Knowledge Compilation for Nondeterministic Action Languages

Sergej Scheck, Alexandre Niveau, Bruno Zanuttini

Knowledge Compilation for Nondeterministic Action Languages

Sergej Scheck, Alexandre Niveau, Bruno Zanuttini
Normandie Univ.; UNICAEN, ENSICAEN, CNRS, GREYC, 14 000 Caen, France
sergej.scheck@unicaen.fr, alexandre.niveau@unicaen.fr, bruno.zanuttini@unicaen.fr

Abstract

We study different languages for representing nondeterministic actions in planning from the point of view of knowledge compilation. Precisely, we consider succintness issues (how succinct is the description of an action in each language?) and complexity issues (tractability or hardness of several queries which arise naturally in planning and belief tracking). We study an abstract, nondeterministic version of PDDL, nondeterministic conditional STRIPS, the language NNFAT of NNF action theories, and the language $\text{NPDDL}_{\text{seq}}$ obtained by adding a sequence operator to nondeterministic PDDL. We show that these languages have different succinctness and different complexity even for the most natural queries.

Introduction

In automated planning, a central aspect of the description of problems is the formal representation of actions. Such representations are indeed needed for specifying the actions available to the agent (PDDL (McDermott 1998) is a standard language for this), and also for the planners to operate on them while searching for a plan.

We consider different representation languages in the formal framework of the knowledge compilation map (Darwiche and Marquis 2002). This framework deals with the study of formal languages under the point of view of queries (how efficient is it to answer various queries depending on the language?), transformations (how efficient is it to transform or combine different representations in a given language?), and succinctness (how concise is it to represent knowledge in each language?).

The knowledge compilation map has been introduced for representations of Boolean functions, such as Boolean formulas in negation normal form, ordered binary decision diagrams, etc. (Darwiche and Marquis 2002). As far as we know there has been no systematic study of languages for representing actions per themselves. This is however an important problem, as planners need to query action representations again and again while searching for a plan (e.g., to find out which actions are applicable at the current node of the search tree). Hence having a clear picture of the properties of languages is of interest for developing such planners.

For instance, if a planner uses an efficient representation of actions (like binary decision diagrams) but this representation is exponentially larger than the specification of the action (for instance, in PDDL), then it may not be worth overall to convert the specification into the efficient representation, and this might suggest to make the planner work directly with the specification. The complexity of querying the various representations (like checking whether the action is applicable in some state, or whether it can lead from some state to another one) is also of importance for the development of algorithms: depending on the type of queries used by the algorithm, one language may be preferred to another one because it allows to answer them more efficiently. Of course, there is typically a tradeoff between succinctness of languages and tractability of queries.

However, there have been few papers studying these aspects for planning. Nebel (2000) has considered questions very similar to ours; his study uses a rather powerful notion of compilation, where translations from one formal language to another are allowed to change the set of variables and the set of actions. This captures compilation schemes where one is interested in preserving the existence of plans and their size. Contrastingly, we are interested in a strict notion of compilation, where the set of variables and the specification of initial states and goals are unchanged by the translation, while each action is translated into one with the same semantics. This is more restricted , but makes translations applicable in broader settings (for instance, to problems where we want to count or enumerate plans). Bäckström and Jonsson (2012) have studied representations of plans with respect to their size and to the complexity of retrieving the individual actions which they prescribe at each step . This is also related to our work, but with a focus on languages for representing plans, while we study languages for representing actions. Also related is the translation of HTN specifications into PDDL proposed by Alford, Kuter, and Nau (2009). The objects of interest there (HTN specifications) are different from ours (plain action specifications), though our study of action specifications using a construct for sequences is reminiscent of HTN decompositions.

We are interested in (purely) nondeterministic actions, as in fully observable nondeterministic planning and conformant planning (Rintanen 2004; Albore, Palacios, and Geffner 2010; Geffner and Bonet 2013; Muise, McIlraith,

and Belle 2014; To, Son, and Pontelli 2015; Geffner and Geffner 2018). We also focus on propositional domains, in which states are assignments to a given set of propositions.

We consider four representation languages: **NPDDL**, which is a grounded, nondeterministic version of **PDDL**, a language typically used for specifying actions (by a domain expert); **NSTRIPS**, a nondeterministic version of conditional STRIPS; **NNF** action theories, which are typically used as an internal representation by solvers (Cimatti, Roveri, and Traverso 1998; To, Son, and Pontelli 2015); and $\mathbf{NPDDL}_{\text{seq}}$, obtained by enriching **NPDDL** by a sequence operator, like in **DL-PA** (Balbiani, Herzig, and Troquard 2013). Despite different typical uses, these are all languages for representing actions (as nondeterministic mappings from states to states). We choose these because they use different constructs, different structural restrictions on expressions, and different representations of persisting values. Our midterm goal is to give a systematic picture of languages arising from combinations of such constructs and restrictions.

Orthogonally, we also study two concrete representations of expressions, as syntactic *trees* or as more compact *circuits*, where identical subexpressions are not repeated. The former representation gives a natural measure of the size of action specifications, while the latter is more compact and is the one typically used for algorithmic efficiency (in particular for binary decision diagrams (Bryant 1992)).

We first give background about actions and logic, then we define the action languages which we consider. We then give our results: polynomial-time translations between the languages, then results about the complexity of queries, and finally separation results, which allow us to determine the relative succinctness of the languages. Finally, we conclude.

Preliminaries

We consider a countable set of propositional *state* variables $\mathbb{P} = \{p_i \mid i \in \mathbb{N}\}$. Let $P \subset \mathbb{P}$ be a finite set of state variables; a subset of P is called a *P-state*, or simply a *state*. The intended interpretation of a state $s \in 2^P$ is the assignment to P in which all variables in s are true, and all variables in $P \setminus s$ are false. For instance, for $P = \{p_1, p_2, p_3\}$, $s = \{p_1, p_3\}$ denotes the state in which p_1, p_3 are true and p_2 is false. We write $\mathrm{V}(\varphi)$ for the set of variables occuring in an expression φ; note that expressions may involve both variables in $\mathbb{P}$ and variables not in $\mathbb{P}$, so in general we do *not* have $\mathrm{V}(\varphi) \subseteq \mathbb{P}$.

Actions We consider (purely) nondeterministic actions, which map states to sets of states. Hence a single state may have several successors through the same action.

Definition 1 (action)**.** Let $P \subset \mathbb{P}$ be a finite set of variables. A *P-action* is a mapping a from 2^P to $2^{(2^P)}$. The states in $a(s)$ are called *a-successors* of s.

Note that $a(s)$ is defined for all states s. We will consider a to be *applicable* in s if and only if $a(s) \neq \emptyset$.

Example 2. Consider a hunting example, described by whether the rabbit is in sight and/or alive, and whether the rifle is loaded. Let $P = \{\text{in\_sight}, \text{alive}, \text{loaded}\}$.

The action shoot_rabbit can be described as "if alive then: if loaded and in_sight, then not loaded, and either alive and not in_sight, or not alive and in_sight; otherwise state unchanged". The action is applicable only if the rabbit is alive (otherwise it is not sensible to shoot at him). Then, if the hunter is ready to shoot (the rifle is loaded and he sees the rabbit), he shoots the rabbit (he might miss the rabbit who hears the shot and runs away, so the action is nondeterministic), and if he is not ready to shoot, then nothing happens.

Let $s = \{\text{in\_sight}, \text{alive}, \text{loaded}\}$ be the state where all variables are true. Then shoot_rabbit(s) is the set of states ("successors") $\{s', s''\}$ with $s' = s \setminus \{\text{loaded}, \text{in\_sight}\} = \{\text{alive}\}$ and $s'' = s \setminus \{\text{loaded}, \text{alive}\} = \{\text{in\_sight}\}$.

