Knowledge Compilation for Action Languages

Sergej Scheck Alexandre Niveau Bruno Zanuttini

Normandie Univ.; UNICAEN, ENSICAEN, CNRS, GREYC, 14000 Caen, France

sergej.scheck,alexandre.niveau,bruno.zanuttini@unicaen.fr

Résumé

Nous étudions différents langages permettant de représenter des actions non déterministes pour la planification automatique, du point de vue de la compilation de connaissances. Précisément, nous considérons la question de la concision des langages (quelle est la taille de la description d'une action dans chaque langage?) et des questions de complexité (quelle est la complexité algorithmique de décider si un état est un successeur d'un autre pour une action décrite dans l'un de ces langages?). Nous étudions une version abstraite et nondéterministe de PDDL, le langage des théories d'actions en NNF, et DL-PPA, la logique dynamique des affectations propositionnelles parallèles. Nous montrons que ces langages ont une concision différente, et une complexité de requête différente : DL-PPA est le plus concis et NNF le moins concis, et décider si un état est successeur d'un autre est déjà NP-complet pour PDDL nondéterministe.

Mots Clef

Planification, compilation de connaissances, logique dynamique des affectations propositionnelles, planning domain definition language, théories d'actions

Abstract

We study different languages for representing nondeterministic actions in automated 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 (how hard is it to decide whether a state is a successor of another one through some action described in one of these languages?). We study an abstract, nondeterministic version of PDDL, the language of NNF action theories, and DL-PPA, the dynamic logic of parallel propositional assignments. We show that these languages have different succinctness and different complexity of queries: DL-PPA is the most succinct one and NNF is the least succinct, and deciding successorship is already NP-complete for nondeterministic PDDL.

Keywords

Planning, knowledge compilation, dynamic logic of propositional assignments, planning domain definition language, action theories

1 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 [15] is a standard language for this), and also for the planners to manipulate them while searching for a plan.

In this paper, we consider different representation languages from the point of view of knowledge compilation [9]. Knowledge compilation is 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?). Most work in knowledge compilation has been done on representations of Boolean functions, for instance, by Boolean formulas in negation normal form, by ordered binary decision diagrams, etc. [9].

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 (for instance, to find out which actions are applicable at the current node of the search tree), and typically start by transforming the action specifications into some representation suited for this. Hence having a clear picture of the properties of languages is clearly of interest for the development of such planners.

However, there have been a few papers studying aspects related to knowledge compilation for planning. For instance, Nebel has considered questions very similar to ours [17]. 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.[1] 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 demanding, but makes translations applicable in broader settings (for instance, to problems

[1] Actions are called "operators" there.

where we want to count or enumerate plans). Bäckström and Jonsson 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 [3]. This is also related to our work, but with a focus on languages for representing plans, while we study languages for representing actions.

We are interested here in (purely) nondeterministic actions, which lie at the core of fully observable nondeterministic planning and of conformant planning [19, 1, 12, 16, 20, 13]. We moreover consider propositional domains, in which states are assignments to a given set of propositions. The languages which we consider are of different natures: (grounded) $\mathbf{PDDL}$ is a specification language, $\mathbf{NNF}$ action theories are typically used as an internal representation by solvers, and $\mathbf{DL\text{-}PPA}$ is a logic allowing to specify programs and to reason about them. However, all of them can be viewed as languages for representing actions (as nondeterministic mappings from states to states), and their diversity (allowed constructs, representation of persisting values) allows us to give a clear picture. Our mid-term goal is to give a systematic picture of languages arising from all combinations of allowed constructs among the ones introduced in the literature (like nondeterministic choice, iteration, persistency by default, etc.).

The paper is structured as follows. In Section 2 we give the necessary background about actions and logic, and in Section 3 we formally define the action languages which we will consider. We then give our results: in Section 4 we prove positive results about polynomial-time translations between the languages, then in Section 5 we study the complexity of deciding whether a state is a possible successor of another state given an action description in one of these languages, and in Section 6 we give negative results about polynomial translations, which allows us to determine which languges are strictly more succinct than others. Finally, we conclude in Section 7.

