Proceedings of the Twenty-Sixth International Conference on
Automated Planning and Scheduling (ICAPS 2016)

Assessing the Expressivity of Planning Formalisms through the Comparison to Formal Languages

Daniel Höller and **Gregor Behnke** and **Pascal Bercher** and **Susanne Biundo**
Institute of Artificial Intelligence, Ulm University, D-89069 Ulm, Germany
{daniel.hoeller, gregor.behnke, pascal.bercher, susanne.biundo}@uni-ulm.de

Abstract

From a theoretical perspective, judging the expressivity of planning formalisms helps to understand the relationship of different representations and to infer theoretical properties. From a practical point of view, it is important to be able to choose the best formalism for a problem at hand, or to ponder the consequences of introducing new representation features. Most work on the expressivity is based either on compilation approaches, or on the computational complexity of the plan existence problem. Recently, we introduced a new notion of expressivity. It is based on comparing the structural complexity of the set of solutions to a planning problem by interpreting the set as a formal language and classifying it with respect to the Chomsky hierarchy. This is a more direct measure than the plan existence problem and enables also the comparison of formalisms that can not be compiled into each other. While existing work on that last approach focused on different hierarchical problem classes, this paper investigates STRIPS with and without conditional effects; though we also tighten some existing results on hierarchical formalisms. Our second contribution is a discussion on the language-based expressivity measure with respect to the other approaches.

1 Introduction

There has been some work on comparing the expressivity of different planning formalisms. From a theoretical point of view, this is interesting in order to understand the relationship of alternative representations or to infer theoretical properties. From a more practical point of view, it is useful when pondering the consequences of introducing new representation features and it may help when adapting existing heuristics to a new class of problems. When having a new application domain at hand, it may also be insightful to judge which formalism is rich enough to realize a meaningful model without making reasoning too hard.

There are several ways to compare different planning formalisms. The comparison of the time complexity of the *plan existence problem* is one of them, i.e. the problem of deciding whether a planning instance has a solution or not. This enables an indirect measure of the expressivity of the representation language. For propositional STRIPS planning, it ranges from **P**, for severely restricted problems, to **PSPACE-complete** for the full formalism (Bylander 1994). The complexity of this problem in Hierarchical Task Network (HTN) planning ranges up to **undecidable** (Erol, Hendler, and Nau 1994; 1996; Geier and Bercher 2011; Alford, Bercher, and Aha 2015a). When allowing the insertion of tasks apart from decomposition (this problem setting is called HTN planning with task insertion, TI-HTN planning), the plan existence problem becomes decidable (Geier and Bercher 2011) and is up to **NEXPTIME-complete** for propositional representations and **2-NEXPTIME-complete** when having variables (Alford, Bercher, and Aha 2015b).

A second approach to compare different formalisms is based on the compilation of planning problems under restrictions on (1) the space used for the generated representation, (2) the length of solutions, as well as (3) the runtime of the transformation (Erol, Hendler, and Nau 1994; 1996; Bäckström 1995; Nebel 2000). Bäckström (1995) uses polynomial time compilations that do not change the length of the resulting plans. Nebel (2000) makes no restrictions on computation time and distinguishes different growths in plan size up to polynomial space. Compilation is done without regarding the initial state and goal description to guarantee non-trivial solutions (Nebel 2000).

In (Höller et al. 2014) we introduced a new notion of expressivity. We interpreted the set of solutions to a planning problem as a formal language and investigated which classes of formal languages can be expressed by HTN planning formalisms under different restrictions. This enables a comparison of how complex the structure of generated plans can be. Compared to the runtime-based method, this is a more direct measure of expressivity, though there are some interesting connections between the approaches, because the plan existence problem in planning equals the emptiness problem of the problem's language (Behnke, Höller, and Biundo 2015). Further analogies between planning- and language-related problems are given in the discussion. Compared to compilation-based approaches, it enables also the comparison of formalisms that can not be compiled into each other.

This paper extends the work on the language-based expressivity measure by results concerning the STRIPS formalism with and without conditional effects. We further give some HTN-related results to make all subset relations strict. The remaining results complement the language hierarchy and show that all its intersections are non-empty. The discussion elucidates the properties of the approach and gives a more detailed comparison to related work.

2 Formal Framework

This section introduces the formal framework used in the remaining paper. It first introduces the STRIPS and HTN planning formalisms, defines the language of a planning problem and explains the notation used throughout the paper.

The used formalism is based on the HTN formalism by Geier and Bercher (2011) that we also used in (Höller et al. 2014). However, we aim to provide a unified formalism both for STRIPS, STRIPS with conditional effects (denoted STRIPS-CE), and HTN planning, i.e. the definition of our (slightly adapted) HTN formalism is an extension of the STRIPS formalism. The definition of conditional effects is based on the formalization by Röger, Pommerening, and Helmert (2014), though unifying the formalisms required some changes. A major change to the original formalism is the introduction of a goal description in HTN planning. This extension enables a more direct interpretation of TI-HTN planning to be an extension of STRIPS planning. However, it is no problem to compile the goal description away by introducing a new primitive task that occurs last in any solution and that uses that goal description as precondition. Section 4.1 is a discussion on the impact of this change on the languages and on the results we build on.