We now introduce a formal setting for studying different *representations* of actions.

Definition 3 (action language)**.** An *action language* is an ordered pair $\langle L, I\rangle$, where L is a set of expressions and I is a partial function on $L \times 2^{\mathbb{P}}$ such that, when defined on $\alpha \in L$ and $P \subset \mathbb{P}$, $I(\alpha, P)$ is a P-action.

We call the expressions in L *action descriptions*, and call I the *interpretation function* of the language. Observe that those sets P's such that $I(\alpha, P)$ is defined are *a priori* not related to $\mathrm{V}(\alpha)$; α may involve auxiliary variables (not in $\mathbb{P}$) which are not part of the state descriptions, and dually, a state may assign variables of $\mathbb{P}$ which do not occur in α. If L, I, P are fixed or clear from the context, then we write $\alpha(s)$ instead of $I(\alpha, P)(s)$ for the set of all α-successors of s.

Definition 4 (translation)**.** A *translation* from an action language $\langle L_1, I_1\rangle$ to another language $\langle L_2, I_2\rangle$ is a function $f : L_1 \times 2^{\mathbb{P}} \to L_2$ such that $I_1(\alpha, P) = I_2(f(\alpha, P), P)$ holds for all $\alpha \in L_1$ and $P \subset \mathbb{P}$ such that $I_1(\alpha, P)$ is defined.

In words, this means that the L_1-expression α and the L_2-expression $f(\alpha, P)$ describe the same P-action. Again, when P is clear from the context, we write $f(\alpha)$ for $f(\alpha, P)$.

The translation f is said to be *polynomial-time* if it can be computed in time polynomial in the size of α and P, and *polynomial-size* if the size of $f(\alpha, P)$ is bounded by a fixed polynomial in the size of α and P. Clearly, a polynomial-time translation is necessarily also a polynomial-size one, but the converse is not true in general.

Negation Normal Form A Boolean formula φ over a set Q of variables is in *negation normal form* (NNF) if it is built up from literals using conjunctions and disjunctions, *i.e.*, if it is generated by the grammar $\varphi ::= q \mid \neg q \mid \varphi \wedge \varphi \mid \varphi \vee \varphi$, where q ranges over Q. Like other expressions, Q may involve state variables (in $\mathbb{P}$) and other variables (not in $\mathbb{P}$).

The set of NNF formulas is complete, i.e., any Boolean function can be described by an NNF formula. It is important to note that a formula φ with $\mathrm{V}(\varphi) \subseteq Q$ for some set of variables Q can be viewed as a formula over Q (and the truth value of the corresponding Boolean function does not depend on the variables in $Q \setminus \mathrm{V}(\varphi)$). For a Boolean formula φ over Q and an assignment t to Q, we write $t \models \varphi$ ("t satisfies φ") if φ evaluates to true under the assignment t. We always use notation $s, t, \ldots$ for states, $a, b, \ldots$ for actions, $\alpha, \beta, \ldots$ for action descriptions, and $\varphi, \psi, \ldots$ for logical formulas.

Action Languages

The first language which we consider is the well-known planning domain description language (**PDDL**). This language is a standardized one used for specifying actions at the relational level, widely used as an input for planners, especially in the international planning competitions (McDermott 1998; Fox and Long 2002, 2003). Since we are interested in nondeterministic actions, we consider a nondeterministic variant of **PDDL** inspired by **NPDDL** (Bertoli et al. 2003), and so as to abstract away from the precise syntax of the specification language, we consider an idealized version. Finally, we consider a grounded (propositional) version.

Definition 5 (**NPDDL**)**.** An **NPDDL** *action description* is an expression α generated by the grammar

$$\alpha \ ::= \varepsilon \mid +p \mid -p \mid \varphi \rhd \alpha \mid (\alpha \cup \alpha) \mid \alpha \sqcap \alpha$$

where p ranges over $\mathbb{P}$ and φ over formulas in NNF over $\mathbb{P}$.

Intuitively,

- ε describes the action with no effect ($\forall s\colon \varepsilon(s) = \{s\}$),
- $+p$ (resp. $-p$) is the action which sets p true (resp. false),
- $\rhd$ denotes conditional execution,
- $\cup$ denotes (exclusive) nondeterministic choice,
- $\sqcap$ denotes simultaneous execution,

and, importantly, variables not explicitly set by the action are assumed to keep their value. Also observe that auxiliary variables are not allowed — only variables in $\mathbb{P}$ can occur.

We insist that this syntax is an idealization of nondeterministic (grounded) **PDDL**; for instance, the action which we write $x \rhd \big(+y \cup (-y \sqcap +z)\big)$ would be written

```
when x (oneof y (and (not y) z))
```

with the syntax of **NPDDL** (Bertoli et al. 2003).

In order to define the interpretation function I of **NPDDL**, we first introduce a function J which for given α, P, s with $\mathrm{V}(\alpha) \subseteq P$ and $s \subseteq P$, gives a set of ordered pairs $\langle Q^+, Q^- \rangle$, where Q^+ (resp. Q^-) is a set of variables of P which are *explicitly* set to true (resp. to false) by α when executed in s (the other variables being left unchanged from s):

- $J(\varepsilon, P, s) := \{\langle \emptyset, \emptyset \rangle\}$,
- $J(+p, P, s) := \{\langle \{p\}, \emptyset \rangle\}$, and $J(-p, P, s) := \{\langle \emptyset, \{p\} \rangle\}$,
- $J(\varphi \rhd \alpha, P, s) := J(\alpha, P, s)$ if $s \models \varphi$, and $J(\varphi \rhd \alpha, P, s) := \{\langle \emptyset, \emptyset \rangle\}$ if $s \not\models \varphi$,
- $J(\alpha \cup \beta, P, s) := J(\alpha, P, s) \cup J(\beta, P, s)$,
- $J(\alpha \sqcap \beta, P, s) := \{\langle Q^+_\alpha \cup Q^+_\beta, Q^-_\alpha \cup Q^-_\beta \rangle \mid \langle Q^+_\alpha, Q^-_\alpha \rangle \in J(\alpha, P, s), \langle Q^+_\beta, Q^-_\beta \rangle \in J(\beta, P, s), Q^+_\alpha \cap Q^-_\beta = Q^-_\alpha \cap Q^+_\beta = \emptyset\}$

where the last line can be read "when executed in s, $\alpha \sqcap \beta$ has all those effects which are the consistent union of an effect of α in s and one of β in s". As an example, for $\alpha = (+p_1 \cup (-p_2 \sqcap +p_3)) \sqcap (-p_2 \cup +p_2)$, $P = \{p_1, p_2, p_3\}$, and any s, we have $J(\alpha, P, s) = \{\langle \{p_1\}, \{p_2\} \rangle, \langle \{p_1, p_2\}, \emptyset \rangle, \langle \{p_3\}, \{p_2\} \rangle\}$, since in the last combination, $-p_2 \sqcap +p_3$ and $+p_2$ are not consistent.[1]

[1] Note that this is in contrast to the usual semantics of STRIPS, where addition would override deletion, thus "forgetting" $-p_2$.

Then the interpretation function just formalizes the fact that the other variables retain their value; for all **NPDDL** expressions α and sets of variables P with $\mathrm{V}(\alpha) \subseteq P \subset \mathbb{P}$, $\forall s \subseteq P : I(\alpha, P)(s) := \{(s \cup Q^+) \setminus Q^- \mid \langle Q^+, Q^- \rangle \in J(\alpha, P, s)\}$ and $I(\alpha, Q)$ is undefined for $Q \not\supseteq \mathrm{V}(\alpha)$.