2 Preliminaries

For any planning problem we consider a fixed finite set $P = \{p_1, \ldots, p_n\}$ of propositional 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. As an example, for $P = \{p_1, p_2, p_3\}$, $s = \{p_1, p_3\}$ denotes the state in which p_1 and p_3 are true and p_2 is false We write $\mathbb{P}$ for $\{p_i \mid i \in \mathbb{N}\}$.

Actions In this article we consider (purely) nondeterministic actions, which map states to sets of states. This means that a single state may have several successors through the same action, in contrast with deterministic actions (which map states to states), and that no relative likelihood is encoded between the successors of a state, in contrast with stochastic actions (which map states to probability distributions over states).

Definition 1 (action)**.** *Let P be a finite set of propositional variables. A* nondeterministic P-action *is a mapping a from 2^P to $2^{(2^P)}$. The elements of $a(s)$ are called* a-successors *of s.*

In the literature, actions are often considered together with preconditions which have to be satisfied to allow the execution of the action. However, for the results in this paper it is not important whether we require the action preconditions to be written explicitly, so for simplicity we assume them to be implicit. This means that an action a is applicable to a state s if and only if there exists at least one a-successor state s' of s.

Example 2. *Consider the following hunting example. Let*

$$P = \{\text{rabbit\_in\_sight}, \text{ rabbit\_alive}, \text{ loaded\_rifle}\}.$$

The action shoot_rabbit *can be described as "if* rabbit_alive *then: if* loaded_rifle *and* rabbit_in_sight, *then not* loaded_rifle *and either* rabbit_alive *and not* rabbit_in_sight *or not* rabbit_alive *and* rabbit_in_sight, *otherwise state unchanged". The action is applicable only if the rabbit is alive (otherwise it is not sensible to shoot at him). In this case, if the hunter is ready to shoot (the rifle is loaded and he can see the rabbit), then he tries to shoot 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 $s = \{\text{rabbit\_in\_sight}, \text{ rabbit\_alive}, \text{ loaded\_rifle}\}$ be the state where all three variables are true. Then $\text{shoot\_rabbit}(s)$ is the set of states ("successors") $\{s', s''\}$ with $s' = s \setminus \{\text{rabbit\_alive}, \text{loaded\_rifle}\} = \{\text{rabbit\_in\_sight}\}$ and $s'' = s \setminus \{\text{rabbit\_in\_sight}, \text{loaded\_rifle}\} = \{\text{rabbit\_alive}\}$.

In this article, we are interested in the properties of *representations* of actions in various langugages.

Definition 3 (action language)**.** *An* action language *is an ordered pair $\langle L, I\rangle$, where L is a set of* action descriptions *and I is an* interpretation function*. Action descriptions are ordered pairs $\langle \alpha, P\rangle$ where α is a formula and P is a finite subset of $\mathbb{P}$. The interpretation function I maps every action description $\langle \alpha, P\rangle \in L$ to a P-action $I(\alpha, P)$.*

Observe that P is *a priori* not related to the variables of α (this depends on the language). For instance, variables of P not mentioned in an $\mathbf{NPDDL}$ expression α are assumed to persist, and a formula may also use variables outside of P and even outside of $\mathbb{P}$ (called *auxiliary* variables), as in $\mathbf{NNFAT}$.
If the language $\langle L, I\rangle$ and the set P are clear from the context (or we just consider them to be fixed), then we write $\alpha(s)$ instead of $I(\alpha, P)(s)$ for the set of all α-successors of s.
In this article, we are mostly interested in translations between languages.

