bäckström (1995) proposed to measure the expressiveness of planning formalisms using his *ESP-reductions*. These reductions are, roughly speaking, polynomial many-one reductions[3] 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. Furthermore, he conjectured that "disjunctive preconditions most likely increase the expressive power [of STRIPS]."[4] Bäckström's approach, however, has two drawbacks. First, it does not seem to be intuitive to define expressiveness using a resource-limited mapping. Second, it does not seem to be possible to prove negative results.

In order to address these problems, we will use the *compilability framework* (Nebel, 1999a; Nebel, 1998), which is inspired by Bäckström's (1995) approach, by the work of Gazen and Knoblock (1997) to compile conditional effects away, and by the *knowledge compilation* framework (Cadoli and Donini, 1997). The main idea behind this approach is that a language feature does not increase the expressiveness if planning operators containing this language feature can be translated into operators that do not contain this feature without blowing up the operator description too much and without enlarging the resulting plans too much. It differs from Bäckström's (1995) *ESP-reductions* in that we only consider *structured* transformations that do not translate the initial state or goal specification, in that we allow for arbitrary computational power in the transformation, and in that we do not require that translated plans have *exactly* the same length.

Using this approach, we show that Bäckström's conjecture that preconditions in conjunctive normal form (CNF) are more expressive than basic STRIPS is indeed correct. We also prove that CNF preconditions do not add to the expressive power if we have already conditional effects – proving a weak version of Anderson's *et al.* (1998) conjecture. However, general Boolean formulae as preconditions are incomparable to conditional effects. From that it also follows that Gazen and Knoblock's (1997) preprocessing scheme for conditional effects is optimal in the sense that we always get a super-polynomial blowup when translating conditional effects to basic STRIPS, provided we require that the plans do not grow more than linearly.

Although these results are purely theoretical, they also have some significance for designing planning algorithms. Provided we can prove that a language feature can be "compiled away" easily, i.e., that it can be regarded as "syntactic sugar," a planning algorithm for the original language can be easily extended to deal with the new feature. Conversely, if we can prove that a language feature cannot be compiled away, we most probably will have significant problems

when we want to extend the planning algorithm for the original language to deal with the new language feature.

The rest of the paper is structured as follows. The next section introduces general terminology and definitions. In Section 3. we introduce the notion of compilability between planning formalisms. Using this framework, we show in Section 4. that conditional effects cannot be compiled away. We then analyze the relationship between STRIPS with Boolean preconditions and basic STRIPS in Section 5.. We show that DNF preconditions can be compiled away and that Bäckström's (1995) conjecture holds in the compilability framework. In Section 6. we show that CNF preconditions can be compiled away if we have conditional effects, but that general Boolean preconditions cannot be compiled to conditional effects. Finally, in Section 7. we summarize and discuss the results.

2. PROPOSITIONAL PLANNING FORMALISMS

Let Σ be a finite set of *propositional atoms* and $\widehat{\Sigma}$ be the set consisting of the constants $\top$ (denoting truth) and $\bot$ (denoting falsity) as well as atoms and negated atoms, i.e., the *literals*, over Σ. The set of all *Boolean formulae* over Σ is denoted by $\mathbf{B}_\Sigma$. The set of *formulae in conjunctive normal form (CNF)* over Σ is denoted by $\mathbf{C}_\Sigma$ and the set of *formulae in disjunctive normal form (DNF)* over Σ by $\mathbf{D}_\Sigma$. Finally, by $\mathbf{L}_\Sigma$ we refer to the set of *formulae that are conjunctions of literals* over Σ, and the set of *formulae that are conjunctions of atoms* is denoted by $\mathbf{A}_\Sigma$. In general, we will use the symbol $\mathcal{L}_\Sigma$ to refer to a possibly restricted language over Σ.

Given a set of literals $L \subseteq \widehat{\Sigma}$, by $pos(L)$ we refer to the *positive literals* in L, by $neg(L)$ we refer to the *negative literals* in L, and $\neg L$ denotes the *element-wise negation* of the literals in L. *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 an element of $\mathcal{L}_\Sigma$. The set *post*, the set of *postconditions*, consists of *conditional effects* each having the form $\varphi \Rightarrow L$, where $\varphi \in \mathcal{L}_\Sigma$ is called *effect condition* and the elements of $L \subseteq \widehat{\Sigma}$ are called *effects*. If all postconditions of an operator have the form $\top \Rightarrow L$, then we say that the operator is *unconditional* and we write the postconditions as a set of literals containing all *effects*.

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. By $\bot$ we represent the *illegal state*. Given a state S and an operator o, we define the *active effects* $A(S, o)$ as follows:

$$A(S, o) \quad = \quad \bigcup \{L \mid (\varphi \Rightarrow L) \in post(o), S \models \varphi\}.$$

Using this function, the result of executing operator o in state S can be specified as:

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

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 $\mathbf{O}$,
- $\mathbf{I} \subseteq \Sigma$ is the *initial state*, 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 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 S is recursively defined as follows:

$$\begin{aligned} Res(S, \langle\rangle) &= S, \\ Res(S, \langle o_1, o_2, \ldots, o_n \rangle) &= Res(R(S, o_1), \langle o_2, \ldots, o_n \rangle). \end{aligned}$$

A sequence of operators Δ is said to be a *plan for* Π or a *solution of* Π iff

1. $Res(\mathbf{I}, \Delta) \neq \bot$ and
2. $Res(\mathbf{I}, \Delta) \models \mathbf{G}$.

The most general planning language we consider in this paper is $\text{STRIPS}_B^{\mathcal{C}}$, which permits conditional effects and general Boolean formulae in preconditions and effect conditions. Without the index $\mathcal{C}$, we refer to planning languages without conditional effects, i.e., all conditional effects have the form $\top \Rightarrow L$. If we have C, D or L instead of B, we refer to languages that permit only for CNF or DNF formulae or conjunctions of literals in preconditions and effect conditions, respectively. The language STRIPS (without any index), finally, is identical to basic STRIPS, i.e., it requires that all preconditions are conjunctions of atoms and all effects are unconditional. In this paper, however, we assume that all formulae may contain literals, i.e., the least expressive language we consider is STRIPS_L.[5] In Figure 1.1, the partial order induced by the syntactic restrictions is shown. 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 diagram depicting the partial order.

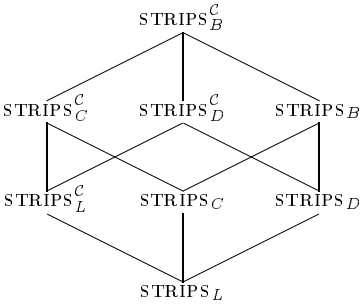

Figure 1.1 Partial order of STRIPS-variants induced by syntactic restrictions

While one would expect that planning in STRIPS_L is much easier than planning in $\text{STRIPS}_B^{\mathcal{C}}$, it turns out that this is not the case, provided one takes a computational complexity perspective. The plan existence problem (PLANEX) is PSPACE-complete for all the formalisms we consider in this paper.