Note that the expression $+p \sqcap -p$ (for an arbitrary $p \in \mathbb{P}$) defines an action with no successor (which can be interpreted as an execution failing). We use $\bot$ as a shorthand for it (hence $I(\bot, P)(s) = \emptyset$ for all $P \subseteq \mathbb{P}, s \subseteq P$).

Example 6 (continued)**.** The following is an **NPDDL** action description for the action shoot_rabbit of Example 2.

$$(\neg \mathrm{alive} \rhd \bot) \sqcap \begin{pmatrix} (\mathrm{alive} \wedge \mathrm{in\_sight} \wedge \mathrm{loaded}) \\ \rhd (-\mathrm{loaded} \sqcap (-\mathrm{in\_sight} \cup -\mathrm{alive})) \end{pmatrix}$$

We also study a natural extension of conditional STRIPS to nondeterministic actions.

Definition 7 (**NSTRIPS**)**.** An **NSTRIPS** *action description* is an **NPDDL** expression of the form

$$\prod\nolimits_{i \in I} \varphi_i \rhd \big((\ell_i^{1,1} \sqcap \ldots \sqcap \ell_i^{1,j_1}) \cup \ldots \cup (\ell_i^{k_i,1} \sqcap \ldots \sqcap \ell_i^{k_i,j_{k_i}})\big)$$

where each $\ell_i^{k,j}$ is either $\bot$, or $+p$ or $-p$ for some $p \in \mathbb{P}$.

In words, an **NSTRIPS** action description specifies a set of conditions so that, when the action is applied in a state s, for each condition satisfied by s exactly one of the corresponding effects occurs.

Example 8 (continued)**.** The following is an **NSTRIPS** action description for the action shoot_rabbit of Example 2:

$$\begin{array}{ll} & (\neg \mathrm{alive} \rhd \bot) \\ \sqcap & \begin{pmatrix} (\mathrm{alive} \wedge \mathrm{in\_sight} \wedge \mathrm{loaded}) \rhd \\ ((-\mathrm{loaded} \sqcap -\mathrm{in\_sight}) \cup (-\mathrm{loaded} \sqcap -\mathrm{alive})) \end{pmatrix} \end{array}$$

The language **NSTRIPS** is complete, since any action a can at least be represented by one condition for each state s (satisfied by exactly s), associated to either (1) a choice ($\cup$) between some "conjunctions" of atoms, one conjunction per a-successor s' of s (setting all variables as in s'), or (2) to the degenerate choice of conjunctions $\bot$, when $a(s)$ is empty.

We are also interested in the language **NPDDL** as extended by the sequential execution operator ";".

Definition 9 (**NPDDL**$_{\mathrm{seq}}$)**.** An **NPDDL**$_{\mathrm{seq}}$ *action description* is an expression α generated by the grammar

$$\alpha \ ::= \ \varepsilon \mid +p \mid -p \mid \varphi \rhd \alpha \mid (\alpha \cup \alpha) \mid \alpha \sqcap \alpha \mid \alpha \,;\, \alpha$$

where p ranges over $\mathbb{P}$ and φ over formulas in NNF over $\mathbb{P}$.

Let us emphasize that ";" is not bound to occur only at the root of the expression. The interpretation function is the same as for **NPDDL** expressions, augmented with

$$\forall s \subseteq P\colon \ I(\alpha; \beta, P)(s) := \{s' \subseteq P \mid \exists t \in \alpha(s)\colon s' \in \beta(t)\}$$

Example 10. The **NPDDL**$_{\mathrm{seq}}$ description

$$\begin{array}{l} +p_{\mathrm{even}}; \\ \big(+p_1 \sqcap (p_{\mathrm{even}} \rhd -p_{\mathrm{even}}) \sqcap (\neg p_{\mathrm{even}} \rhd +p_{\mathrm{even}})\big) \ \cup \ -p_1; \\ \cdots \\ \big(+p_n \sqcap (p_{\mathrm{even}} \rhd -p_{\mathrm{even}}) \sqcap (\neg p_{\mathrm{even}} \rhd +p_{\mathrm{even}})\big) \ \cup \ -p_n; \\ \neg p_{\mathrm{even}} \rhd \bot \end{array}$$

describes an action which maps any state to the set of all states with an even number of p_i's, and p_{even}, set to true.

Since **NSTRIPS** is a sublanguage of **NPDDL** and **NPDDL**$_{\text{seq}}$, **NPDDL** and **NPDDL**$_{\text{seq}}$ are also complete.

We finally define the language of (NNF) action theories. Such representations are typically used by planners which reason explicitly on *sets of states* (aka *belief states*), since they allow for symbolic operations on belief states (Cimatti and Roveri 2000; Bryce, Kambhampati, and Smith 2006; To, Son, and Pontelli 2015). Note that such planners in fact use more efficient representations, like OBDDs or DNFs, which are defined by further restrictions on NNF. However, as our results show, NNF action theories are already more efficient (for queries) but larger (less succinct) than the other languages which we study, so this holds for all sublanguages of NNF, too. Then a sublanguage of NNF to work with can be chosen using the knowledge compilation map for Boolean functions (Darwiche and Marquis 2002).

To prepare the definition we associate an auxiliary variable $p' \notin \mathbb{P}$ to each variable $p \in \mathbb{P}$; p' denotes the value of p *after* the action took place, while p denotes the value *before*.

Definition 11 (**NNFAT**)**.** An **NNFAT** *action description* is a Boolean formula α in NNF over $P_\alpha \cup \{p' \mid p \in P_\alpha\}$ for some set of state propositions $P_\alpha \subset \mathbb{P}$.

The interpretation of an **NNFAT** action description α is defined for all $P \subseteq \mathbb{P}$ such that $P \supseteq \mathrm{V}(\alpha) \cap \mathbb{P}$ (that is, when all state propositions have a value) by

$$\forall s \subseteq P\colon\ I(\alpha, P)(s) = \{s' \mid s \cup \{p' \mid p \in s'\} \models \alpha\}$$

where $s \cup \{p' \mid p \in s'\}$ is the assignment to $P \cup \{p' \mid p \in P_\alpha\}$ induced by s, s'. For $P \not\supseteq \mathrm{V}(\alpha) \cap \mathbb{P}$, $I(\alpha, P)$ is not defined. In words, an **NNFAT** expression represents the set of all ordered pairs $\langle s, s' \rangle$ such that s' is a successor of s, as a Boolean formula over variables in $\mathbb{P} \cup \{p' \mid p \in \mathbb{P}\}$.

Importantly, **NNFAT** does not assume persistency of values, so that if, for example, a variable does not appear at all in an **NNFAT** expression, then this means that its value after the execution of the action can be arbitrary. For instance, $I(p'_1 \vee \neg p_2, \{p_1, p_2, p_3\})(\{p_2\}) = \{\{p_1\}, \{p_1, p_2\}, \{p_1, p_3\}, \{p_1, p_2, p_3\}\}$

Obviously, every action can be represented in **NNFAT**, since NNF is a complete language for Boolean functions.