Definition 4 (translation)**.** *Let $\langle L_1, I_1\rangle$ and $\langle L_2, I_2\rangle$ be two action languages. A function $f : L_1 \to L_2$ is a* (proper) translation *if $I_1(\alpha, P) = I_2(f(\alpha, P), P)$ holds for all $\langle \alpha, P\rangle \in L_1$.*

In words, this means that the L_1-action description $\langle \alpha, P\rangle$ and the L_2-formula $f(\alpha, P)$ describe the same P-action. Again, when P is clear from the context, we write $f(\alpha)$ for $f(\alpha, P)$.
The function f is called a *polynomial-time* translation if it can be computed in time polynomial in the size of α and P. It is called a *polynomial-size* translation if the size of $f(\alpha, P)$ is bounded by a fixed polynomial in the size of α together with the size of P. Clearly, a polynomial-time translation is necessarily also a polynomial-size one, but a polynomial-size translation may not be polynomial-time.

Logic A Boolean formula φ is said to be in *negation normal form* ($\mathbf{NNF}$ for short) if it is built up from literals using conjunctions and disjunctions, *i.e.*, if it is generated by the grammar

$$\varphi ::= p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi$$

where p ranges over $\mathbb{P}$. We also use the shorthand notation $\top$ for $p \vee \neg p$ and $\bot$ for $p \wedge \neg p$, for an arbitrary $p \in \mathbb{P}$. For such a formula φ, $\mathrm{V}(\varphi)$ denotes the set of variables occuring in φ.
The set of formulas in $\mathbf{NNF}$ is complete, that is, every Boolean function can be described by an $\mathbf{NNF}$ formula. It is important to note that a formula φ with $\mathrm{V}(\varphi) \subseteq P$ for some set of variables P can be regarded as a formula over P (and the truth value of the corresponding Boolean function does not depend on the variables in $P \setminus \mathrm{V}(\varphi)$). For a boolean formula φ over a set of variables P and a state $s \subseteq P$, we write $s \models \varphi$ if φ evaluates to $\top$ under the assignment s.

Notation As a general rule, we use variables $a, b, \dots$ for actions, $\alpha, \beta, \dots$ for action expressions (in some language), and $\varphi, \psi, \dots$ for logical formulas. Since action descriptions are also formulas in some language, we reserve the term "expression" for action descriptions and the term "formula" for logical formulas occuring in them.

Representations In the whole article we assume the expressions and formulas of action languages to be "flat", *i.e.*, that the amount of memory space required to store the expression or formula is its number of symbols (without the parentheses). This is to be contrasted with representations of NNF formulas (in particular) in which isomorphic subformulas are assumed to be represented only once, with any superformula pointing to this shared representation, a representation widely used in the literature about knowledge compilation [9]. We however wish to highlight that all our results would go through if we assumed such "circuit" (or "DAG") representations for formulas (we leave the case of circuit representations of expressions for future work).

3 Action Languages

In this section we formally define the action languages which we study later.

Variants of PDDL The first language which we consider is the well-known planning domain description language ($\mathbf{PDDL}$). 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 [15, 10, 11]. Since we are interested in nondeterministic actions, we consider a nondeterministic variant of $\mathbf{PDDL}$ inspired by $\mathbf{NPDDL}$ [6], and so as to abstract away from the precise syntax of the specification language, we consider an idealized version. Finally, we consider a grounded version of $\mathbf{PDDL}$, namely, a propositional one. Still we use the name "$\mathbf{NPDDL}$", since we use essentially the same constructs.
We first define the syntax of $\mathbf{NPDDL}$.

Definition 5 ($\mathbf{NPDDL}$ action descriptions)**.** *An* $\mathbf{NPDDL}$ action description *is an ordered pair $\langle \alpha, P\rangle$, where α is an expression generated by the grammar*

$$\alpha ::= \varepsilon \mid p \mid \neg p \mid \alpha \,\&\, \alpha \mid \varphi \rhd \alpha \mid (\alpha \mid \alpha)$$

where p ranges over P and φ over Boolean formulas in $\mathbf{NNF}$ *over P.*

Intuitively,

- ε describes the action with no effect (the only successor of s is s itself),
- p (resp. $\neg p$) is the action which makes p true (resp. false),
- $\&$ denotes simultaneous execution (with no successor if the operands are inconsistent together),
- $\rhd$ denotes conditional execution,
- $\mid$ denotes nondeterministic choice,

and, importantly, variables not explicitly modified by the action are assumed to keep their value.
We insist that this syntax is an idealization of nondeterministic (grounded) $\mathbf{PDDL}$; for instance, the action which we write $x \rhd \big(y|(\neg y \,\&\, z)\big)$ would be written

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

with the syntax of $\mathbf{NPDDL}$ [6].
Action descriptions in $\mathbf{NPDDL}$ are interpreted as actions as follows.