2.1 STRIPS Planning

A STRIPS planning problem is a tuple $P = (L, A, s_0, g, \delta)$, where L is a set of propositional environment facts, $s_0 \subseteq L$ is the initial state and $g \subseteq L$ the goal description. A is a set of *action names* that forms the set of terminal symbols of the resulting language as well. We will often just refer to its elements as *actions*, though these are just the names that do not have the often-used tuple form. Their preconditions and positive as well as negative effects are given by the functions $prec$, add and del that are included in δ, i.e. $\delta = (prec, add, del)$. $prec$ is defined as $prec : A \to 2^L$. In case of ordinary STRIPS planning (not including conditional effects), the functions add and del are defined in the same way. In STRIPS-CE planning, the latter two functions map action names to *conditional effects*, i.e. to (a set of) pairs that each include a set of preconditions as well as a set of effects as elements $add, del : A \to 2^{2^L \times 2^L}$. Let ce be a conditional effect. We will refer to its preconditions as $p\text{-}ce(ce)$ and to its effects as $e\!f\!f(ce)$.

The function $\tau : A \times 2^L \to \{true, false\}$ returns whether an action is applicable to a state, i.e. $\tau(a, s) \Leftrightarrow prec(a) \subseteq s$. Whenever an action is applicable, the state transition function $\gamma : A \times 2^L \to 2^L$ returns the state resulting from applying an action to a state:

$$\gamma(a, s) = [s \setminus \bigcup_{ce \in del(a) \text{ with } p\text{-}ce(ce) \subseteq s} e\!f\!f(ce)] \cup \bigcup_{ce' \in add(a) \text{ with } p\text{-}ce(ce') \subseteq s} e\!f\!f(ce')$$

As STRIPS is a (syntactical) special case of STRIPS-CE, we omit its definition of γ.

A sequence $(a_0 a_1 \dots a_n)$ of actions is applicable to a state s_0 when each action a_i with $0 \leq i \leq n$ is applicable to the state s_i. For $1 \leq i \leq n+1$, the state s_i is defined as $\gamma(a_{i-1}, s_{i-1})$. The state s_{n+1} *results* from applying the sequence, denoted by $s_{n+1} = \gamma^*((a_0 a_1 \dots a_n), s)$. A *goal state* is a state s with $g \subseteq s$. An action sequence is a *plan* (also *solution*) to a problem if and only if it is applicable to s_0 and results in a goal state when applied to s_0.

2.2 HTN Planning

In HTN planning, the objective is to fulfill an overall task rather than to find a sequence of actions that transforms a given state to a goal state. Therefore, so-called *abstract* (or *compound*) tasks are decomposed iteratively into subtasks until the level of abstraction is detailed enough to execute the resulting tasks. These tasks are called to be *primitive*. They can not be decomposed anymore and are equal to the definition of actions given above. Like the action names A, the set N_p defines the primitive task names. Further, $\delta = (prec, add, del)$, where $prec$, add, and del are functions $f : N_p \to 2^L$. Applicability and state transitions are defined as given for STRIPS. Let N_a be the set of abstract task names. Note that we use the terms *abstract task name* and *abstract task* synonymously. The set of all task names is abbreviated by $N = N_a \cup N_p$.

Tasks are organized in so-called *task networks*. A task network tn is a tuple $(T, \prec, \alpha)$, where T is a (possibly empty) set of unique task identifier symbols, which are mapped to their actual task names by the function $\alpha : T \to N$. $\prec \subseteq T \times T$ defines a partial order on the task identifiers. This enables a single task name to appear more than once in a task network. Two task networks $tn = (T, \prec, \alpha)$ and $tn' = (T', \prec', \alpha')$ are called *isomorphic* (written $tn \cong tn'$) if they differ solely in their identifiers, i.e. there is a bijection $\sigma : T \to T'$ so that for all identifiers $t, t' \in T$ holds that $[(t, t') \in \prec] \Leftrightarrow [(\sigma(t), \sigma(t')) \in \prec']$ and $\alpha(t) = \alpha'(\sigma(t))$.

Abstract tasks are decomposed by the application of *(decomposition) methods*. A method m is a pair (c, tn) that maps the abstract task $c \in N_a$ to the task network tn, which specifies the *subtasks* of c and their order. Each subtask can be primitive or again abstract. More precisely, a method (c, tn) decomposes a task network $tn_1 = (T_1, \prec_1, \alpha_1)$ into a task network $tn_2 = (T_2, \prec_2, \alpha_2)$ if $t \in T_1$ with $\alpha_1(t) = c$ and if there is a task network $tn' = (T', \prec', \alpha')$ with $tn' \cong tn$ and $T_1 \cap T' = \emptyset$. The resulting task network tn_2 is defined as

$$\begin{aligned} tn_2 &= ((T_1 \setminus \{t\}) \cup T', \prec' \cup \prec_D, (\alpha_1 \setminus \{t \mapsto c\}) \cup \alpha') \\ \prec_D &= \{(t_1, t_2) \mid (t_1, t) \in \prec_1, t_2 \in T'\} \cup \\ &\quad \{(t_1, t_2) \mid (t, t_2) \in \prec_1, t_1 \in T'\} \cup \\ &\quad \{(t_1, t_2) \mid (t_1, t_2) \in \prec_1, t_1 \neq t \wedge t_2 \neq t\} \end{aligned}$$

We will write $tn \to^*_{TD} tn'$ to denote that the task network tn can be decomposed into the task network tn' by applying one or more decompositions.