Theorem 1 *$\mathcal{X}$-PLANEX is PSPACE-complete for all $\mathcal{X}$ with $\text{STRIPS}_L \sqsubseteq \mathcal{X} \sqsubseteq \text{STRIPS}_B^{\mathcal{C}}$.*

Proof. PSPACE-hardness of STRIPS_L-PLANEX follows from Corollary 3.2 of Bylander's (1994) analysis. Membership of $\text{STRIPS}_B^{\mathcal{C}}$-PLANEX in PSPACE follows because guessing a plan step by step and storing each state can be done in polynomial space. Further, each plan step can be verified in polynomial time. Hence, the problem is in NPSPACE, which is identical to PSPACE. ■

3. COMPILATION SCHEMES

We will consider a planning formalism $\mathcal{X}$ *as expressive as* another formalism $\mathcal{Y}$ if planning domains and plans formulated in formalism $\mathcal{Y}$ are *concisely expressible* in $\mathcal{X}$. In order to say that something in one formalism $\mathcal{Y}$ can be *expressed* in another formalism $\mathcal{X}$, there must exist a *mapping* from $\mathcal{Y}$-objects to $\mathcal{X}$-objects that *preserves* some important properties. The mapping needs not to be computable, though, because expressibility does not mean that there is an easy way to transform the objects from one formalism to another. In particular, if we want to prove negative results, we better show that there exists no mapping whatsoever.

While the computational resources of the mapping are irrelevant, the size of the result is important, provided we want to express something *concisely*. In the following, we will only consider mappings with results that are *polynomially bounded in size*.

Finally, we have to make up our mind what this mapping should preserve. Certainly, it should preserve *solution existence*. Moreover, since we required

above that planning domain and plans should be expressed concisely, the mapping should preserve the *length of a plan* to a certain degree. We could, of course, require more than that. For instance, we may want to require that there is a function that maps plans in the target formalism back to plans in the source formalism. As we will see, however, such functions can be constructed in all cases we are interested in.

If we now use mappings from $\mathcal{Y}$ planning instances to $\mathcal{X}$ instances that (1) preserve solution existence, (2) do not blow up the plans too much, and (3) have a polynomially sized result, we get the somewhat counter-intuitive result that all the planning formalisms we introduced above have the same expressiveness. The reason is that the mapping can be constructed as follows. Because it is not limited in its computational power, it can test whether there exists a plan for the original $\mathcal{Y}$ instance. If so, it generates an $\mathcal{X}$ instance that has a plan of the same length. Otherwise it constructs an unsolvable $\mathcal{X}$ instance.

This problem could be circumvented by requiring that the computational resources of the mapping are limited, for instance, to polynomial time. In this case, the mapping would be a variation of Bäckström's (1995) ESP-reduction. However, then the question is what the right resource bound is.

Having a closer look at the mapping reveals that it does not measure expressiveness at all but solves the planning instance. However, when we talk about the expressiveness of planning formalisms, we usually mean the expressiveness of the language that is used to specify operators. So the right way to go seems to be to require that the domain structure is transformed independently from the initial state and goal description. And this is, in fact, what we will do. Mappings between domain structures can be viewed as *compiling* the domain structure off-line and enabling us to use the compiled domain structure for different pairs of initial states and goals. For this reason, we will call these mappings *compilation schemata*[6]

A compilation schema maps a $\mathcal{Y}$ domain structure Ξ into an $\mathcal{X}$ domain structure Ξ', which is only polynomially larger than Ξ. In addition, the compilation schema may introduce some auxiliary propositional atoms that are used to control the execution of newly introduced operators. These atoms should most likely have an initial value and may appear in the goal specification of planning instances in the target formalism. Finally, it is required that the resulting minimal $\mathcal{X}$-plan Δ' for Ξ' is bounded in length by the length of the minimal plan Δ for Ξ (see Figure 1.2).

Although Figure 1.2 gives a good picture of the *compilation framework*, it is not completely accurate. If we want to compile a formalism that permits for literals in preconditions and goals to one that requires atoms, some trivial translations of the initial state and goal description are necessary. Similarly, if we want to compile a formalism that permits us to use partial states to a formalism that requires complete state, a translation of the initial state specification is

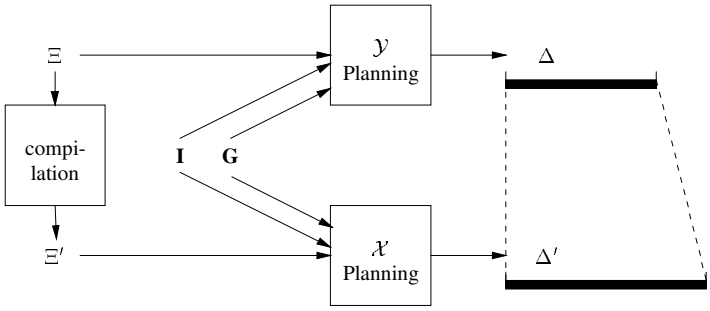

Figure 1.2 The compilation framework

necessary. However, since we do not deal with incomplete state specifications or planning formalisms that allow only atoms in the precondition, we can ignore this issue here.[7]

Let us now define formally what a compilation scheme is. A *compilation scheme from* $\mathcal{X}$ *to* $\mathcal{Y}$ is a tuple of functions $\mathbf{f} = \langle f_\xi, f_i, f_g \rangle$ 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:

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

and satisfies the following conditions:

1. there exists a plan for Π iff there exists a plan for $F(\Pi)$,
2. and 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 an 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 $\mathcal{X} \preceq_p^x \mathcal{Y}$ if the compilation can be done in polynomial time. The super-script x can be 1, c, or p depending on whether the scheme preserves plan size exactly, linearly, or polynomially, respectively.

From a practical point of view, one could regard compilability preserving *plan size exactly* or *linearly* as an indication that the planning formalism we use as the target formalism is *at least as expressive* as the source formalism. Conversely, if a *super-linear* (be it polynomial or super-polynomial) blowup of the plans in the target formalism is required by any compilation scheme, this is an indication that the source formalism *is more expressive* than the target formalism – since it indicates that a planning algorithm for the target formalism would be forced to generate significantly longer plans for compiled instances, making it probably infeasible to try to solve such instances. However, compilations preserving plan size polynomially may nevertheless be of practical value (see also Section 7.).

As is easy to see, all the notions of compilability introduced above are reflexive and transitive, i.e., compilability induces a pre-order on planning formalisms.

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

Furthermore, it is obvious that when moving upwards in the diagram displayed in Figure 1.1, there is always a polynomial-time compilation scheme preserving plan size exactly. If π_i denotes the projection to the i-th argument and $\emptyset$ the function that returns always the empty set, the generic compilation scheme for moving upwards in the partial order is $\mathbf{f} = \langle \pi_1, \emptyset, \emptyset \rangle$.