Example 12 (continued)**.** The action shoot_rabbit of Example 2 can be written as follows (using $\rightarrow$ and $\not\leftrightarrow$ for readability, but keeping in mind that they are shorthand notation for longer NNF formulations):

$$\begin{array}{l} \text{alive} \\ \wedge \left(\begin{array}{ll} & (\text{loaded} \wedge \text{in\_sight}) \\ \rightarrow & \neg\text{loaded}' \wedge \left(\begin{array}{c} (\text{in\_sight}' \wedge \neg\text{alive}') \\ \vee\ (\neg\text{in\_sight}' \wedge \text{alive}') \end{array} \right) \end{array} \right) \\ \wedge \left(\left(\begin{array}{c} (\text{alive} \not\leftrightarrow \text{alive}') \\ \vee\ (\text{in\_sight} \not\leftrightarrow \text{in\_sight}') \\ \vee\ (\text{loaded} \not\leftrightarrow \text{loaded}') \end{array} \right) \rightarrow \left(\begin{array}{c} \text{loaded} \\ \wedge\ \text{in\_sight} \end{array} \right) \right) \end{array}$$

As can be seen, encoding the fact that the values of variables persist unless stated otherwise requires subexpressions (here the last conjunct) playing the same role as successor-state (or frame) axioms in the situation calculus (Reiter 1991). This typically requires much space, for what we give a formal meaning later in the paper (Proposition 32).

Representations Since we study the succinctness of languages, it is crucial to define the *size* of action descriptions. For this we consider two variants of each language.

The first variant corresponds to a representation of the expressions α in the language by their syntactic *tree* (including the Boolean formulas occuring in the expression, if any). The second variant corresponds to the representation of these expressions α by the directed acyclic graph, or *circuit*, obtained from the syntactic tree of α by iteratively identifying the roots of two isomorphic subexpressions to each other, until no more reduction is possible (like for binary decision diagrams (Bryant 1992)). Clearly, for all expressions α, the circuit associated to α in this manner is unique, and it can be computed in polynomial time from the tree or from a nonreduced circuit.

As an illustration, Figure 1 gives the tree (left) and the associated circuit (right) for the same **NPDDL**$_{\text{seq}}$ expression.

Depending on the representation, we obtain two languages based on **NNFAT**, two based on **NPDDL**, and two based on **NPDDL**$_{\text{seq}}$. We write **T-NNFAT**, **T-NPDDL**, and **T-NPDDL**$_{\text{seq}}$ for those languages with expressions represented as trees, and **C-NNFAT**, **C-NPDDL**, and **C-NPDDL**$_{\text{seq}}$ for those languages with expressions represented as reduced circuits. Since **NSTRIPS** is flat (the depth of the underlying graph is bounded), there is no difference between the circuit and the tree versions up to polynomial-time transformations, except for the representation of conditions φ; since, as it turns out, the representation of conditions does not affect the results in this paper, we only write **NSTRIPS** without specifying the representation, and we say that **NSTRIPS** is a sublanguage of **T-NPDDL** or **C-NPDDL**, meaning each time the corresponding version of **NSTRIPS**.

Finally, for all languages and representations, we write $|\alpha|$ for the size of an expression α in the representation, namely the number of nodes and edges in the tree or DAG.

Polynomial-Time Translations

In this section we give some (easy) results about polynomial-time translations between languages.

Proposition 13. *The identity function is a polynomial-time translation from* **NSTRIPS** *to* **T-NPDDL***, from* **NSTRIPS** *to* **C-NPDDL***, from* **T-NPDDL** *to* **T-NPDDL**$_{\text{seq}}$*, and from* **C-NPDDL** *to* **C-NPDDL**$_{\text{seq}}$.

We now proceed to show that there is a polynomial-time translation from **NNFAT** to **NPDDL**, for both representations. The translation looks like a simple rewriting of α, but we have to take care about the fact that in **NNFAT**, a variable not explicitly set to a value can take any value in the next state s', contrary to persistency by default in **NPDDL**.

Proposition 14. *There is a polynomial-time translation from* **T-NNFAT** *to* **T-NPDDL***, and from* **C-NNFAT** *to* **C-NPDDL**.

PROOF. Let f be defined inductively as follows, for all **NNFAT** expressions α and sets of variables $P \supseteq \mathrm{V}(\alpha) \cap \mathbb{P}$:

1. for $p \in P$, $f(p) = (\neg p \rhd \bot) \sqcap \left(p \rhd \bigsqcap_{q \in P} (+q \cup -q) \right)$;

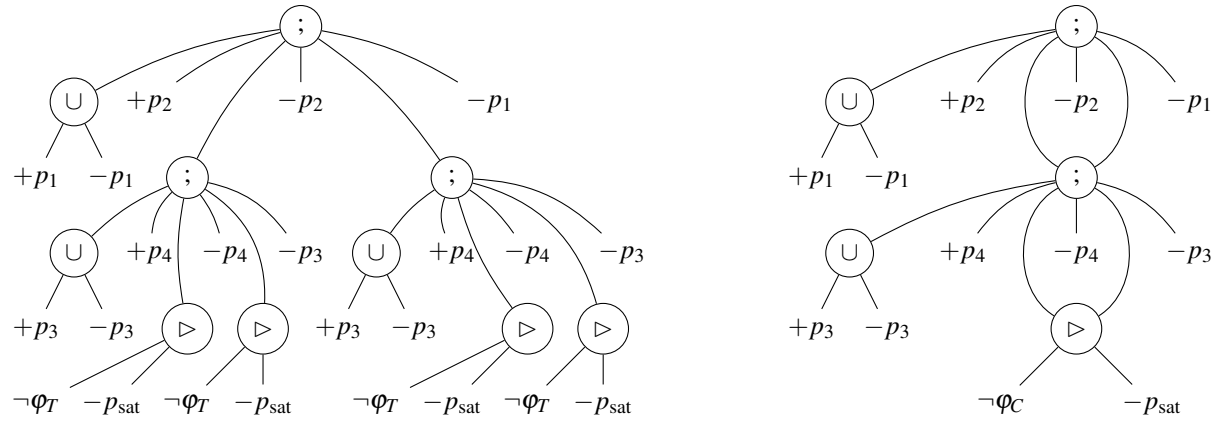

Figure 1: Tree (left) and circuit (right) representations of the same **NPDDL**$_{\text{seq}}$ action. Children are ordered from left to right; φ_T (resp. φ_C) denotes a tree (resp. circuit) for φ. On the right diagram, nodes $-p_1$ and $-p_3$ are duplicated for readability.

2. for $p \in P$, $f(\neg p) = (p \rhd \bot) \sqcap \left(\neg p \rhd \bigsqcap_{q \in P}(+q \cup -q)\right)$;
3. for $p \in P$, $f(p') = +p \sqcap \left(\bigsqcap_{q \in P, q \neq p}(+q \cup -q)\right)$;
4. for $p \in P$, $f(\neg p') = -p \sqcap \left(\bigsqcap_{q \in P, q \neq p}(+q \cup -q)\right)$;
5. $f(\beta \wedge \gamma) = f(\beta) \sqcap f(\gamma)$;
6. $f(\beta \vee \gamma) = f(\beta) \cup f(\gamma)$.

Items 1–4 take care of the fact that there is no implicit persistency of values in **NNFAT**. Then it is easy to show by induction that f is indeed a translation from **NNFAT** to **NPDDL**. In particular, Case 5 works despite the different semantics of $\wedge$ and $\sqcap$ because (by induction) all effects are explicit in both $f(\beta)$ and $f(\gamma)$, so that $\sqcap$ acts as $\wedge$ in this precise translation. Case 6 works despite the fact that $\vee$ is inclusive but $\cup$ is not because the effects brought about by β *and* γ are by definition brought about by, say, β alone and hence, by induction, by $f(\beta)$ alone, so that $\cup$ acts as $\vee$ in this translation.