In common HTN planning, decomposition is the only way to refine a task network. However, there is another variant of HTN planning that is called HTN *planning with task insertion* (TI-HTN planning) (Geier and Bercher 2011;

Alford, Bercher, and Aha 2015b). It allows for the insertion of primitive tasks into a task network. This possibility allows for specifying only the basic structure of a solution in the hierarchy and find the remaining tasks by task insertion, but it also decreases the expressivity of the model (Höller et al. 2014; Alford, Bercher, and Aha 2015b). The task network $tn' = (T', \prec', \alpha')$ resulting from inserting a task $p \in N_p$ into a task network $tn = (T, \prec, \alpha)$ is defined as $T' = T \cup \{t\}$ with $t \notin T$, $\prec' = \prec$ and $\alpha' = \alpha \cup \{(t \mapsto p)\}$. We will write $tn \rightarrow^*_{TI} tn'$ to denote that tn' can be achieved by inserting an arbitrary number of tasks into tn.

An HTN or TI-HTN planning problem is a tuple $P = (L, N_a, N_p, M, s_0, c_I, g, \delta)$ where L is a set of propositional environment facts, N_a and N_p are the sets of abstract and primitive task names, M is the set of methods, $s_0 \in 2^L$ is the initial state, $g \in 2^L$ is the goal description and c_I the initial task. We will denote the task network containing solely the initial task c_I as tn_I.

A task network $tn_S = (T_S, \prec_S, \alpha_S)$ is a solution to a planning problem P if and only if

- All tasks are primitive and there is an applicable sequence $(t_1 t_2 \dots t_n)$ of the task identifiers that is in line with $\prec_S$ and $s_n = \gamma^*(\alpha(t_1)\alpha(t_2)\dots\alpha(t_n), s_0)$ is a goal state.
- $tn_I \rightarrow^*_{TD} tn_S$ holds for a solution in HTN planning.
- $tn_I \rightarrow^*_{TD} tn'$ and $tn' \rightarrow^*_{TI} tn_S$ holds for a solution in TI-HTN planning.

Let $Sol_{HTN}(P)$ and $Sol_{TI}(P)$ be the set of all HTN and TI-HTN solutions to the problem P, respectively.

2.3 The Language of a Planning Problem

In formal languages, a set of rules is given that describe the structure of valid words and the language is the set of these words. In planning there is a set of rules given that describe the structure of a valid plan, so the set of all plans forms the language of the problem. For STRIPS the language of a problem $P = (L, A, s_0, g, \delta)$ is defined as

$$\mathcal{L}(P) = \{\omega = a_1 a_2 \dots a_n \mid a_i \in A, n \geq 0, \gamma^*(\omega, s_0) \supseteq g\}$$

For HTN and TI-HTN planning the definition is more complicated due to the partial order of the solutions. To be able to compare their language with formal languages and other planning formalisms, its language is defined as the set of all applicable linearizations of the solution task networks:

$$\mathcal{L}(P) = \{\omega = \alpha(t_1)\alpha(t_2)\dots\alpha(t_n) \mid \gamma^*(\omega, s_0) \supseteq g, \omega \text{ is a linearization of a } tn = (T, \prec, \alpha) \text{ with } tn \in Sol_{HTN|TI}(P)\}$$

We will clarify the meaning of abbreviations by using different font styles. When mentioning a planning problem in text, we will use small capitals and write e.g. STRIPS planning. For languages we will use a calligraphical font style and write e.g. $\mathcal{STRIPS}$ to denote the set of all languages generated by STRIPS planning problems.

3 Languages

This section shows the relations between TI-HTN, STRIPS, and acyclic HTN planning problems (denoted HTN-AC). Afterwards it gives results related to STRIPS-CE.

The following proposition simplifies some of the proofs in this section. It states that applying an action more than once subsequently does not change the state. It assumes that the action is applicable (1) before its first and (2) after its first application. Given that the state does not change anymore afterwards, it stays applicable.

Proposition 1. *Let a be an arbitrary (fixed) action without conditional effects and s_0 an arbitrary (fixed) state. Given that the action is applicable in s_0 and in $\gamma(a, s_0)$, it holds $\gamma^*((a), s_0) = \gamma^*((aa), s_0) = \dots = \gamma^*((aa\dots a), s_0)$.*

Proof Sketch. W.l.o.g. lets call the states after the first and second application s_1 and s_2, respectively. Then it holds:

$$s_1 = (s_0 \setminus del(a)) \cup add(a)$$
$$s_2 = (((s_0 \setminus del(a)) \cup add(a)) \setminus del(a)) \cup add(a)$$

When a is applied the first time, the resulting state s_1 does not contain state features in $del(a)$ apart from those also included in $add(a)$. When applying a a second time, all state features that have to be deleted are therefore exactly those also included in $add(a)$ and are added right after they have been deleted. The second deletion does not change the state. After the first application of a, the resulting state s_1 does, by definition, contain all state feature included in $add(a)$. When it is applied the second time, $add(a)$ inserts precisely the state features in $del(a) \cap add(a)$ and does therefore not change the state. It holds $s_1 = s_2$. This argument can be made recursively. □

Theorem 1 ($\mathcal{STRIPS} \subsetneq \mathcal{TI\text{-}HTN}$)**.** *The set of languages generated by* STRIPS *planning problems is a strict subset of the languages generated by* TI-HTN *planning problems.*

Proof. We show that for every STRIPS planning problem, there is an TI-HTN planning problem with the same set of solutions. Let $P = (L, A, s_0, g, \delta)$ be a STRIPS planning problem. We now define a TI-HTN planning problem $P' = (L, \{c_I\}, A, \{(c_I, (\emptyset, \emptyset, \emptyset))\}, s_0, c_I, g, \delta)$. P' contains a single initial task that can only be decomposed into an empty task network, so every primitive task in a solution is introduced by task insertion. The two problems share the initial state, the goal description, the actions' preconditions and effects as well as the state transition function. Obviously it holds that $\mathcal{L}(P) = \mathcal{L}(P')$.