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

Now there is, of course, the question whether there are more compilability relationships than those implied by the two propositions. This is the question, we will analyze in the next three sections.

4. COMPILING CONDITIONAL EFFECTS AWAY

Gazen and Knoblock (1997) proposed a particular way of transforming $\text{STRIPS}_L^{\mathcal{C}}$ domains to STRIPS_L domains, which blows up the operator set exponentially. There is, of course, the question whether there are better transformations. As we will show, this is not the case. A simple counting argument shows that with conditional effects there are more solvable initial state/goal pairs than there are when conditional effects are not allowed.

In order to illustrate this point, let us consider an example. We start with a set of n propositional atoms $\Sigma_n = \{p_1, \ldots, p_n\}$. Consider now the following $\text{STRIPS}_L^{\mathcal{C}}$ domain structure:

$$o_n \quad = \quad \langle \top, \{p_i \Rightarrow \neg p_i, \neg p_i \Rightarrow p_i \mid p_i \in \Sigma_n\} \rangle$$

$$\begin{aligned} \mathbf{O}_n &= \{o_n\}, \\ \Xi_n &= \langle \Sigma_n, \mathbf{O}_n \rangle. \end{aligned}$$

This means that the operator o_n "inverts" the truth value of all atoms. From that it is evident that there exist $O(2^n)$ pairs $(\mathbf{I}, \mathbf{G})$ such that there exists a one-step plan that solves $\langle \Xi_n, \mathbf{I}, \mathbf{G} \rangle$.

Trying to find a STRIPS_L or STRIPS_B domain structure polynomially sized in $||\Xi_n||$ with the same property seems to be impossible, even if we allow for c-step plans. The reason is that there are only polynomially many different c-step plans.

Theorem 4 $\text{STRIPS}_L^{\mathcal{C}} \not\preceq^c \text{STRIPS}_B$.

Proof. Assume for contradiction that there exists a compilation scheme $\mathbf{f}$ from $\text{STRIPS}_L^{\mathcal{C}}$ to STRIPS_B preserving plan size linearly, which compiles the $\text{STRIPS}_L^{\mathcal{C}}$ domain structure Ξ_n defined above into the STRIPS_B domain structure

$$f_\xi(\Xi_n) = \Xi'_n = \langle \Sigma'_n, \mathbf{O}'_n \rangle.$$

Let us now consider all pairs of initial states and goals $(\mathbf{I}, \mathbf{G})$ such that $\mathbf{G}$ is the "negation" of $\mathbf{I}$, i.e.,

$$\mathbf{G} = \neg\mathbf{I} \cup (\Sigma - \mathbf{I}).$$

For these pairs, there exist obviously one-step plans. By assumption, the STRIPS_B instance

$$\langle \Xi'_n, \mathbf{I} \cup f_i(\Xi_n), \mathbf{G} \cup f_g(\Xi_n) \rangle$$

should then have a c-step plan. Since there are only $O(|\mathbf{O}'_n|^c)$ different c-step plans, which is a number polynomial in the size of Ξ_n, the same plan Δ is used for different $(\mathbf{I}, \mathbf{G})$ pairs—provided n is sufficiently large.

Suppose the plan Δ is used for the pairs $(\mathbf{I}'_1, \mathbf{G}'_1)$, $(\mathbf{I}'_2, \mathbf{G}'_2)$, which result from $(\mathbf{I}_1, \mathbf{G}_1)$ and $(\mathbf{I}_2, \mathbf{G}_2)$. Since $\mathbf{I}_1 \neq \mathbf{I}_2$, $\mathbf{I}_1$ and $\mathbf{I}_2$ must differ on at least one atom, say p. Without loss of generality we assume $p \in \mathbf{I}_1$ and $p \notin \mathbf{I}_2$. Since Δ is a successful plan from $\mathbf{I}'_1$ to $\mathbf{G}'_1$, it follows that $Res(\mathbf{I}'_1, \Delta) \models \neg p$. Similarly, we have $Res(\mathbf{I}'_2, \Delta) \models p$. Since Δ is a plan without conditional effects it always adds and deletes the same atoms. Further, since we assumed $p \in \mathbf{I}_1$ (hence,$p \in \mathbf{I}'_1$), the plan must delete p and not reestablish it. Then, however, it is impossible that we have $Res(\mathbf{I}'_2, \Delta) \models p$, which is a contradiction.

This means, our initial assumption that there exists a compilation scheme from $\text{STRIPS}_L^{\mathcal{C}}$ to STRIPS_B preserving plan size linearly must be wrong. Such a scheme cannot exist. ■

In other words, conditional effects are essential and they cannot be compiled away even if we allow for general Boolean preconditions and a linear increase in plan length.

5. COMPILING BOOLEAN PRECONDITIONS AWAY

Now the question is whether we can compile Boolean preconditions away. We start with restricted forms of Boolean preconditions and continue to move the partial order displayed in Figure 1.1 upwards.

It is folklore in the planning community that DNF preconditions can be regarded as syntactic sugar. For each operator $o = \langle (c_1 \vee c_2 \vee \ldots \vee c_k), L \rangle$, where $c_i \in \mathbf{L}_\Sigma$ and $L \subseteq \widehat{\Sigma}$, one simply generates k new operators $o_i = \langle c_i, L \rangle$. This translation is obviously a *polynomial-time compilation scheme preserving plan size exactly.*

Proposition 5 $\text{STRIPS}_D \preceq_p^1 \text{STRIPS}_L$.

For CNF preconditions the situation is much less clear. As mentioned above, Bäckström (1995) conjectured that CNF preconditions add to the expressiveness of STRIPS_L, but he was not able to prove this conjecture using his framework of ESP-reductions. Using our compilability framework for measuring expressiveness, we can, however, prove his conjecture. In order to do so, we first introduce a variation of the planning problem.

The *fixed plan-size initial-state existence problem* (c-FISEX) is defined as follows. Given a domain structure $\Xi = \langle \Sigma, \mathbf{O} \rangle$, a goal $\mathbf{G} \subseteq \widehat{\Sigma}$, a state $\mathbf{I} \subseteq \Sigma$, and a subset of the atoms called *choice set* $\mathbf{C} \subseteq \Sigma$, the question is whether there exists a set $C \subseteq \mathbf{C}$ such that $\langle \Xi, \mathbf{I} \cup C, \mathbf{G} \rangle$ can be solved by a plan with size c, where c is a positive constant.

Although this problem appears to be slightly harder than the ordinary plan existence problem, fixing the plan length to the positive constant c makes the problem easy, at least for the planning formalism STRIPS_L.