Since f rewrites each node of α independently of the others, it is polynomial-time both when applied to an expression in **T-NNFAT** (with a result in **T-NPDDL**) and to an expression in **C-NNFAT** (with a result in **C-NPDDL**). □

Complexity of Queries

We now turn to studying the complexity of *queries* to expressions. We concentrate on three natural queries for planning, corresponding to checking the existence of a transition, deciding applicability of an action, and deciding whether a (sequential) plan reaches a goal. Let $\langle L, I\rangle$ be a fixed action language, and let $\alpha, \alpha_1, \ldots, \alpha_k$ denote expressions in L, $P \subseteq \mathbb{P}$ denote a set of variables such that $I(\alpha, P)$ and $I(\alpha_1, P), \ldots, I(\alpha_k, P)$ are defined, and s, s' denote P-states.

Definition 15 (IS-SUCC)**.** The decision problem IS-SUCC takes as input α, P, s, s', and asks whether $s' \in \alpha(s)$.

Definition 16 (IS-APPLIC)**.** The decision problem IS-APPLIC takes as input α, P, s, and asks whether $\alpha(s) \neq \emptyset$.

Definition 17 (ENTAILS)**.** The decision problem ENTAILS takes as input $\alpha_1, \ldots, \alpha_k, P, s$, and an **NNF** formula φ over P, and asks whether s' satisfies φ for all $s' \in (\alpha_1; \ldots; \alpha_k)(s)$.

It is important to note that if s has no successor through $\alpha_1; \ldots; \alpha_k$, then it entails any formula φ.

It turns out that the query IS-SUCC is central, hence we start with it.

Proposition 18. IS-SUCC *can be solved in linear time for* **T-NNFAT** *and* **C-NNFAT**.

PROOF. By definition of **NNFAT**, $s' \in \alpha(s)$ holds if and only if the assignment to $P \cup \{p' \mid p \in P\}$ induced by s, s' satisfies α, which can be decided by replacing each leaf in the representation of α by its value in s or s', then evaluating α in a bottom-up fashion. Clearly, this can be done in linear time for both tree and circuit representations. □

Proposition 19. IS-SUCC *is in* **NP** *for* **T-NPDDL**$_{\text{seq}}$.

PROOF. We define a witness for a positive instance to be composed of either β or γ for each subexpression $\beta \cup \gamma$ of α. Such a witness is clearly of polynomial size, and verifying it amounts to deciding successorship for a deterministic **NPDDL**$_{\text{seq}}$ description, which is clearly doable in polynomial time by simulating the (only) execution. □

For hardness, we first define an encoding of a 3-CNF formula φ over n variables as an assignment to polynomially many state variables.

Notation 20. Let $n \in \mathbb{N}$, and let $X_n = \{x_1, \ldots, x_n\}$ be a set of variables. Observe that there are a cubic number N_n of clauses of length 3 over X_n. We fix an arbitrary enumeration $\gamma_1, \gamma_2, \ldots, \gamma_{N_n}$ of all these clauses, and we define $P_n \subset \mathbb{P}$ to be the set of state variables $\{p_1, p_2, \ldots, p_{N_n}\}$. Write $\ell \in \gamma_i$ if the literal ℓ occurs in the clause γ_i. Then to any 3-CNF formula φ we associate the P_n-state $s(\varphi) = \{p_i \mid i \in \{1, \ldots, N_n\}, \gamma_i \in \varphi\}$, and dually, to any P_n-state s, we associate the 3-CNF formula over X_n, written $\varphi(s)$, which contains exactly those clauses γ_i for which $p_i \in s$ holds.

Example 21. Consider an enumeration of all clauses over $X_2 = \{x_1, x_2\}$ which starts with $\gamma_1 = (x_1 \vee x_1 \vee x_2), \gamma_2 = (x_1 \vee x_1 \vee \neg x_2), \gamma_3 = (x_1 \vee \neg x_1 \vee x_2), \ldots$ Then the 3-CNF $\varphi = (x_1 \vee x_1 \vee x_2) \wedge (x_1 \vee \neg x_1 \vee x_2)$ is encoded by $s(\varphi) = \{p_1, p_3\}$.

Proposition 22. IS-SUCC *is* **NP**-*hard for* **NSTRIPS**.

PROOF. We give a reduction from 3-SAT. Let φ be a 3-CNF formula over variables $x_1,\dots,x_n$ and containing (w.l.o.g.) the clauses $\gamma_1,\dots,\gamma_k$ (encoded by $p_1,\dots,p_k$). We define

$$\alpha \coloneqq \prod_{x_i} \left(\top \rhd \left(\Big(\prod_{\gamma_j \in \varphi\colon x_i \in \gamma_j} +p_j \Big) \cup \Big(\prod_{\gamma_j \in \varphi\colon \neg x_i \in \gamma_j} +p_j \Big) \right) \right)$$

Clearly, α is a polynomial-time constructible **NSTRIPS** expression. We claim that the state $\{p_1,\dots,p_k\}$ is an α-successor of $\emptyset$ if and only if φ is satisfiable. Indeed, α can be seen as deciding nondeterministically, for every variable x_i, whether to set it true or false, and then setting p_j true for all γ_j which are satisfied by the chosen value. Thus, if all p_j's have been set to true, all clauses are satisfied by the chosen assignment to the x_i's, and hence φ is satisfiable. Conversely, if φ is satisfiable, then there exists a consistent choice of literals $\ell_1,\dots,\ell_n$ which satisfies all γ_j's. Then the execution of α which, for each x_i, chooses the left or the right side of the $\cup$ choice according to the polarity of ℓ_i, indeed witnesses that $\{p_1,\dots,p_k\}$ is an α-successor of $\emptyset$. □

As **NSTRIPS** is a sublanguage of **NPDDL** and **NPDDL**$_{\text{seq}}$, using Propositions 19 and 22 we get:

Corollary 23. IS-SUCC *is* **NP**-*complete for* **NSTRIPS**, **T-NPDDL**, *and* **T-NPDDL**$_{\text{seq}}$.

IS-SUCC is **NP**-complete for **C-NPDDL** as well:

Proposition 24. IS-SUCC *is* **NP**-*complete for* **C-NPDDL**.

PROOF. Hardness follows from Corollary 23, since **NSTRIPS** is a sublanguage of **C-NPDDL**. For membership, the intuition is to guess a branch to be chosen for each $\cup$-node in the circuit as in the proof of Proposition 19. However, in circuits, a $\cup$-node might have exponentially many paths from the root to it, inducing parallel executions, and in each of these a new child may be chosen.[2]

For that reason, we define a witness of $s' \in \alpha(s)$ to be the number of times each node in the circuit is executed along an execution of α leading from s to s'. Hence a witness is a labelling of each node and edge in the circuit by a number in $[0..2^{|\alpha|}]$ (using $O(|\alpha|)$ bits per node). We then claim that such a labelling is indeed a witness that $s' \in \alpha(s)$ holds, if and only if the following six conditions hold:

1. the root is labelled with 1,
2. each edge out of a $\sqcap$-node has the same label as its source,
3. the labels of the edges out of a $\cup$-node sum up to the label of their source,
4. the edge from a $\rhd$-node α of the form $(\varphi \rhd \beta)$ to node β has the same label as α if s satisfies φ, and 0 otherwise,
5. the label of each node is the sum of the labels of the edges pointing to it,
6. the action $\ell_1 \sqcap \dots \sqcap \ell_k$, where $\ell_1,\dots,\ell_k$ are those leaves of the circuit (that is, elementary assignments $\pm p$) whose label is nonzero, maps s to s' (in particular, these assignments are consistent together).

[2]For instance, the bottom left $\cup$ node in the expression of Figure 1 (right) is executed twice along any execution of the action.