Now we show that there is (at least) one planning problem that can be expressed by TI-HTN planning, but not by STRIPS planning. Consider the following TI-HTN planning problem:

$$P_{a^{2+n}} = (\emptyset, \{c_I\}, \{a\}, M, s_0 = \emptyset, c_I, g = \emptyset, \delta) \text{ with}$$
$$M = \{(c_I, (\{t_1, t_2\}, \emptyset, \{(t_1 \mapsto a), (t_2 \mapsto a)\}))\} \text{ and}$$
$$\delta = (\{a \mapsto \emptyset\}, \{a \mapsto \emptyset\}, \{a \mapsto \emptyset\})$$

The initial task network ensures that at least two occurrences of a are included in every plan. Afterwards, the planning system is allowed to include an arbitrary number of additional as. The language of $P_{a^{2+n}}$ is obviously $\mathcal{L}(P_{a^{2+n}}) = L_{a^{2+n}} = \{a^n \mid n \geq 2\}$.

This language can not be expressed in STRIPS. The action a has to be applicable in the initial state and stays applicable after an arbitrary number of applications. Due to Prop. 1, the state of a STRIPS planning problem converges after the first application of a. Thus, in STRIPS without conditional effects, the state after the first application of a necessarily fulfills the goal description, the word a is also included in the language and it is not possible to enforce the second application. □

So far we have seen that $\mathcal{STRIPS}$ is a strict subset of $\mathcal{TI\text{-}HTN}$. In previous work we showed that $\mathcal{TI\text{-}HTN}$ is included in the regular languages (Höller et al. 2014, Thm. 5). Intuitively, this is because planning systems may stop any recursion at first possibility and add missing parts to the plan by task insertion (Alford, Bercher, and Aha 2015b). The set of tasks inserted by decomposition is in this case finite and thus regular. As $\mathcal{STRIPS}$ is in $\mathcal{REG}$ too, their intersection is also regular.

However, in (Höller et al. 2014) we did not show if this subset relation is strict. The next theorem states that it is.

Theorem 2 ($\mathcal{TI\text{-}HTN} \subsetneq \mathcal{REG}$)**.** *The set of languages generated by* TI-HTN *planning problems is a strict subset of the regular languages.*

Proof. We show that there is a regular language that can not be generated by any TI-HTN planning problem. Consider the following regular grammar (consisting of a terminal and a non-terminal alphabet, the set of production rules and the start symbol):

$$G_{a^{3n}} = (\{a\}, \{S_0, S_1, S_2\}, \{S_0 \to aS_1, S_1 \to aS_2, S_2 \to aS_0, S_2 \to a\}, S_0)$$

It generates the (regular) language including all words that contain a number of as that can be divided by three. $\mathcal{L}(G_{a^{3n}}) = \{a^{3\times n} \mid n \in \mathbb{N}\}$. Due to Prop. 1, it is not possible to make any state change (in STRIPS) by applying an action more than once. Thus it is not possible to generate the language given above by preconditions and effects. Since TI-HTN planning systems are allowed to insert actions apart from the hierarchy, there is no way to exclude words having a different number of as. □

The last theorems introduce a strict inclusion hierarchy of the language classes $\mathcal{STRIPS}$ (the least expressive class considered here), $\mathcal{TI\text{-}HTN}$, and $\mathcal{REG}$. Now we come to another restriction on HTN planning problems that make their language regular and classify its languages with respect to the classes given so far. We denote the set of HTN planning problems without recursion as HTN-AC and will call it *acyclic* or *non-recursive*. Intuitively, this means that when a task is decomposed the same task can not be reached anymore. Formally, it means that the size of possible decomposition trees is limited by some constant (Höller et al. 2014, Def. 8). We showed that $\mathcal{HTN\text{-}AC}$, the set of languages generated by such problems, is strictly included in the regular languages (Höller et al. 2014, Thm. 3). More precisely, the set that is generated is exactly the set of all finite languages. Since there are finite languages that are expressible in STRIPS, the next corollary holds:

Corollary 1 ($\mathcal{STRIPS} \cap \mathcal{HTN\text{-}AC} \neq \emptyset$)**.** *The intersection of $\mathcal{STRIPS}$ and $\mathcal{HTN\text{-}AC}$ is non-empty.*

Because there are languages included in $\mathcal{STRIPS}$ that are infinite, the following corollary holds:

Corollary 2 ($\mathcal{STRIPS} \setminus \mathcal{HTN\text{-}AC} \neq \emptyset$)**.** *There are languages included in $\mathcal{STRIPS}$ that are not included in $\mathcal{HTN\text{-}AC}$.*

Consider the language including the single word aaa. Due to Prop. 1, it can not be expressed in STRIPS. Using the initial task network of TI-HTN planning, one can introduce a minimum number of as in each word. However, there is no way to limit the number of as to three occurrences. Thus the following corollary holds:

Corollary 3 ($\mathcal{HTN\text{-}AC} \setminus \mathcal{TI\text{-}HTN} \neq \emptyset$)**.** *There are languages included in $\mathcal{HTN\text{-}AC}$ that are not included in $\mathcal{TI\text{-}HTN}$.*

Intuitively, the initial task network of TI-HTN planning problems allows to force specific tasks to be in every plan. We have seen that this can be used to express languages that can not be expressed by STRIPS. This raises the question whether there is one such language that is finite and can thus also be expressed in HTN-AC. The next theorem states that there are such languages.