Theorem 6 STRIPS_L-c-FISEX *can be decided in polynomial time.*

Proof. Given an STRIPS_L-c-FISEX instance $(\Xi, \mathbf{I}, \mathbf{G}, \mathbf{C})$, the number of possible operator sequences is $O(|\mathbf{O}|^c)$, which is polynomial in the instance size. For each such sequence, we can do a regression analysis starting with the goal $\mathbf{G}$ computing the weakest precondition, which is a set of literals. This can be done in polynomial time. Finally, one can easily check in polynomial time, whether there is a subset $C \subseteq \mathbf{C}$ that leads to an initial state $\mathbf{I} \cup C$ that entails the weakest precondition. This means, the problem can be solved in polynomial time for any fixed c. ■

If we allow for CNF preconditions, the problem becomes harder. Even if the plan length is restricted to one, 3SAT can be obviously reduced to the 1-FISEX problem in STRIPS_C.

Proposition 7 STRIPS_C-*1*-FISEX *is* NP-*complete.*

From that it follows immediately that there cannot exist any *polynomial-time* compilation scheme from STRIPS_C to STRIPS_L preserving plan size linearly—provided $\mathsf{P} \neq \mathsf{NP}$.[8] We will show a stronger result, namely that there cannot exist *any* compilation scheme preserving plan size linearly by employing a proof technique first used by Kautz and Selman (1992).

In order to prove this result, we need the notion of *advice-taking Turing machines*. These are machines with an *advice oracle*, which is a (not necessarily recursive) function a from positive integers to bit strings. On input I, the machine loads the bit string $a(||I||)$ and then continues as usual. Note that the oracle derives its bit string only from the length of the input and not from the contents of the input. An advice is said to be a *polynomial advice* if the oracle string is polynomially bounded by the instance size. Further, if X is a complexity class defined in terms of resource-bounded machines, e.g., P or NP, then $\mathsf{X/poly}$ (also called *non-uniform* X) is the class of problems that can be decided on machines with the same resource bounds and polynomial advice.

Because of the advice oracle, the class $\mathsf{P/poly}$ appears to be much more powerful than P. However, it seems unlikely that $\mathsf{P/poly}$ contains all of NP. In fact, one can prove that $\mathsf{NP} \subseteq \mathsf{P/poly}$ implies certain relationships between uniform complexity classes that are believed to be very unlikely. In particular, Karp and Lipton (1980) have shown that $\mathsf{NP} \subseteq \mathsf{P/poly}$ implies that the polynomial hierarchy collapses on the second level—which is considered to be very unlikely.

Theorem 8 $\text{STRIPS}_C \not\preceq^c \text{STRIPS}_L$*, unless the polynomial hierarchy collapses.*

Proof. As a first step, we construct for each n a STRIPS domain structure Ξ_n, a choice set $\mathbf{C}_n$, and a goal specification $\mathbf{G}_n$ with size polynomial in n and the following properties. Satisfiability of an arbitrary 3CNF formula φ_n of size n is equivalent to 1-step plan existence for the STRIPS_C-1-FISEX instance $(\Xi_n, \mathbf{G}_n, \mathbf{I}_{\varphi_n}, \mathbf{C}_n)$, where $\mathbf{I}_{\varphi_n}$ can be computed in polynomial time from φ_n.

Given a set of n atoms, denoted by $\mathbf{P}_n$, 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 new atoms p_γ corresponding one-to-one to the clauses γ in $\mathbf{A}_n$. Finally, let s be a new atom which is not in $\mathbf{P}_n \cup \mathbf{D}_n$.

Now we construct a STRIPS_C domain structure $\Xi_n = \langle \Sigma_n, \mathbf{O}_n \rangle$ goal $\mathbf{G}_n$, and the choice set $\mathbf{C}_n$ as follows:

$$
\begin{aligned}
\Sigma_n &= \mathbf{P}_n \cup \mathbf{D}_n \cup \{\mathsf{s}\}, \\
\mathbf{O}_n &= \{o_n\}, \\
o_n &= \langle (\bigwedge_{\gamma \in \mathbf{A}_n} (p_\gamma \vee \gamma)), \{\mathsf{s}\} \rangle, \\
\mathbf{G}_n &= \{\mathsf{s}\},
\end{aligned}
$$

$$\mathbf{C}_n \quad = \quad \mathbf{P}_n.$$

Let $cl(\varphi)$ be the set of clauses appearing in φ. Based on that we define $\mathbf{I}_{\varphi_n}$ as follows:

$$\mathbf{I}_{\varphi_n} \quad = \quad \{p_\gamma \in \mathbf{D}_n \mid \gamma \notin cl(\varphi_n)\}.$$

Now it is easy to see that φ_n is satisfiable iff for the STRIPS_C-1-FISEX instance $(\Xi_n, \mathbf{G}_n, \mathbf{I}_{\varphi_n}, \mathbf{C}_n)$ there exists a set of choices $C \subseteq \mathbf{C}_n$ such that the resulting planning instance $\langle \Xi_n, \mathbf{I}_{\varphi_n} \cup C, \mathbf{G}_n \rangle$ is solved by a one-step plan.

Let us now assume that there exists a compilation scheme from STRIPS_C to STRIPS_L preserving plan size linearly. Using this compilation scheme, we compile the STRIPS_C domain structure Ξ_n into the STRIPS_L domain structure $\Xi'_n = \langle \Sigma'_n, \mathbf{O}'_n \rangle$. Further, $\mathbf{G}'_n$, $\mathbf{I}'_{\varphi_n}$, and $\mathbf{C}'_n$ are defined as follows:

$$\begin{aligned} \mathbf{G}'_n &= \{\mathsf{s}\} \cup f_g(\Xi_n), \\ \mathbf{I}'_{\varphi_n} &= \mathbf{I}_{\varphi_n} \cup f_i(\Xi_n), \\ \mathbf{C}'_n &= \mathbf{C}_n. \end{aligned}$$

Note that all these sets can be computed in time polynomial in n, once we know the values of $f_g(\Xi_n)$ and $f_i(\Xi_n)$.

From the construction, it follows that the following statements are equivalent:

1. φ_n is satisfiable,
2. for the STRIPS_L-1-FISEX instance $(\Xi_n, \mathbf{G}_n, \mathbf{I}_{\varphi_n}, \mathbf{C}_n)$, there exists a set of choices $C \subseteq \mathbf{C}_n$ such that $\Pi = \langle \Sigma_n, \mathbf{O}_n, \mathbf{I}_{\varphi_n} \cup C, \mathbf{G}_n \rangle$ has a one-step plan,
3. for the STRIPS_L-c-FISEX instance $(\Xi'_n, \mathbf{G}'_n, \mathbf{I}'_{\varphi_n}, \mathbf{C}'_n)$, there exists a set of choices $C' \subseteq \mathbf{C}'_n$ such that $\Pi' = \langle \Sigma'_n, \mathbf{O}'_n, \mathbf{I}'_{\varphi_n} \cup C', \mathbf{G}'_n \rangle$ has a c-step plan,

One can now construct an advice-taking Turing machine that on input of a formula φ_n of size n loads the polynomial advice $\langle \Xi'_n, f_g(\Xi_n), f_i(\Xi_n) \rangle$ and then decides STRIPS_L-c-FISEX for the instance $(\Xi'_n, \mathbf{G}'_n, \mathbf{I}'_{\varphi_n}, \mathbf{C}'_n)$, which by Theorem 6 can be done in polynomial time. Since the problem 3SAT, which is solved by this deterministic, advice-taking machine in polynomial time is NP-complete, we conclude that $\mathsf{NP} \subseteq \mathsf{P/poly}$. This implies by Karp and Lipton's (1980) result that the polynomial hierarchy collapses on the second level, which proves the claim. ■

The result above implies that adding CNF preconditions to STRIPS_L adds to the expressiveness. However, it is not immediately obvious whether a further generalization from CNF formulae to arbitrary Boolean formulae would add another level of expressiveness. We will defer this question to the next section.