Definition 6 (semantics of $\mathbf{NPDDL}$)**.** *The* interpretation function for $\mathbf{NPDDL}$ *is the function I defined by $I(\alpha, P)(s) = \{(s \cup e^+) \setminus e^- \mid \langle e^+, e^-\rangle \in M(\alpha, s)\}$, where $M(\alpha, s)$ is the set of possible modifications of s caused by α, defined inductively by*

- $M(\varepsilon, s) = \{\langle \emptyset, \emptyset\rangle\}$,

- $M(p, s) = \{\langle\{p\}, \emptyset\rangle\}$ *and* $M(\neg p, s) = \{\langle\emptyset, \{p\}\rangle\}$,
- $M(\varphi \rhd \alpha, s) = M(\alpha, s)$ *if* $s \models \varphi$, *else* $\{\langle\emptyset, \emptyset\rangle\}$,
- $M(\alpha_1 \,\&\, \alpha_2, s) = \{\langle e_1^+ \cup e_2^+, e_1^- \cup e_2^-\rangle \mid \langle e_1^+, e_1^-\rangle \in M(\alpha_1, s), \langle e_2^+, e_2^-\rangle \in M(\alpha_2, s), e_1^+ \cap e_2^- = e_1^- \cap e_2^+ = \emptyset\}$,
- $M(\alpha_1 \mid \alpha_2, s) = M(\alpha_1, s) \cup M(\alpha_2, s)$.

When we want to denote simultaneous execution of all action descriptions in a set A, we write $\&_{\alpha \in A}\, \alpha$. Also note that the action description $p \,\&\, \neg p$ (for an arbitrary $p \in \mathbb{P}$) defines a nonexecutable action. Hence it can be used as a subaction for encoding a precondition, and we use $\bot$ as shorthand notation for it.

Example 7 (continued)**.** *The following is an* $\mathbf{NPDDL}$ *action description for the action* shoot_rabbit *of Example 2.*

$$\begin{array}{ll} & (\neg\text{rabbit\_alive} \rhd \bot) \\ \& & \big((\text{rabbit\_alive} \wedge \text{rabbit\_in\_sight} \wedge \text{loaded\_rifle}) \\ & \quad \rhd(\neg\text{loaded\_rifle} \\ & \qquad \&(\neg\text{rabbit\_alive} \mid \neg\text{rabbit\_in\_sight}))\big) \end{array}$$

We are also interested in the language $\mathbf{NPDDL}$ as extended by the sequential execution operator ";".

Definition 8 ($\mathbf{NPDDL}_{\text{seq}}$)**.** *The language* $\mathbf{NPDDL}_{\text{seq}}$ *is the language in which action descriptions are generated by the following grammar:*

$$\alpha \;::=\; \varepsilon \mid p \mid \neg p \mid \alpha \,\&\, \alpha \mid \varphi \rhd \alpha \mid (\alpha \mid \alpha) \mid \alpha; \alpha$$

and the interpretation function is the same as that of $\mathbf{NPDDL}$ *for all constructs, augmented with*

$$(\alpha_1; \alpha_2)(s) = \{s'' \mid \exists s' \in \alpha_1(s) : s'' \in \alpha_2(s')\}$$

NNF action theories We now define the second language which we consider, namely that 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 and action descriptions [8, 7, 20]. We consider action theories represented in $\mathbf{NNF}$, which encompasses representations usually used like OBDDs or DNFs.