Theorem 3 ($((\mathcal{TI\text{-}HTN} \setminus \mathcal{STRIPS}) \cap \mathcal{HTN\text{-}AC}) \neq \emptyset$)**.** *There are languages that can be expressed by both* TI-HTN *and* HTN-AC *planning problems, but not by* STRIPS*.*

The following proof shows that the language $L_{\neg abc} = \{acb, bac, bca, cab, cba\}$ of all permutations of the symbols a, b and c except abc is such a language.

Proof. *$L_{\neg abc} \in \mathcal{HTN\text{-}AC}$ and $L_{\neg abc} \in \mathcal{TI\text{-}HTN}$.* We give an HTN planning problem that is acyclic and prevents task insertion by its preconditions and effects. Thus it forms an HTN-AC as well as an TI-HTN planning problem that generates the language $L_{\neg abc}$.

Let $P_{\neg abc} = (L, N_a, N_p, M, c_I, s_0, g, \delta)$ with $L = \{pa, pb, pc\}$, $N_a = \{c_I\}$, $N_p = \{a, b, c\}$, $g = \emptyset$, $\delta = (prec, add, del)$ with $prec = \{a \mapsto pa, b \mapsto pb, c \mapsto pc\}$, $add = \{a \mapsto \emptyset, b \mapsto \emptyset, c \mapsto \emptyset\}$, $del = \{a \mapsto pa, b \mapsto pb, c \mapsto pc\}$ and $s_0 = \{pa, pb, pc\}$.

$$\begin{aligned} M = \{&(c_I, (\{t_1, t_2, t_3\}, \{(t_1, t_2), (t_2, t_3), (t_1, t_3)\}, \alpha)) \\ &\text{with } \alpha \in \{\{t_1 \mapsto a, t_2 \mapsto c, t_3 \mapsto b\}, \\ &\qquad \{t_1 \mapsto b, t_2 \mapsto a, t_3 \mapsto c\}, \\ &\qquad \{t_1 \mapsto b, t_2 \mapsto c, t_3 \mapsto a\}, \\ &\qquad \{t_1 \mapsto c, t_2 \mapsto a, t_3 \mapsto b\}, \\ &\qquad \{t_1 \mapsto c, t_2 \mapsto b, t_3 \mapsto a\}\}\} \end{aligned}$$

When the ordinary HTN solution criterion (Criterion 2.a) is applied to this acyclic problem, the initial task is decomposed into one of the desired words, so it obviously holds that $\mathcal{L}(P_{\neg abc}) = L_{\neg abc}$. When the TI-HTN solution criterion is applied, the preconditions and effects of the primitive tasks prevent any action to be more than once in a solution. Since the decomposition of the initial task inserts every action once in a plan, there will be no task insertion at all and we get the same language as with the HTN solution criterion.

$L_{\neg abc} \notin \mathcal{STRIPS}$. Suppose there is a STRIPS planning problem $P = (L, A, s_0, g, \delta)$ with $\delta = (prec, add, del)$ with $\mathcal{L}(P) = L_{\neg abc}$. W.l.o.g. $A = \{a, b, c\}$.

1. acb is in the language, i.e. a is applicable in s_0 and $\tau(a, s_0)$ holds.
2. bac is in the language, $s_0 \supseteq prec(b)$, b is applicable in the initial state.
3. cab is in the language, i.e.

$$\tau(b, \gamma(a, \gamma(c, s_0))) \Leftrightarrow ((((s_0 \setminus del(c)) \cup add(c)) \\ \setminus del(a)) \cup add(a)) \supseteq prec(b)$$

   Applying a to a state does not delete any state feature included in $prec(b)$, i.e. $del(a) \cap prec(b) = \emptyset$ (assuming, w.l.o.g., that $del(a) \cap add(a) = \emptyset$).
4. cab is in the language, $s_0 \supseteq prec(c)$.
5. Given that acb and bca are in the language, neither a nor b does delete any state feature in $prec(c)$.

Given (1) it holds that $\tau(a, s_0)$, including (2) and (3), it holds $\tau(b, \gamma(a, s_0))$. Given (4) and (5) it holds that $\tau(c, \gamma(b, \gamma(a, s_0)))$. The action sequence abc is applicable in s_0.

It now has to be shown that the resulting state fulfills the goal description. Given that the application of any order except abc result in a goal state, it holds that $s_0 \cup add(a) \cup add(b) \cup add(c) \supseteq g$. Since cba, acb and bac are in the language, neither of the actions deletes any environment property that is in the goal description. Thus abc is applicable to s_0 and results in a goal state. Our assumption that there is a STRIPS planning problem generating $L_{\neg abc}$ must be wrong. □

Corollary 4 ($\mathcal{TI}$-$\mathcal{HTN} \setminus \mathcal{STRIPS} \setminus \mathcal{HTN}$-$\mathcal{AC} \neq \emptyset$). *The language $L_{a^{2+n}}$ given in Thm. 1 is non-finite. So we have already introduced a language that is included in $\mathcal{TI}$-$\mathcal{HTN}$, not included in $\mathcal{STRIPS}$ (due to Thm. 1) and not included in $\mathcal{HTN}$-$\mathcal{AC}$.*

This corollary concludes the classification of HTN planning problems without recursion. It has been shown that it has non-empty intersections with every language class introduced so far.