6. COMPILING BOOLEAN PRECONDITIONS INTO CONDITIONAL EFFECTS

As mentioned in the Introduction, sometimes the expressive power of conditional effects and of Boolean preconditions are claimed to be related. In this section, we will analyze this claim using the compilability framework.

As in the case of unconditional actions, it is commonly agreed that DNF formulae can be regarded as "syntactic sugar." Any operator containing a DNF precondition with k disjuncts can be split into k new operators containing only conjunctions of literals in the precondition. Similarly, any conditional effect with a DNF effect condition $c_1 \vee \ldots \vee c_n \Rightarrow L$ can be equivalently expressed by a set of n conditional effects $c_i \Rightarrow L$. Obviously, this transformation can be viewed as a polynomial-time compilation scheme preserving plan size exactly.

Proposition 9 $\text{STRIPS}^{\mathcal{C}}_{D} \preceq^{1}_{p} \text{STRIPS}^{\mathcal{C}}_{L}$.

Interestingly, CNF preconditions and effect conditions do not appear to add to the expressive power once we have conditional effects—provided we accept that two formalisms have the same expressive power, if they are compilable to each other preserving plan size *linearly*. The main idea behind proving this is that operators with conditional effects can be used to evaluate the truth of clauses.

Theorem 10 $\text{STRIPS}^{\mathcal{C}}_{C} \preceq^{c}_{p} \text{STRIPS}^{\mathcal{C}}_{L}$.

Proof. Assume that the operators of the $\text{STRIPS}^{\mathcal{C}}_{C}$ domain structure $\Xi = \langle \Sigma, \mathbf{O} \rangle$ contain n clauses $\gamma_1, \gamma_2, \ldots, \gamma_n$ with $\gamma_i = l_{i1} \vee \ldots \vee l_{ik_i}$ in preconditions and effect conditions. For each clause γ_i, a new atom p_{γ_i} is introduced, and the set of these new atoms is denoted by Γ. Now, the operator $eval$, which will evaluate the truth values of all the clauses in a given state, can be defined as follows:

$$eval = \langle \top, \{l_{ij} \Rightarrow p_{\gamma_i}\} \rangle.$$

If all clauses γ_i in $\mathbf{O}$ are replaced by the new atoms p_{γ_i}—leading to the new set $\widehat{\mathbf{O}}$—the only remaining changes that are necessary are that we enforce that the $eval$ operator is always executed before an operator from $\widehat{\mathbf{O}}$ is executed and that all operators from $\widehat{\mathbf{O}}$ set all the atoms from Γ to false.

In order to enforce sequences of operators alternating between operators from $\widehat{\mathbf{O}}$ and the $eval$-operator, one can introduce a new atom e that is added to the initial state. In addition, we modify the $eval$ operator and all operators $\widehat{o} \in \widehat{\mathbf{O}}$ as follows:

$$\begin{aligned} eval' &= \langle \mathsf{e},\, \mathit{post}(eval) \cup \{\top \Rightarrow \{\neg \mathsf{e}\}\} \rangle \\ o' &= \langle \neg \mathsf{e} \wedge \mathit{pre}(\widehat{o}),\, \mathit{post}(\widehat{o}) \cup \{\top \Rightarrow \{\mathsf{e}\}\} \cup \{\top \Rightarrow \neg\Gamma\} \rangle. \end{aligned}$$

We can now specify a compilation scheme from $\text{STRIPS}_C^{\mathcal{C}}$ to $\text{STRIPS}_L^{\mathcal{C}}$ as follows:

$$\begin{array}{lrcl} f_\xi\colon & \langle \Sigma, \mathbf{O} \rangle & \mapsto & \langle \Sigma \cup \Gamma \cup \{\mathsf{e}\}, \{o' | \widehat{o} \in \widehat{\mathbf{O}}\} \cup \{eval'\} \rangle, \\ f_i\colon & \langle \Sigma, \mathbf{O} \rangle & \mapsto & \{\mathsf{e}\}, \\ f_g\colon & \langle \Sigma, \mathbf{O} \rangle & \mapsto & \emptyset. \end{array}$$

This is obviously a polynomial-time compilation scheme that leads to $\text{STRIPS}_L^{\mathcal{C}}$ plans that are twice as long as the original $\text{STRIPS}_C^{\mathcal{C}}$ plans. ■

This result appears to be relevant for practical planning algorithms because it suggests how to extend planning algorithms for conditional operators to algorithms for dealing with CNF preconditions and effect conditions. However, one may wonder whether we can improve on this result, coming up with a compilation scheme preserving plan size exactly. Interestingly, there does not appear to be an obvious way to do that. Further, it is completely unclear how to prove that such a compilation scheme is impossible.

Having shown that CNF preconditions can be compiled away when conditional effects are present, one might hope that this can also be done with general Boolean preconditions. Unfortunately, this does not work, though. In order to show that, we need the notion of *Boolean circuits* and *families of circuits* .

A *Boolean circuit* is a directed, acyclic graph $C = (V, E)$, where the nodes V are called *gates*. Each gate $v \in V$ has a type $type(v) \in \{\neg, \vee, \wedge, 1, 0\} \cup \{x_1, x_2, \ldots\}$. The gates with $type(v) \in \{1, 0, x_1, x_2, \ldots\}$ have in-degree zero, the gates with $type(v) \in \{\neg\}$ have in-degree one, and the gates with $type(v) \in \{\wedge, \vee\}$ have in-degree two. All gates except one have at least one outgoing edge. The gate with no outgoing edge is called the *output gate*. The gates with no incoming edges are called the *input gates*. The *depth* of a circuit is the length of the longest path from an input gate to the output gate. The *size* of a circuit is the number of gates in the circuit. Given a *value assignment* to the variables $\{x_1, x_2, \ldots\}$, the circuit computes the value of the output gate in the obvious way.