To prepare the definition we associate a variable $p' \in \mathbb{P}'$ to each variable $p \in \mathbb{P}$, where $\mathbb{P}'$ is a disjoint copy of $\mathbb{P}$; p' is intended to denote the value of p *after* the action took place, while p denotes the value *before*.

Definition 9 ($\mathbf{NNFAT}$)**.** *An* $\mathbf{NNFAT}$ *action description is an ordered pair* $\langle\alpha, P\rangle$ *where* α *is a Boolean formula in* $\mathbf{NNF}$ *over the set of variables* $P \cup \{p' \mid p \in P\}$*. The interpretation of* $\langle\alpha, P\rangle$ *is defined by*

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

In words, an (**NNF**) action theory 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 $P \cup \{p' \mid p \in P\}$.

Importantly, $\mathbf{NNFAT}$ does not assume the frame axiom, so that if, for example, a variable does not appear at all in an $\mathbf{NNFAT}$ action description, then this means that its value after the execution of the action can be arbitrary. For instance, the action description $\langle x' \vee (\neg y \vee z'), \{x, y, z\}\rangle$ represents an action which either (1) sets x to true and y, z to any value (nondeterministically), or (2) sets z to true and x, y to any value, in case y is true in the initial state, and otherwise sets each variable to any value, or (3) performs any consistent combination of (1) and (2).

Observe that a conjunct over variables in P in an $\mathbf{NNFAT}$ action description in fact encodes a precondition.

Example 10 (continued)**.** *The action* shoot_rabbit *of Example 2 can be written as (we use* $\rightarrow$ *and* $\leftrightarrow$ *for readability)*

$$\begin{array}{ll} & \text{rabbit\_alive} \\ \wedge & (\text{loaded\_rifle} \wedge \text{rabbit\_in\_sight}) \\ & \quad \rightarrow \Big(\big((\neg\text{rabbit\_in\_sight}' \wedge \text{rabbit\_alive}') \\ & \qquad \vee(\text{rabbit\_in\_sight}' \wedge \neg\text{rabbit\_alive}')\big) \\ & \qquad \wedge\big(\neg\text{loaded\_rifle}'\big)\Big) \\ \wedge & \big((\text{rabbit\_alive} \not\leftrightarrow \neg\text{rabbit\_alive}') \\ & \quad \vee(\text{rabbit\_in\_sight} \not\leftrightarrow \text{rabbit\_in\_sight}') \\ & \quad \vee(\text{loaded\_rifle} \not\leftrightarrow \text{loaded\_rifle}')\big) \\ & \rightarrow (\text{loaded\_rifle} \wedge \text{rabbit\_in\_sight}) \end{array}$$

As can be seen, encoding in $\mathbf{NNFAT}$ *the fact that the values of variables persist unless stated otherwise, typically requires subformulas (here the last conjunct) playing the same role as successor-state axioms in the situation calculus [18]. This typically requires a lot of space. We will give a formal meaning to this remark later in the paper (Proposition 30).*

Obviously, every action can be represented in this language, since the language of $\mathbf{NNF}$ formulas is complete for Boolean functions. We will see later that all the other languages that we study in this article are at least as succinct as $\mathbf{NNFAT}$; hence in particular, they are all complete as well.

DL-PPA The last language that we consider in this paper is the *dynamic logic of parallel propositional assignments* (**DL-PPA** for short), which has been introduced by Herzig *et al.* as an extension of the language **DL-PA** [14]. **DL-PA** was initially proposed for reasoning about imperative programs [5]. For instance, deciding whether there exists a plan from a given initial state to a goal characterized by a Boolean formula φ using actions $\alpha_1, \ldots, \alpha_k$ amounts to deciding whether the initial state satisfies the **DL-PA** formula $\langle(\alpha_1 \cup \ldots \cup \alpha_k)^*\rangle\varphi$. However, **DL-PPA** can also be used as an action language [14].