Clearly, this can be verified in polynomial time. Now it can be checked by induction that a labelling of the circuit of α is correct, as defined by Items 1–5 above, if and only if the action $\ell_1 \sqcap \dots \sqcap \ell_k$ of Item 6 is a valid execution of α, which concludes the proof. □

The proof of Proposition 24 relies on the fact that there is no sequence operator in **C-NPDDL**, and hence that when a node is executed several times, the order does not matter. For this reason, it cannot be generalized to **C-NPDDL**$_{\text{seq}}$.

Proposition 25. IS-SUCC *is* **PSPACE**-*complete for* **C-NPDDL**$_{\text{seq}}$.

PROOF. For membership, observe that we can simulate all possible executions (induced by nondeterministic choices) of the circuit in polynomial space, and simply decide whether one witnesses that s' is a successor of s.

Now for hardness, we give a reduction from the problem of deciding the validity of a QBF of the form $\Phi = \exists p_1 \forall p_2 \exists p_3 \forall p_4 \dots \exists p_{2n-1} \forall p_{2n}\colon \varphi$, where φ is a 3-CNF formula. Given such a QBF, let $p_{\text{sat}} \in \mathbb{P}$ be a fresh variable, define α_n to be the **NPDDL**$_{\text{seq}}$ expression $\neg\varphi \rhd -p_{\text{sat}}$, and for $i = n-1,\dots,0$, define α_i to be

$$(+p_{2i+1} \cup -p_{2i+1})\,;+p_{2i+2}\,;\alpha_{i+1}\,;-p_{2i+2}\,;\alpha_{i+1};-p_{2i+1}$$

Clearly, α_0 has a circuit of size polynomial in φ (*e.g.*, Fig. 1 (right) depicts α_0 for $n=2$).

Intuitively, the action guesses a value for p_{2i+1}, then verifies that together with p_{2i+2} set to true, the nested formula α_{i+1} is valid, then that together with p_{2i+2} set to false instead, α_{i+1} is again valid, and finally resets p_{2i+1} to false. Then it can be shown that the reduction is correct, using induction on $i = n, n-1,\dots,0$ with the following hypothesis:

> for all literals $\ell_1,\dots,\ell_{2i}$ over $p_1,\dots,p_{2i}$, respectively, the state $s = \{p_{\text{sat}}\} \cup \{p_j \mid \ell_j = p_j\}$ is an α_i-successor of itself if and only if $\Phi_i \coloneqq \exists p_{2i+1} \forall p_{2i+2} \dots \exists p_{2n-1} \forall p_{2n} \varphi_{|\ell_1,\dots,\ell_{2i}}$ is valid,

where $\varphi_{|\ell_1,\dots,\ell_{2i}}$ denotes propositional conditioning, that is, φ with the occurrences of $p_1,\dots,p_{2i}$ replaced by their value and simplified. In the end, we obtain that $s = \{p_{\text{sat}}\}$ is an α_0-successor of itself if and only if Φ_0 is valid, which concludes the proof since by construction we have $\Phi_0 = \Phi$. □

We now turn to the query IS-APPLIC.

Proposition 26. IS-APPLIC *is* **NP**-*complete for* **T-NNFAT** *and* **C-NNFAT**.

PROOF. Membership is clear, because applicability of α in s can be justified by giving a successor, which can be verified in linear time for both **T-NNFAT** and **C-NNFAT** by Proposition 18. For hardness, let φ be a propositional formula over variables $p_1,\dots,p_n$, and let α be the **T-NNFAT** expression obtained from φ by replacing each p_i with p'_i. Then clearly, α is applicable in any state (say in $s = \emptyset$) if and only if φ is satisfiable. Hence IS-APPLIC is **NP**-hard for **T-NNFAT**, and clearly this entails **NP**-hardness for **C-NNFAT** also. □

For the other languages, we will use the following lemma.

Lemma 27. IS-SUCC *is polynomial-time reducible to* IS-APPLIC *for* **NSTRIPS**, **T-NPDDL**, **C-NPDDL**, **T-NPDDL**$_{\text{seq}}$ *and* **C-NPDDL**$_{\text{seq}}$.

PROOF. For L any of these languages, let α be an L expression and s, s' be two states. Then $s' \in \alpha(s)$ if and only if

$$\beta := \alpha \sqcap \left(\top \rhd \left(\left(\bigsqcap_{p \in s' \setminus s} +p \right) \sqcap \left(\bigsqcap_{p \in s \setminus s'} -p \right) \right) \right)$$

is applicable in s. Indeed, β is just the action α executed simultaneously with the deterministic action which necessarily leads from s to s'. Thus β is applicable in s if and only if α allows the transition from s to s'. Now it is easy to see that β is indeed an L expression and can be built in polynomial time, which concludes the proof. □

Corollary 28. IS-APPLIC *is* **NP**-*complete for* **NSTRIPS**, **T-NPDDL**, **C-NPDDL**, *and* **T-NPDDL**$_{\text{seq}}$, *and it is* **PSPACE**-*complete for* **C-NPDDL**$_{\text{seq}}$.

PROOF. Hardness follows from Lemma 27 together with the previous results about IS-SUCC. For membership in **NP**, since IS-SUCC is in **NP** for all these languages, it suffices to define a witness for IS-APPLIC with input α, s, to be a successor s' together with a witness for $s' \in \alpha(s)$. Finally, for membership in **PSPACE**, it suffices to guess a successor and verify it in polynomial space (Proposition 25). □

We finally turn to the query ENTAILS.

Proposition 29. ENTAILS *is* **coNP**-*complete for* **T-NNFAT**, **C-NNFAT**, **NSTRIPS**, **T-NPDDL**, **C-NPDDL**, *and* **T-NPDDL**$_{\text{seq}}$. *Hardness holds even if the input sequence is restricted to contain only one action.*

PROOF. For membership, recall that deciding successorship is in **NP** for all the languages. Fix a language L. Suppose we are given a state s, a formula φ, and L-actions $\alpha_1, \ldots, \alpha_k$. To show that the given sequence of actions does *not* entail φ it is enough to guess a sequence of states $s_1, s_2, \ldots s_k$ with $s_1 \in \alpha_1(s), s_2 \in \alpha_2(s_1), \ldots, s_k \in \alpha_k(s_{k-1})$, and $s_k \not\models \varphi$, and guess the corresponding certificates for $s_i \in \alpha_i(s_{i-1})$. Then the non-entailment can be verified in polynomial time.

For hardness, fix $P = \{p_1, \ldots, p_n\}$. Observe that all languages can encode the "omnipotent" action a ($\forall s, a(s) := 2^P$) efficiently, either by $\alpha = p'_1 \vee \neg p'_1$ (for **NNFAT**) or by $\alpha = \bigsqcap_{p \in P}(+p \cup -p)$. Now let φ be a 3-CNF formula over P. Then φ is *not* satisfiable if and only if all assignments to P satisfy $\neg\varphi$, which amounts to all α-successors of an arbitrary state (say $s = \emptyset$) satisfying $\neg\varphi$, which is indeed an entailment problem with a single action. □

Proposition 30. ENTAILS *is* **PSPACE**-*complete for* **C-NPDDL**$_{\text{seq}}$.