Instead of using circuits for computing Boolean functions, we can also use them for accepting words of length n in $\{0,1\}^\star$. A word $w = x_1 \ldots x_n \in \{0,1\}^n$ is now interpreted as a value assignment to the n input variables $x_1, \ldots, x_n$ of a circuit. The word is *accepted* iff the output gate has value 1 for this word. In order to deal with words of different length, we need one circuit for each possible length. A *family of circuits* is an infinite sequence $\mathbf{C} = (C_0, C_1, \ldots)$, where C_n has n input variables. The language accepted by such a family of circuits is the set of words w such that $C_{||w||}$ accepts w.

Usually, one considers so-called *uniform* families of circuits, i.e., circuits that can be generated on a Turing machine with a $\log n$-space bound. Sometimes, however, also non-uniform families are interesting. For example, the class of

languages accepted by non-uniform families of polynomially-sized circuits is just the class P/poly introduced in Section 5..

Using restrictions on the size and depth of the circuits, we can now define new complexity classes, which in their uniform variants are all subsets of P. One class that is important in the following is the class of languages accepted by uniform families of circuits with polynomial size and logarithmic depth, named NC^1. Another class which proves to be important for us is defined in terms of non-standard circuits, namely circuits with gates that have *unbounded fan-in*. Instead of restricting the in-degree of each gate to be two at maximum, we now allow an unbounded in-degree. The class of languages accepted by families of polynomially sized circuits with unbounded fan-in and constant depth is called AC^0.

From the definition, it follows almost immediately that $\mathsf{AC}^0 \subseteq \mathsf{NC}^1$. Moreover, it has been shown that there are some languages in NC^1 that are not in the non-uniform variant of AC^0, which implies that $\mathsf{AC}^0 \neq \mathsf{NC}^1$ (Furst et al., 1984).

In order to prove that we cannot compile Boolean preconditions to conditional effects, we will view families of domain structures with fixed goals and fixed size plans as "machines" that accept languages, similar to families of circuits. For all words w consisting of n bits, let

$$\Xi_n = \langle \Sigma_n \cup \{g\}, \mathbf{O}_n \rangle.$$

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 an initial state

$$\mathbf{I}_w = \{p_i \mid \text{ iff the } i\text{th bit of } w \text{ is } 1\}.$$

We now say that the n-bit word w is *accepted with a one-step or c-step plan* by Ξ_n iff there exists a one-step or c-step plan, respectively, for the instance

$$\Pi_n = \langle\langle \Sigma_n \cup \{g\}, \mathbf{O}_n \rangle, \mathbf{I}_w \cup \{\neg g\}, \{g\}\rangle.$$

Similarly to families of circuits, we also define families of domain structures, $\boldsymbol{\Xi} = (\Xi_0, \Xi_1, \ldots)$. The language accepted by such a family with a one-step (or c-step) plan is the set of words accepted using the domain structure Ξ_n for words of length n. Borrowing the notion of uniformity as well, we say that a family of domain structures is *uniform* if it can be generated by a $\log n$-space Turing machine.

Papadimitriou (1994) has pointed out that the languages accepted by *uniform polynomially-sized Boolean expressions* is identical to (uniform) NC^1. As is easy to see, a family of STRIPS_B domain structures is nothing more than a family of Boolean expressions, provided we use one-step plans for acceptance.

Proposition 11 *The class of languages accepted by uniform families of* STRIPS_B *domain structures using one-step plan acceptance is identical to* NC^1.

If we now have a closer look at what the power of c-step plan acceptance for families of $\text{STRIPS}_L^{\mathcal{C}}$ domain structures is, it turns out that it is less powerful than NC^1. In order to show that, we will first prove the following lemma that relates c-step $\text{STRIPS}_L^{\mathcal{C}}$ plans to circuits with gates of unbounded fan-in.

Lemma 12 *Let* $\Xi = \langle \Sigma, \mathbf{O} \rangle$ *be a* $\text{STRIPS}_L^{\mathcal{C}}$ *domain structure, let* $\mathbf{G} \subseteq \widehat{\Sigma}$*, and let* Δ *be a* c*-step plan over* Ξ*. Then there exists a polynomially sized Boolean circuit* C *with unbounded fan-in and depth* $7c + 2$ *such that* Δ *is a plan for* $\langle \Xi, \mathbf{I}, \mathbf{G} \rangle$ *iff the circuit* C *has value 1 for the input* $w_{\mathbf{I}}$.

Proof. The general structure of a circuit for a c-step $\text{STRIPS}_L^{\mathcal{C}}$ plan is displayed in Figure 1.3. For each plan step (or level) j and each atom p_i, there is a

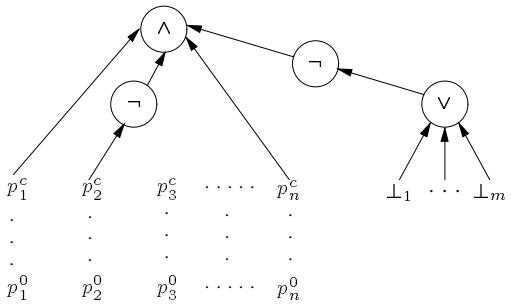

Figure 1.3 Circuit structure and goal testing for a c-step $\text{STRIPS}_L^{\mathcal{C}}$ plan

connection p_i^j. The connections on level 0 are the input gates, i.e., $p_i^0 = x_i$. The goal test is performed by an $\wedge$-gate that checks that all the goals are true on level c, in our case $\mathbf{G} = \{p_1, \neg p_2, p_n\}$. Further, using the $\vee$-gate, it is checked that no inconsistency was generated when executing the plan.

For each plan step j, it must be computed whether the precondition is satisfied and what the result of the conditional effects are. Figure 1.4 (a) displays the precondition test for the precondition $\{p_1, p_2, \neg p_3\}$. If the conjunction of the precondition literals is not true, $\bot_k$ becomes true, which is connected to the $\vee$-gate in Figure 1.3.

Without loss of generality (using a polynomial transformation), we assume that all conditional effects have the form $L \Rightarrow l$. Whether the effect l is activated on level j is computed by a circuit as displayed in Figure 1.4 (b), which shows the circuit for $\{p_1, \neg p_3\} \Rightarrow \neg p_i$.