Now we investigate another well-known planning formalism, STRIPS planning with conditional effects, denoted STRIPS-CE. The next theorem states that the corresponding set of languages $\mathcal{STRIPS}$-$\mathcal{CE}$ equals the class of regular languages.

Theorem 4 ($\mathcal{STRIPS}$-$\mathcal{CE} = \mathcal{REG}$). *The set of languages generated by* STRIPS-CE *planning problems equals the set of regular languages.*

Proof. $\mathcal{STRIPS}$-$\mathcal{CE} \subseteq \mathcal{REG}$. To proof that the class $\mathcal{STRIPS}$-$\mathcal{CE}$ is included in $\mathcal{REG}$, we show that for every STRIPS-CE planning problem $P = (L, A, s_0, g, \delta)$ with $\delta = (prec, add, del)$, there is a deterministic finite automaton (DFA) that accepts the language $\mathcal{L}(P)$. Let $dfa = (\Sigma, S, d, i, F)$ be a DFA where Σ is its input alphabet, S its set of states, i its initial state, $d : S \times \Sigma \to S$ its state-transition function and F its set of final states. We define $\Sigma = A$. The states of the DFA equal the set of possible states in the planning problem, $S = 2^L$, i.e. each state of the DFA is given by the subset of L that holds in the corresponding state of the planning problem. So i is the state of the DFA that contains exactly the literals that hold in s_0 and every state including the literals in g is included in F. The state transition function of the DFA is defined according to the state transition function of the planning problem:

$$d(s, a) = \begin{cases} s', & \textit{iff } (\tau(a, s) \wedge \gamma(a, s) = s') \\ \textit{undefined}, & \textit{else} \end{cases}$$

The DFA has the same state space as the original problem. By definition, those states that fulfill the goal description of P are included in the final states. The state transition function simulates the state transitions of the planning problem. For symbols that belong to actions that are not applicable in the current state, d is undefined. Starting in the initial state representing s_0, the DFA accepts exactly the sequences of actions transforming that state to a goal state of the planning problem. Thus it accepts exactly the words in $\mathcal{L}(P)$.

$\mathcal{STRIPS}$-$\mathcal{CE} \supseteq \mathcal{REG}$. Here we show that for every regular language there is a STRIPS-CE planning problem that generates it. For every regular language there is a DFA without ε-transitions that accepts it. Let $dfa = (\Sigma, S, d, i, F)$ be such a DFA. We assume that d is total (if it is not, it can be made total by introducing a new state that forms a dead end for the transformation). We define a STRIPS-CE planning problem $P = (L, A, s_0, \{g\}, \delta)$ where $L = S \cup \{g\}$ and $g \notin S$. The initial state is $s_0 = \{i\}$. Further, g is included in s_0 if and only if $i \in F$. The set of actions A equals the alphabet Σ and the functions in δ are defined as follows:

$$\forall a \in A : prec(a) = \emptyset$$
$$add(a) = \{(\{s\}, \{s'\} \cup G') \mid d(s, a) = s'\}$$
$$\text{with } G' = \begin{cases} \{g\}, & \text{if } s' \in F \\ \emptyset, & \text{else} \end{cases}$$
$$del(a) = \{(\emptyset, L)\}$$

There is one action for every terminal symbol of the DFA. Since d is total, every action has a conditional effect for every state of the DFA, simulating the d function. The delete effects delete all state features. Disregarding the goal description g, the number of state features that hold is not changed by the application of an action. Starting planning in the state where only the initial state of the DFA holds, there holds (apart from g) exactly one state feature in every reachable state; and action application simulates the state transition of the DFA. The language accepted by dfa equals the language of the planning problem $\mathcal{L}(P)$.

This completes the overall proof that $\mathcal{STRIPS}\text{-}\mathcal{CE} = \mathcal{REG}$. $\square$

4 Discussion and Related Work

We extended the HTN and TI-HTN formalism by adding a goal description to obtain a unifying formalization for hierarchical and non-hierarchical planning. Sec. 4.1 discusses the impact of this alteration, in particular why previous results still hold despite the changes. Sec. 4.2 discusses the results of this paper with respect to related work on the formal-language-based expressivity measure. We show the complete dependencies between the Chomsky hierarchy and the studied classes of planning problems. Sec. 4.3 elaborates the connections between the expressivity measure regarding formal languages and other measures known from the literature, such as the complexity of the plan existence problem or compilation techniques.

4.1 Consequences of Adding a Goal State

We first want to discuss the consequences of adding a goal description to the formalism. As we have seen in Thm. 1, it enables an elegant overall formalism where TI-HTN complements the STRIPS formalism by adding an hierarchical part of constraints that increase expressivity.

Since the case where there is no goal description is a special case, the problem can not become simpler, but may become harder. However, the goal description can be compiled away by adding a primitive task to the initial task network that is placed at the end of each plan and has the goal properties as precondition (Geier and Bercher 2011). For the HTN formalization by Erol, Hendler, and Nau (1996), this was first formally shown in the hardness proof of Thm. 5 stating that HTN planning is at least as expressive as STRIPS planning. So, from a computational complexity point of view, the impact of having a goal description is quite limited.