PROOF. Suppose we are given a state $s \subseteq P$, a formula φ over $P = \{p_1, \ldots, p_n\}$, and $\alpha_1 ; \ldots ; \alpha_k$ with α_i in **C-NPDDL**$_{\text{seq}}$. **PSPACE**-membership is obvious for the same reason as for IS-SUCC: we simulate every execution of $\alpha_1 ; \ldots ; \alpha_k$ (each time using a polynomial amount of space) and check whether the resulting state satisfies φ. Now we recall that if a state has no successors through α then it entails any formula φ. Hardness follows from a reduction of the complement of IS-APPLIC to ENTAILS: α is not applicable in s if and only if $\alpha ; -p_1$ entails φ in s, with $\varphi := p_1$. Indeed, if α is applicable in s, then $\alpha ; -p_1$ leads to at least one state s' with $p_1 \notin s'$, while if it not applicable in s, then all its successors (vacuously) satisfy φ. □

Succinctness

We now give negative results about the existence of polynomial-*size* translations, hence about the relative *succinctness* of languages (Darwiche and Marquis 2002). Recall that all the languages which we study are complete.

Our separation results use the *nonuniform* complexity class $\mathbf{P/poly}$ of all decision problems such that for all $n \in \mathbb{N}$, there is a polynomial-time algorithm which decides the problem for all inputs of size n (Arora and Barak 2009). The assumption $\mathbf{NP} \not\subseteq \mathbf{P/poly}$ which we use is a standard one; in particular, $\mathbf{NP} \subseteq \mathbf{P/poly}$ would imply a collapse of the polynomial hierarchy at the second level (Karp-Lipton theorem).

We start with **NPDDL** and **NNFAT**. Using Notation 20, for all $n \in \mathbb{N}$ we define the **NPDDL** (but not **NSTRIPS**) expression α_n^{sat} by

$$\alpha_n^{\text{sat}} = \bigsqcap_{x \in X_n} \left(\left(\bigsqcap_{\gamma_i :\, x \in \gamma_i} (+p_i \cup \varepsilon) \right) \bigcup \left(\bigsqcap_{\gamma_i :\, \neg x \in \gamma_i} (+p_i \cup \varepsilon) \right) \right)$$

Intuitively, α_n^{sat} chooses an assignment (true or false) to each variable in X_n (outermost $\cup$). Whenever it chooses an assignment for x, for each possible clause which is satisfied by this assignment (innermost $\sqcap$), it chooses whether to include this clause into the result, or not(innermost $\cup$). Hence it builds a formula which is satisfied at least by its choices, and clearly, any satisfiable 3-CNF formula can be built in this manner.

Lemma 31. *Let $n \in \mathbb{N}$, and let φ be a* 3-CNF *formula over X_n. Then φ is satisfiable if and only if $s(\varphi)$ is an α_n^{sat}-successor of the state $\emptyset$.*

Proposition 32. *There is no polynomial-size translation from* **T-NPDDL** *to* **T-NNFAT**, *nor from* **C-NPDDL** *to* **C-NNFAT**, *unless* $\mathbf{NP} \subseteq \mathbf{P/poly}$ *holds.*

PROOF. The size of α_n^{sat} is clearly polynomial in n in both the tree and circuit representations. Assume that there is a polynomial-size translation f from **T-NPDDL** to **T-NNFAT**, and for all $n \in \mathbb{N}$ let $\beta_n^{\text{sat}} = f(\alpha_n^{\text{sat}})$. Then the following is a nonuniform polynomial time algorithm for the 3-SAT problem; given a 3-CNF φ :

1. encode φ into a state $s(\varphi)$ as in Notation 20;
2. decide whether $s(\varphi)$ is a β_n^{sat}-successor of the state $\emptyset$;
3. if yes, claim that φ is satisfiable, otherwise unsatisfiable.

All steps are polynomial-time (Proposition 18), the algorithm is correct (Lemma 31), and it depends only on the number of variables in φ (which is polynomially related to its size), hence this is indeed a nonuniform polynomial time algorithm for 3-SAT. Hence $\mathbf{NP} \subseteq \mathbf{P/poly}$ holds.

The proof is exactly the same for circuits. □

Before proceeding to show that **NPDDL**$_{\text{seq}}$ is strictly more succinct than **NPDDL**, we need the following technical result about restricting an action to pairs of states $\langle s, s' \rangle$ which satisfy a given assignment to a subset of the variables.

Lemma 33. *Let P and Q be disjoint sets of variables, and α be a* **T-NPDDL** *(resp.* **C-NPDDL***) expression for a $(P \cup Q)$-action. Let $t \subseteq Q$ be an assignment to the variables in Q. Then we can compute a* **T-NPDDL** *(resp.* **C-NPDDL***) expression $\alpha_{|t}$ for a P-action which satisfies $\alpha_{|t}(s) = \{s' \setminus t \mid s' \in \alpha(s \cup t), s' \cap Q = t\}$ in time polynomial in $|\alpha|$.*

Query	**NSTRIPS**	**T-NNFAT**	**C-NNFAT**	**T-NPDDL**	**C-NPDDL**	**T-NPDDL**$_{\text{seq}}$	**C-NPDDL**$_{\text{seq}}$
IS-SUCC	**NP**-cpl.	lin. time	lin. time	**NP**-cpl.	**NP**-cpl.	**NP**-cpl.	**PSPACE**-cpl.
IS-APPLIC	**NP**-cpl.	**NP**-cpl.	**NP**-cpl.	**NP**-cpl.	**NP**-cpl.	**NP**-cpl.	**PSPACE**-cpl.
ENTAILS	**coNP**-cpl.	**coNP**-cpl.	**coNP**-cpl.	**coNP**-cpl.	**coNP**-cpl.	**coNP**-cpl.	**PSPACE**-cpl.

Table 1: Summary of the complexity results for the queries. The languages (except **NSTRIPS**) are ordered according to their succinctness, i.e. the tree (circuit) representation of a language on the left is strictly less succinct than the tree (circuit) representation of a language on the right. We do not compare succincntess of tree and circuit representations.

PROOF. Simply replace all leaves of the form $+q$ with $q \in Q \cap t$ by ε, all leaves of the form $+q$ with $q \in Q \setminus t$ by $\bot$ (failure), and dually for $-q$, and for all subexpressions $\varphi \rhd \beta$, simplify φ by the assignment t to Q. □

Proposition 34. *There is no polynomial-size translation from* **T-NPDDL**$_{\text{seq}}$ *to* **T-NPDDL**, *nor from* **C-NPDDL**$_{\text{seq}}$ *to* **C-NPDDL**, *unless* $\mathbf{NP} \subseteq \mathbf{P/poly}$ *holds.*

PROOF. Let $n \in \mathbb{N}$, and let φ be a 3-CNF formula over a set of variables $\{p_1^\varphi, \dots, p_n^\varphi\} \subseteq \mathbb{P}$. Recall from Notation 20 that we can encode φ over a set $P_n \subseteq \mathbb{P}$ (we assume w.l.o.g. $P_n \cap \{p_1^\varphi, \dots, p_n^\varphi\} = \emptyset$). Finally, let $p_{\text{sat}} \in \mathbb{P}$ be a fresh variable. We define γ_n^{sat} to be the following **NPDDL**$_{\text{seq}}$ expression:

$$\prod_{i=1}^{n}(+p_i^\varphi \cup -p_i^\varphi); (\psi_n \rhd +p_{\text{sat}}) \sqcap (\neg\psi_n \rhd -p_{\text{sat}}); \prod_{i=1}^{n} -p_i^\varphi$$

where ψ_n is the NNF $\bigwedge_{i=1}^{N_n}(\neg p_i \vee \bigvee_{\ell \in \gamma_i} \ell)$, which is satisfied if and only if each clause γ_i which is in φ (as witnessed by p_i being true) is also satisfied. In words, γ_n^{sat} guesses an assignment to $\mathrm{V}(\varphi)$, then sets p_{sat} according to whether φ is satisfied by this assignment, and finally resets all guessed variables to false. Note that γ_n^{sat} depends on n but not on φ, and that γ_n^{sat} is polynomial in n.