Finally, all activated effects are combined by the circuit shown in Figure 1.4 (c). For all atoms p_i, we check whether both p_i and $\neg p_i$ have been

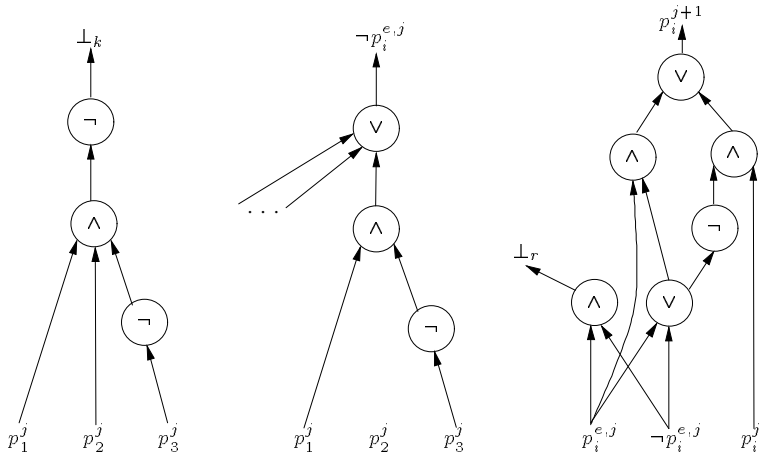

Figure 1.4 Circuit structure for precondition testing (a), conditional effects (b), and the computation of effects (c) for $\text{STRIPS}_L^{\mathcal{C}}$ operators

activated, which would set $\bot_r$ true. This is again one of the inputs of the $\vee$-gate in Figure 1.3. If neither p_i nor $\neg p_i$ have been activated, the value of p_i on level $j+1$ is determined by the value of p_i on level j. Otherwise the value of p_i on level $j+1$ is determined by the value of $p_i^{e,j}$, i.e., the activation value of the positive effect p_i on level j.

The depths of the circuits in Figure 1.4 (b) and (c) dominate the depth of the circuit necessary to represent one plan step leading to the conclusion that a plan step can be represented using a circuit of depth 7. Adding the depth of the goal testing circuit, the claim follows. ■

The lemma implies that $\text{STRIPS}_L^{\mathcal{C}}$ c-step plan acceptance is indeed less powerful than STRIPS_B 1-step plan acceptance, which means that a compilation scheme from STRIPS_B to $\text{STRIPS}_L^{\mathcal{C}}$ preserving plan size linearly is impossible.

Theorem 13 $\text{STRIPS}_B \not\preceq^c \text{STRIPS}_L^{\mathcal{C}}$.

Proof. Assume for contradiction that $\text{STRIPS}_B \preceq^c \text{STRIPS}_L^{\mathcal{C}}$. Let $\boldsymbol{\Xi} = (\Xi_0, \Xi_1, \ldots)$ be a uniform family of STRIPS_B domain structures and $\boldsymbol{\Xi}' = (\Xi_0', \Xi_1', \ldots)$ be the $\text{STRIPS}_L^{\mathcal{C}}$ domain structures generated by a compilation scheme $\mathbf{f}$ that preserves plan size linearly. By Lemma 12 we know that for each $\text{STRIPS}_L^{\mathcal{C}}$ domain structure $\Xi_n' = \langle \Sigma_n', \mathbf{O}_n' \rangle$ and given goal $\mathbf{G}'$ we can generate a polynomially sized, unbounded fan-in circuit with depth $7c+2$ that tests whether a particular c-step plan achieves the goal. In order to decide c-step plan existence, we must test $O(|\mathbf{O}_n'|^c)$ different plans, which is polynomial in the

size of Ξ_n because $\mathbf{f}$ is a compilation scheme. For each plan, we can generate one test circuit, and by adding another $\vee$-gate we can decide c-step plan existence using a circuit with depth $7c + 3$ and size polynomial in the size of Ξ_n. Since by Proposition 11 all languages in NC^1 are accepted by uniform families of STRIPS_B domain structures using one-step plan acceptance, our assumption $\text{STRIPS}_B \preceq^c \text{STRIPS}_L^{\mathcal{C}}$ implies that we can accept all language in NC^1 by (possibly non-uniform) AC^0 circuits, which is impossible by the result of Furst *et al.* (1984). ■

Using the two results above, can now easily give an answer to the question posed in the end of the previous section, namely, whether general Boolean preconditions are more expressive than CNF preconditions.

Theorem 14 $\text{STRIPS}_B \not\preceq^c \text{STRIPS}_C$.

Proof. Assume for contradiction that there is a compilation scheme from STRIPS_B to STRIPS_C preserving plan size linearly. Since by Proposition 3 we have $\text{STRIPS}_C \preceq^c \text{STRIPS}_C^{\mathcal{C}}$ and by Theorem 10 we have $\text{STRIPS}_C^{\mathcal{C}} \preceq^c \text{STRIPS}_L^{\mathcal{C}}$, we can conclude $\text{STRIPS}_B \preceq^c \text{STRIPS}_L^{\mathcal{C}}$ using Proposition 2 twice. This, however, contradicts Theorem 13. ■

This leaves us with the question whether general Boolean preconditions and effect conditions are more expressive than CNF preconditions and effect conditions. However, assuming that $\text{STRIPS}_B^{\mathcal{C}} \preceq^c \text{STRIPS}_C^{\mathcal{C}}$ leads immediately to the conclusion that $\text{STRIPS}_B^{\mathcal{C}} \preceq^c \text{STRIPS}_L^{\mathcal{C}}$ (using Theorem 10 and Proposition 2), which is impossible because of Theorem 13.

Proposition 15 $\text{STRIPS}_B^{\mathcal{C}} \not\preceq^c \text{STRIPS}_C^{\mathcal{C}}$.

7. SUMMARY AND DISCUSSION

Using the *compilability framework* (Nebel, 1998), we analyzed the expressive power of disjunctive preconditions and conditional effects. In general, our results provide a complete classification of the relative expressiveness of STRIPS-like languages with restricted formulae and conditional effects – provided that literals are always allowed and states are always complete. Table 1.1 gives an overview of the results (without the trivial $\text{STRIPS}_B^{\mathcal{C}}$ column). The "$\sqsubseteq$" entries mark syntactic specialization relationships (see Figure 1.1). For all other entries we give the strongest compilability result or impossibility result. The number indicates the theorem from which the result has been derived. If the number is in bold face, it is just the statement of the theorem. Otherwise, the result can be derived from the theorem and the application of Propositions 2 and 3. Two particular interesting results are

$\preceq^x$	STRIPS$_L$	STRIPS$_D$	STRIPS$_C$	STRIPS$_B$	STRIPS$_L^C$	STRIPS$_D^C$	STRIPS$_C^C$
STRIPS$_L$	$=$	$\sqsubseteq$	$\sqsubseteq$	$\sqsubseteq$	$\sqsubseteq$	$\sqsubseteq$	$\sqsubseteq$
STRIPS$_D$	$\preceq_p^1$ **(5)**	$=$	$\preceq_p^1$ (5)	$\sqsubseteq$	$\preceq_p^1$ (5)	$\sqsubseteq$	$\preceq_p^1$ (5)
STRIPS$_C$	$\not\preceq^c$ **(8)**	$\not\preceq^c$ (8,5)	$=$	$\sqsubseteq$	$\preceq_p^c$ (10)	$\preceq_p^c$ (10)	$\sqsubseteq$
STRIPS$_B$	$\not\preceq^c$ (14)	$\not\preceq^c$ (14,5)	$\not\preceq^c$ **(14)**	$=$	$\not\preceq^c$ **(13)**	$\not\preceq^c$ (13,9)	$\not\preceq^c$ (13,10)
STRIPS$_L^C$	$\not\preceq^c$ (4)	$\not\preceq^c$ (4)	$\not\preceq^c$ (4)	$\not\preceq^c$ **(4)**	$=$	$\sqsubseteq$	$\sqsubseteq$
STRIPS$_D^C$	$\not\preceq^c$ (4)	$\not\preceq^c$ (4)	$\not\preceq^c$ (4)	$\not\preceq^c$ (4)	$\preceq_p^1$ **(9)**	$=$	$\preceq_p^1$ (9)
STRIPS$_C^C$	$\not\preceq^c$ (4)	$\not\preceq^c$ (4)	$\not\preceq^c$ (4)	$\not\preceq^c$ (4)	$\preceq_p^c$ **(10)**	$\preceq_p^c$ (10)	$=$
STRIPS$_B^C$	$\not\preceq^c$ (4)	$\not\preceq^c$ (4)	$\not\preceq^c$ (4)	$\not\preceq^c$ (4)	$\not\preceq^c$ (13)	$\not\preceq^c$ (13,9)	$\not\preceq^c$ **(15)**