From the formal language point of view, there is an impact: When there is one specific (technical) primitive task at the end of every single solution, the language becomes prefix-free, i.e. there can not be a solution that is the prefix of a second solution. Of course, this is a quite marginal difference that could easily be eliminated by a post processing. However, this is not allowed in the language-based approach because it has a significant impact on the given classes. Consider e.g. that a renaming of actions would be allowed, then a STRIPS-CE action could be simulated by a set of STRIPS actions that are renamed to their original names afterwards. So, when allowing a post-processing, several classes of languages would fall together and it is prohibited.

Knowing that the given modification of the formalism may change the planning problem's language raises the question whether our results in (Höller et al. 2014) still hold in the formalism given here. So we have a closer look at the given proofs and investigate whether they still hold.

One property used in several proofs is that the language generated by an HTN planning problem can be regarded as the intersection of two languages: (1) the language generated by the hierarchical part of the problem and (2) the language generated by the non-hierarchical part (Höller et al. 2014, Sec. 3.1). Since the language generated by the non-hierarchical part of the problem is regular, several proofs rely on the fact that a language class at hand is closed under the intersection with regular languages. So we have to show that the non-hierarchical part of the language is still regular. As STRIPS is a special case of STRIPS-CE, this has already been done in the first part of the proof for Thm. 4. Thus, Thms. 5, 6, 7 in (Höller et al. 2014) still hold.

The proof concerning HTN-AC is based on the fact that the language is finite (this still holds). The results on planning formalisms not showing preconditions and effects are trivially unaffected. These are Cors. 1, 2, 3 and Thms. 8, 9 in (Höller et al. 2014). Thus the changed formalism does not affect the results.

4.2 An Overview of Language-related Expressivity Results

Now we discuss the relation of this paper to former language-related results and give the resulting language hierarchy of planning formalisms. First we summarize (Höller et al. 2014) and add the results of this paper afterwards.

In (Höller et al. 2014, Thm. 8) we have shown that the HTN planning formalism is able to express a strict (and non-context-free) subset of the context-sensitive languages. This is due to the partial ordering between abstract tasks that can not be compiled away without committing to an ordering relation between the subtasks they are decomposed into.

This means that the result holds also for problems that lack preconditions and effects (called HTN-NOOP). The fact that the subset relation between (full) HTN and HTN-NOOP is strict follows trivially from the complexity of the corresponding plan existence problems (Erol, Hendler, and Nau 1994; Alford et al. 2014) (cf. (Höller et al. 2014, Thm. 9)).

The set of languages generated by totally ordered HTN planning problems (i.e. the initial task network as well as all subtask networks of methods are totally ordered), denoted HTN-ORD, equals the set of context-free languages, regardless of whether the problems include preconditions and effects or not (Höller et al. 2014, Thm. 6).

The set of languages generated by TI-HTN planning is included in the set of regular languages (Höller et al. 2014, Thm. 5). Due to Thm. 2 given in this paper, this subset-relation is strict.

Using the modified formalism introduced in this paper, TI-HTN is a straightforward extension of STRIPS and thus its language is unsurprisingly a superset of the language generated by STRIPS (see Thm. 1). We have further seen that this relation is strict.

Interestingly, the introduction of conditional effects results in a class of languages that equals the regular languages (Thm. 4). Thus there are two planning formalisms (HTN-ORD and STRIPS-CE) that result in language classes already known from the Chomsky hierarchy.

The remaining results complement the hierarchy and show that all intersections of the regular language classes are non-empty (see Cors. 1, 2, 3, 4 and Thm. 3). Fig. 1 gives the overall language hierarchy including the results in (Höller et al. 2014) and the results given in this paper.

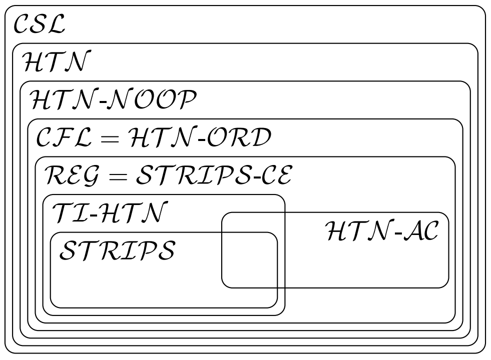
$\mathcal{CSL}$
$\mathcal{HTN}$
$\mathcal{HTN}$-$\mathcal{NOOP}$
$\mathcal{CFL} = \mathcal{HTN}$-$\mathcal{ORD}$
$\mathcal{REG} = \mathcal{STRIPS}$-$\mathcal{CE}$
$\mathcal{TI}$-$\mathcal{HTN}$
$\mathcal{STRIPS}$
$\mathcal{HTN}$-$\mathcal{AC}$

Figure 1: Overview of the expressivity of different planning formalisms. All given subset relations are strict and all given intersections are non-empty. Two language classes generated by planning formalisms equal classes from the Chomsky hierarchy – STRIPS-CE equals the regular languages and HTN-ORD the context-free languages.

4.3 Formal Languages as Measure for the Expressivity of Planning Formalisms

The language-based approach to assess expressivity has originally been motivated by parsing-related approaches to plan and goal recognition (Höller et al. 2014). There has been quite a lot of work on these approaches to that problem (see (Carberry 2001; Krüger et al. 2007; Ingrand and Ghallab 2013; Sukthankar et al. 2014) for overviews). As the models used in plan and goal recognition describe agents' behavior, planning models are a nearby choice for its description. The given expressivity measure is interesting to understand which class of languages has to be parsed when choosing a planning formalism.