Definition 11 (**DL-PPA** action descriptions)**.** *A* **DL-PPA** action description *is an ordered pair* $\langle\alpha, P\rangle$

where α is an expression generated by the following grammar:

$$\begin{array}{lcl}\alpha & ::= & p \leftarrow \varphi \mid \varphi? \mid \alpha;\alpha \mid \alpha \cup \alpha \mid \alpha \sqcap \alpha \mid \alpha \sqcup \alpha \mid \ \alpha^* \\ \varphi & ::= & p \mid \top \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\alpha\rangle\varphi\end{array}$$

where p ranges over P.

In the literature, action descriptions are typically called $\mathbf{DL\text{-}PPA}$ *programs*, and formulas φ as in the definition are typically called $\mathbf{DL\text{-}PPA}$ *formulas*. Intuitively, the symbols mean the following:

- $p \leftarrow \varphi$ evaluates φ in the current state and assigns the resulting value to p,
- $\varphi?$ tests whether φ is satisfied in the current state and fails if it is not the case,
- ; denotes sequential execution,
- $\cup$ denotes (exclusive) nondeterministic choice, that is, execution of exactly one subaction,
- $\sqcap$ denotes parallel execution,
- $\sqcup$ denotes nonexclusive nondeterministic choice, that is, execution of one subaction or of both subactions,
- $*$ denotes looping an arbitrary number of times,
- $\langle\alpha\rangle\varphi$ denotes the modal construction "there is an execution of α ending in a state which satisfies φ".

We also use the shorthand notation $+p$ (resp. $-p$) for $p \leftarrow \top$ (resp. $p \leftarrow \bot$) and $[\alpha]\varphi$ for $\neg\langle\alpha\rangle\neg\varphi$ ("all executions of α end in a state which satisfies φ"). Moreover, as follows from the semantics which is defined below, $\top?$ denotes an empty action mapping any state to itself, and $(\varphi?;\alpha) \cup (\neg\varphi?;\beta)$ denotes the construction "if φ then α else β".
Also observe that every $\mathbf{NNF}$ formula can be rewritten into an equivalent $\mathbf{DL\text{-}PPA}$-formula (without $\wedge$) in linear time, since using De Morgan's laws we can always rewrite $\varphi \wedge \psi$ into the logically equivalent formula $\neg(\neg\varphi \vee \neg\psi)$.
Suppose that we are given the set P. To every formula there is an associated valuation which is the set of states that are models of the formula. The interpretations of programs are ternary relations on the set of states 2^P. We denote the valuation of a formula and the interpretation of a program by $\|\varphi\|$ and $\|\alpha\|$ respectively. $(s, s', w) \in \|\alpha\|$ means that there is an execution of α that leads from s to s' by assigning the variables in w. The formulas in $\mathbf{DL\text{-}PPA}$ are interpreted as follows, where $s, s', \hat{s}\ldots$ denote states and $w, w_1, \hat{w}_1 \ldots$ denote subsets of P:

Definition 12.

$$\begin{aligned}\|p\| &= \{s \mid p \in s\} \\ \|\top\| &= 2^P \\ \|\neg\varphi\| &= 2^P \setminus \|\varphi\| \\ \|\varphi_1 \vee \varphi_2\| &= \|\varphi_1\| \cup \|\varphi_2\| \\ \|\langle\alpha\rangle\varphi\| &= \{s \mid \exists w\ \exists s' : (s, s', w) \in \|\alpha\| \wedge s' \in \|\varphi\|\}\end{aligned}$$