Table 1.1 Compilability between STRIPS variants

1. CNF preconditions add to the power of basic STRIPS, confirming an earlier conjecture by Bäckström (1995);
2. Conditional effects cannot be compiled away even if we allow for general Boolean preconditions and a linear growth of the resulting plans. This result shows that we cannot improve on the preprocessing scheme for conditional effects as proposed by Gazen and Knoblock (1997).
3. CNF preconditions and CNF effect conditions do not add anything to the expressive power if we already have conditional effects, confirming a weak version of a conjecture by Anderson *et al.* (1998).

In particular the latter result may have practical value for the design of planning algorithms. It suggests that when normalizing preconditions and effect conditions it is not necessary to convert them to disjunctive normal form, but conjunctive normal form is another option that can be easily dealt with. This option may sometimes help to avoid excessive space consumption, provided the formulae are already almost CNF.

Is this all one can say regarding practical issues? As has been pointed out in Section 3., instead of considering only compilation schemes which preserve plan size linearly, we might also be interested in compilation schemes that preserve plan size polynomially. What do we get in this case? Interestingly, from this point of view all the formalisms considered in this paper are expressively equivalent (even if we restrict ourselves to polynomial-time compilations) (Nebel, 1998). However, it should be noted that a polynomial blowup of the plan size implies that a planning algorithm has to cope with a potentially much larger search space – which limits the practical value of this result.

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.

Notes

1. This paper is a revised and extended version of a paper first presented at ECP-99 (Nebel, 1999b).

2. Actually the statement is that "disjunctive preconditions . . . are . . . essential prerequisites for handling conditional effects."

3. We assume that the reader has a basic knowledge of complexity theory (Garey and Johnson, 1979; Papadimitriou, 1994), 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.

4. Bäckström meant *CNF preconditions* when we wrote *disjunctive preconditions*.

5. The reason for leaving out basic STRIPS is that it has already been shown that STRIPS and STRIPS_L as well as STRIPS^C and STRIPS_L^C are equivalent with respect to expressiveness (Nebel, 1998). Furthermore, ignoring the case $\mathcal{L}_\Sigma = \mathbf{A}_\Sigma$ simplifies some of the technical problems when specifying compilation schemes.

6. Note that these compilation schemata are very similar to knowledge compilations, which compile the *fixed part* of a problem in order to speed up the overall processing (Cadoli and Donini, 1997).

7. This means, we do not need the *state-translation functions* as introduced in the definition of a compilation scheme in an earlier paper (Nebel, 1998).

8. Here the difference between Bäckström's (1995) ESP-reductions and compilation schemes should become obvious because the former do not allow us to derive such a conclusion.

References

Anderson, C. R., Smith, D. E., and Weld, D. S. (1998). Conditional effects in Graphplan. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS-98)*, pages 44–53. AAAI Press, Menlo Park.

Bäckström, C. (1995). Expressive equivalence of planning formalisms. *Artificial Intelligence*, 76(1–2):17–34.

Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1–2):165–204.

Cadoli, M. and Donini, F. M. (1997). A survey on knowledge compilation. *AI Communications*, 10(3,4):137–150.

Fikes, R. E. and Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208.

Furst, M., Saxe, J. B., and Sipser, M. (1984). Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability—A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA.

Gazen, B. C. and Knoblock, C. (1997). Combining the expressiveness of UCPOP with the efficiency of Graphplan. In Steel and Alami, 1997, pages 221–233.

Kambhampati, S., Parker, E., and Lambrecht, E. (1997). Understanding and extending Graphplan. In Steel and Alami, 1997, pages 260–272.

Karp, R. M. and Lipton, R. J. (1980). Some connections between nonuniform and uniform complexity classes. In *Conference Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing (STOC-80)*, pages 302–309, Los Angeles, California.

Kautz, H. A. and Selman, B. (1992). Forming concepts for fast inference. In *Proceedings of the 10th National Conference of the American Association for Artificial Intelligence (AAAI-92)*, pages 786–793, San Jose, CA. MIT Press.

Koehler, J., Nebel, B., Hoffmann, J., and Dimopoulos, Y. (1997). Extending planning graphs to an ADL subset. In Steel and Alami, 1997, pages 273–285.

McCarthy, J. and Hayes, P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B. and Michie, D., editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, Edinburgh, UK. Also published in Webber and Nilsson, 1981.

Nebel, B. (1998). On the compilability and expressive power of propositional planning formalisms. Technical Report 101, Albert-Ludwigs-Universität, Institut für Informatik, Freiburg, Germany.

Nebel, B. (1999a). Compilation schemes: A theoretical tool for assessing the expressive power of planning formalisms. In Burgard, W., Cremers, A. B., and Christaller, T., editors, *KI-99: Advances in Artificial Intelligence*, pages 183–194, Bonn, Germany. Springer-Verlag.

Nebel, B. (1999b). What is the expressive power of disjunctive preconditions? In Biundo, S. and Fox, M., editors, *Recent Advances in AI Planning. 5th European Conference on Planning (ECP'99)*, Durham, UK. Springer-Verlag. To appear.

Papadimitriou, C. H. (1994). *Computational Complexity*. Addison-Wesley, Reading, MA.

Pednault, E. P. (1989). ADL: Exploring the middle ground between STRIPS and the situation calculus. In Brachman, R., Levesque, H. J., and Reiter, R., editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 1st International Conference (KR-89)*, pages 324–331, Toronto, ON. Morgan Kaufmann.

Steel, S. and Alami, R., editors (1997). *Recent Advances in AI Planning. 4th European Conference on Planning (ECP'97)*, volume 1348 of *Lecture Notes in Artificial Intelligence*, Toulouse, France. Springer-Verlag.

Webber, B. L. and Nilsson, N. J., editors (1981). *Readings in Artificial Intelligence*. Tioga, Palo Alto, CA.