When we have a closer look at the problem of plan and goal recognition in the context of the given language of a planning problem, it is the problem to decide whether there is a word in the language that has the given prefix of observations. But there are other interesting connections between the language and problems studied in automated planning (Behnke, Höller, and Biundo 2015): The problem of plan verification (see e.g. (Nebel and Bäckström 1994; Lang and Zanuttini 2012; Behnke, Höller, and Biundo 2015)) is similar to the problem of deciding whether a given sequence of actions is included in the language, i.e. its word problem. The maybe most studied problem in automated planning is the plan existence problem, which resembles the emptiness problem of the language.

Though one has to carefully regard the representation's blow-up when studying the relations between the different planning- and formal-language-related problems, there are interesting interconnections, as e.g. the (non-)decidability of the emptiness problem of different planning formalisms, or the complexity of the verification for classical as well as totally/partially ordered HTN planning.

The property that the blow-up of the representation is not respected in the approach has both advantages and disadvantages. On one hand it is not as practical as the compilation- or runtime-based view. Compilation (under some restriction in compilation runtime and representation blowup) can be applied quite directly in planning systems. The theoretical runtime of the plan existence (i.e. emptiness) problem is, in practical planning, of less importance than compilation, but of course interesting when relaxing problems for the development of heuristics.

On the other hand, there are also some counterintuitive results (that are totally fine when looking at the original purpose of the approaches) when using runtime or compilation as a measure for expressivity: When using two representations of the same formalisms (e.g. a lifted and a grounded representation), its "expressivity" may increase or decrease. The language-based approach does not have these problems, since it does not represent the blow-up. So it may be the most theoretical approach. From a practical point of view, the most interesting application seems to be plan and goal recognition. But it may also help to choose the best-fitted formalism for a problem at hand, since it enables a look at how complex the structures in generated plans may be (and that seems to be the thing to look at).

Compilations give a relative measure, the absolute value of expressivity is lost, especially when compilation is not possible anymore, e.g. from HTN to STRIPS planning. In the lower regions of expressivity, the runtime of the compilation as well as the change of representation size gives the relative measure between formalisms. In the language-based approach, the Chomsky hierarchy gives a reference framework that enables a more absolute measure.

Compared to the other approaches, runtime-based expressivity is a more indirect measure. It does not state how much can be expressed, but how hard some (interesting) problem is when a specific representation is chosen.

5 Conclusion

In this paper we investigate the expressivity of the STRIPS planning formalism with respect to the measure of expressivity based on the classification with respect to formal languages. It turns out that STRIPS without conditional effects is a strict subset of the languages generated by TI-HTN planning, i.e. it is included in the regular languages. Interestingly, when allowing for conditional effects, the formalism generates exactly the class of regular languages. We further give some results related to the classes generated by hierarchical planning formalisms to show that all subset relations in the language hierarchy of planning formalisms are strict and that all intersections are non-empty. The overall hierarchy is given in Fig. 1. We concluded with a discussion of the properties of the novel expressivity measure and its relation to other approaches.

Acknowledgments

We want to thank the reviewers for their help to improve the paper. This work was done within the Transregional Collaborative Research Centre SFB/TRR 62 "Companion-Technology for Cognitive Technical Systems" funded by the German Research Foundation (DFG).

References

Alford, R.; Bercher, P.; and Aha, D. W. 2015a. Tight bounds for HTN planning. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 7–15. AAAI Press.

Alford, R.; Bercher, P.; and Aha, D. W. 2015b. Tight bounds for HTN planning with task insertion. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)*, 1502–1508. AAAI Press.

Alford, R.; Shivashankar, V.; Kuter, U.; and Nau, D. S. 2014. On the feasibility of planning graph style heuristics for HTN planning. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 2–10. AAAI Press.

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

Behnke, G.; Höller, D.; and Biundo, S. 2015. On the complexity of HTN plan verification and its implications for plan recognition. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*, 25–33. AAAI Press.

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

Carberry, S. 2001. Techniques for plan recognition. *User Modeling and User-Adapted Interaction* 11(1-2):31–48.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. HTN planning: Complexity and expressivity. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI 1994)*, volume 94, 1123–1128. AAAI Press.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1996. Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence* 18(1):69–93.

Geier, T., and Bercher, P. 2011. On the decidability of HTN planning with task insertion. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, 1955–1961. AAAI Press.

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2014. Language classification of hierarchical planning problems. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, 447–452. IOS Press.

Ingrand, F., and Ghallab, M. 2013. Robotics and artificial intelligence: a perspective on deliberation functions. *AI Communications* 27:63–80.

Krüger, V.; Kragic, D.; Ude, A.; and Geib, C. 2007. The meaning of action – a review on action recognition and mapping. *Advanced Robotics* 21(13):1473–1501.

Lang, J., and Zanuttini, B. 2012. Knowledge-based programs as plans – the complexity of plan verification. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, 504–509. IOS Press.

Nebel, B., and Bäckström, C. 1994. On the computational complexity of temporal projection, planning, and plan validation. *Artificial Intelligence* 66(1):125–160.

Nebel, B. 2000. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research (JAIR)* 12:271–315.

Röger, G.; Pommerening, F.; and Helmert, M. 2014. Optimal planning in the presence of conditional effects: Extending lm-cut with context splitting. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, 765–770. IOS Press.

Sukthankar, G.; Goldman, R. P.; Geib, C.; Pynadath, D. V.; and Bui, H. H. 2014. *Plan, Activity, and Intent Recognition*. Elsevier. chapter An Introduction to Plan, Activity, and Intent Recognition.