Clearly, $s(\varphi) \cup \{p_{\text{sat}}\}$ is a γ_n^{sat}-successor of $s(\varphi)$ if and only if φ is satisfiable. Hence the following decision problem is **NP**-hard:

- *Input:* a 3-CNF formula φ
- *Question:* is $s(\varphi) \cup \{p_{\text{sat}}\}$ a γ_n^{sat}-successor of $s(\varphi)$?

Now assume that there is a polynomial-size translation f from **T-NPDDL**$_{\text{seq}}$ to **T-NPDDL**. Fix $n \in \mathbb{N}$, and let $\delta_n^{\text{sat}} = f(\gamma_n^{\text{sat}})$. Let φ be a 3-CNF formula. Since δ_n^{sat} is in **NPDDL**, we can apply Lemma 33 with $Q = P_n \cup \{p_1^\varphi, \dots, p_n^\varphi\}$ and $t = s(\varphi)$, to get an expression in which the only occuring variable is p_{sat}, and $\{p_{\text{sat}}\}$ is a successor of $\emptyset$ if and only if $s(\varphi) \cup \{p_{\text{sat}}\}$ is a δ_n^{sat}-sucessor of $s(\varphi)$, that is, if and only if φ is satisfiable. Now since this expression has only one variable, it is easy to see that successorship can be decided in polynomial time, implying $\mathbf{NP} \subseteq \mathbf{P/poly}$.

The proof is exactly the same for circuits. □

Conclusion

We studied several languages for describing nondeterministic actions along two criteria: succinctness and complexity of different decision problems natural to automated planning. We also considered two representations, by trees and by circuits. Our results are summarized in Table 1, where we have ordered languages with respect to their relative succinctness (we emphasize that we did not study the relative succinctness of two representations of the same language).

One conclusion is that converting an **NPDDL** specification into an **NNFAT** representation necessarily yields an explosion in some cases. Hence it can be worth developing planners which tackle directly the PDDL specification (Lesner and Zanuttini (2011) proposes one for the stochastic case), or, dually, to ask the experts to specify actions directly in **NNFAT**, so as to avoid the conversion. The latter may indeed make sense in some settings, since **NNFAT** can be seen as a declarative language (it is easy to specify that the action sets p_1 to the same value as p_2, for instance), quite complementary to **PDDL**, which is more "imperative".

Also interesting is the different complexity of queries for **NPDDL**$_{\text{seq}}$ with trees or circuits. While it is intuitively clear that there must be some languages which are strictly less compact with trees than with circuits, and languages with less tractable queries with circuits than with trees, this gives a concrete example of this phenomenon. Finally, it is interesting that the complexity of the three queries is the same for tree-represented **NPDDL** with or without sequence. This means that as far as these queries are concerned, one can let an expert use the richer language for specifying actions, and a solver can just compile away sequence operators efficiently and work with plain **NPDDL**, as if the specification was constrained to be in this language.

Our picture is almost complete for the languages which we studied: we only leave open the relative succinctness of **NSTRIPS** and **NNFAT** on the one hand, and **NSTRIPS** and **NPDDL** on the other hand. Our main perspective is a more systematic study, for languages constructed using combinations of features like the sequence operator, modalities, Kleene star, etc. A very expressive language to consider is **DL-PPA** (Herzig, Maris, and Vianey 2019). Another perspective is to consider languages for stochastic actions, and for actions with observations (and hence queries on *belief states* rather than states). We are also interested in studying the complexity of other queries, e.g. counting and enumerating successors or randomly generating a successor state. The ultimate goal is to draw clear pictures of what language to choose depending on the queries which are used by, *e.g.*, a planning algorithm or a simulator.

Acknowledgements

This work has been supported by the French National Research Agency (ANR) through project PING/ACK (ANR-18-CE40-0011).

References

Albore, A.; Palacios, H.; and Geffner, H. 2010. Compiling Uncertainty Away in Non-Deterministic Conformant Planning. In *Proc. 19th European Conference on Artificial Intelligence (ECAI 2010)*, volume 215, 465–470.

Alford, R.; Kuter, U.; and Nau, D. 2009. Translating HTNs to PDDL: A small amount of domain knowledge can go a long way. In *Proc. 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 1629–1634.

Arora, S.; and Barak, B. 2009. *Computational complexity: a modern approach*. Cambridge University Press.

Bäckström, C.; and Jonsson, P. 2012. Algorithms and Limits for Compact Plan Representations. *Journal of Artificial Intelligence Research* 44: 141–177.

Balbiani, P.; Herzig, A.; and Troquard, N. 2013. Dynamic logic of propositional assignments: a well-behaved variant of PDL. In *Proc. 28th Annual ACM/IEEE Symposium on Logic in Computer Science (LiCS 2013)*, 143–152.

Bertoli, P.; Cimatti, A.; Dal Lago, U.; and Pistore, M. 2003. Extending PDDL to nondeterminism, limited sensing and iterative conditional plans. In *Proc. ICAPS 2003 Workshop on PDDL*.

Bryant, R. E. 1992. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR)* 24(3): 293–318.

Bryce, D.; Kambhampati, S.; and Smith, D. E. 2006. Planning Graph Heuristics for Belief Space Search. *Journal of Artificial Intelligence Research* 26: 35–99.

Cimatti, A.; and Roveri, M. 2000. Conformant Planning via Symbolic Model Checking. *Journal of Artificial Intelligence Research* 13: 305–338.

Cimatti, A.; Roveri, M.; and Traverso, P. 1998. Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proc. 15th National Conference on Artificial Intelligence (AAAI 1998)*, 875–881.

Darwiche, A.; and Marquis, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research* 17: 229–264.

Fox, M.; and Long, D. 2002. The Third International Planning Competition: Temporal and Metric Planning. In *Proc. 6th International Conference on Artificial Intelligence Planning Systems (AIPS 2002)*, 333–335.

Fox, M.; and Long, D. 2003. PDDL2. 1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20: 61–124.

Geffner, H.; and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers.

Geffner, T.; and Geffner, H. 2018. Compact Policies for Fully Observable Non-Deterministic Planning as SAT. In *Proc. 28th International Conference on Automated Planning and Scheduling (ICAPS 2018)*, 88–96.

Herzig, A.; Maris, F.; and Vianey, J. 2019. Dynamic logic of parallel propositional assignments and its applications to planning. In *Proc. 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, 5576–5582.

Lesner, B.; and Zanuttini, B. 2011. Efficient Policy Construction for MDPs Represented in Probabilistic PDDL. In *Proc. 21st International Conference on Automated Planning and Scheduling (ICAPS 2011)*.

McDermott, D. 1998. PDDL–the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control. Available at: `www.cs.yale.edu/homes/dvm` (consulted on 2020/03/16).

Muise, C. J.; McIlraith, S. A.; and Belle, V. 2014. Non-Deterministic Planning With Conditional Effects. In *Proc. 24th International Conference on Automated Planning and Scheduling (ICAPS 2014)*, 370–374.

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

Reiter, R. 1991. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Lifschitz, V., ed., *Artificial intelligence and mathematical theory of computation: papers in honor of John McCarthy*, 359–380. Academic Press Professional.

Rintanen, J. 2004. Complexity of Planning with Partial Observability. In *Proc. 14th International Conference on Automated Planning and Scheduling (ICAPS 2004)*, 345–354.

To, S. T.; Son, T. C.; and Pontelli, E. 2015. A generic approach to planning in the presence of incomplete information: Theory and implementation. *Artificial Intelligence* 227: 1–51.