The programs are interpreted in the following way:

$$\begin{aligned}\|p \leftarrow \varphi\| =&\{(s, s \cup \{p\}, \{p\}) \mid s \in \|\varphi\|\} \\ &\cup \{(s, s \setminus \{p\}, \{p\}) \mid s \notin \|\varphi\|\} \\ \|\varphi?\| =&\{(s, s, \emptyset) \mid s \in \|\varphi\|\} \\ \|\alpha_1; \alpha_2\| =&\{(s, s', w) \mid \exists\hat{w}_1\ \exists\hat{w}_2\ \exists\hat{s}' : (s, \hat{s}, \hat{w}_1) \in \|\alpha_1\| \\ &\wedge\ (\hat{s}, s', \hat{w}_2) \in \|\alpha_2\|\ \wedge\ w = \hat{w}_1 \cup \hat{w}_2\} \\ \|\alpha_1 \cup \alpha_2\| =&\|\alpha_1\| \cup \|\alpha_2\| \\ \|\alpha_1 \sqcap \alpha_2\| =&\{(s, s', w) \mid \exists s'_1\ \exists s'_2\ \exists w_1\ \exists w_2 : (s, s'_1, w_1) \\ &\in \|\alpha_1\|)\} \wedge (s, s'_2, w_2) \in \|\alpha_2\| \wedge w_1 \cap w_2 \cap s'_1 \\ &= w_1 \cap w_2 \cap s'_2 \wedge w = w_1 \cup w_2 \\ &\wedge s' = (s \setminus w) \cup (s'_1 \cap w_1) \cup (s'_2 \cap w_2) \\ \|\alpha_1 \sqcup \alpha_2\| =&\|\alpha_1 \cup \alpha_2 \cup (\alpha_1 \sqcap \alpha_2)\| \\ \|\alpha^*\| =& \bigcup_{k\in\mathbb{N}} \|\underbrace{\alpha; \alpha; \ldots; \alpha}_{k\ times}\|\end{aligned}$$

When applying $\mathbf{DL\text{-}PPA}$ *to planning tasks we identify valuations with states and describe actions as programs: an action α is described by a program α with $\alpha(s) = \{s' \mid \exists w : (s, w, s') \in \|\alpha\|\}$.*

Finally, we will be interested in the restriction of $\mathbf{DL\text{-}PPA}$ obtained when disallowing nonexclusive choice, the Kleene star, and modalities.

Definition 13 (restricted $\mathbf{DL\text{-}PPA}$)**.** *The language* restricted $\mathbf{DL\text{-}PPA}$ *is the language in which action descriptions $\langle\alpha, P\rangle$ are generated by the following grammar:*

$$\begin{array}{lcl}\alpha & ::= & p \leftarrow \varphi \mid \varphi? \mid \alpha;\alpha \mid \alpha \cup \alpha \mid \alpha \sqcap \alpha \\ \varphi & ::= & p \mid \top \mid \neg\varphi \mid \varphi \vee \varphi\end{array}$$

where p ranges over P, and whose semantics is the same as $\mathbf{DL\text{-}PPA}$ *restricted to this language.*

Example 14 (continued)**.** *The action* shoot_rabbit *of our running example 2 can be described as follows in (restricted)* $\mathbf{DL\text{-}PPA}$*:*

rabbit_alive?;
((rabbit_in_sight ∧ loaded_rifle)?;
  (−rabbit_alive ∪ −rabbit_in_sight);
  −loaded_rifle)
∪(¬rabbit_in_sight ∨ ¬loaded_rifle?)

Example 15. *The following* $\mathbf{DL\text{-}PPA}$ *program illustrates the meaning of the modal operators and of the Kleene star:*

(⟨shoot_rabbit; shoot_rabbit*⟩¬rabbit_alive?;
  shoot_rabbit)
∪(− loaded_rifle)

This action can be read as follows. If the hunter has a chance to kill the rabbit (which especially means that in the current state the rabbit is alive, as ensured by the first occurrence of shoot_rabbit*), then the hunter will shoot. Otherwise he will be disappointed and shoot in the air because he has no hope for success. But there could be several reasons for him being unable to kill the rabit: the rabbit is already dead, or the rifle is not loaded, or he does not see the rabbit...*

Note that **DL-PPA** has all the features of **NPDDL**, like the implicit frame axiom, and it additionally allows for modal operators. Hence, summarizing, we study languages with and without the sequence operator, with and without the implicit frame axiom, and with and without modalities. A mid-term goal of our work is to study combinations of such features in a systematic way, and we view this restricted set of languages as a meaningful set of representative languages to start with.

4 Polynomial-Time Translations

In this section, we exhibit translations between some of our languages of interest which can be carried out in polynomial time (hence, *a fortiori*, are polynomial-size). We remark that the identity function is an obvious polynomial-time translation from restricted **DL-PPA** into **DL-PPA**. We first show that any **NNFAT** action description α can be translated in polynomial time to an **NPDDL** action description $f(\alpha)$. The translation looks like a simple rewriting of α, but we have to care about (1) 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**, and (2) the fact that $\vee$ is inclusive-or in **NNFAT**, while nondeterministic choice in **NPDDL** is interpreted as one effect taking place (but not both). For (1) we will make explicit in the **NPDDL** translation that these variables can take any value, and for (2) it will turn out that in the translation of $\alpha_1 \vee \alpha_2$ into $f(\alpha_1) \,|\, f(\alpha_2)$, $f(\alpha_1)$ will encode all possible transitions of α_1, including those of $\alpha_1 \wedge \alpha_2$ (the "inclusive part" of the $\vee$), and similarly for $f(\alpha_2)$.

The translation f is defined inductively as follows for an **NNFAT** action description $\langle \alpha, P \rangle$:

1. if $\mathrm{V}(\alpha) \subseteq P$, then $f(\alpha) = \big(\neg\alpha \rhd \bot\big) \,\&\, \big(\alpha \rhd (\underset{p\in P}{\&}(p \,|\, \neg p))\big)$;
2. if $\mathrm{V}(\alpha) \not\subseteq P$ and $\mathrm{V}(\alpha_1) \subseteq P$, then $f(\alpha_1 \vee \alpha_2) = \big(\neg\alpha_1 \rhd f(\alpha_2)\big) \,\&\, \big(\alpha_1 \rhd (\underset{p\in P}{\&}(p \,|\, \neg p))\big)$; dually for $\mathrm{V}(\alpha_2) \subseteq P$;
3. if $\mathrm{V}(\alpha) \not\subseteq P$ and $\mathrm{V}(\alpha_1) \subseteq P$, then $f(\alpha_1 \wedge \alpha_2) = \big(\alpha_1 \rhd f(\alpha_2)\big) \,\&\, \big(\neg\alpha_1 \rhd \bot\big)$; dually for $\mathrm{V}(\alpha_2) \subseteq P$;
4. $f(p') = p \,\&\, \left(\underset{q\in P, q\neq p}{\&}(q \,|\, \neg q)\right)$;
5. $f(\neg p') = \neg p \,\&\, \left(\underset{q\in P, q\neq p}{\&}(q \,|\, \neg q)\right)$;
6. if $\mathrm{V}(\alpha_1), \mathrm{V}(\alpha_2) \not\subseteq P$, $f(\alpha_1 \wedge \alpha_2) = f(\alpha_1) \,\&\, f(\alpha_2)$;
7. if $\mathrm{V}(\alpha_1), \mathrm{V}(\alpha_2) \not\subseteq P$, $f(\alpha_1 \vee \alpha_2) = f(\alpha_1) \,|\, f(\alpha_2)$.

Observe for future reference that for all states s, all possible modifications $\langle e^+, e^- \rangle$ in $M(f(\alpha), s)$ are P-complete in the sense that $e^+ \cup e^- = P$ (all variables are mentioned in e^+ or e^-). This is easily seen by induction on the definition of f.

Proposition 16. *Let $\langle \alpha, P \rangle$ be an* **NNFAT** *action description. Then we have $s' \in \alpha(s) \iff s' \in f(\alpha)(s)$.*

PROOF. The translation is clearly polynomial-time, since the "gadgets" added to the rewriting in the first 5 cases involve no recursive call of f. We now show that it is correct, by induction on the structure of α.

1. First assume $\mathrm{V}(\alpha) \subseteq P$. Then by the semantics of **NNFAT**, $s' \in \alpha(s)$ holds if and only if s satisfies α, which is equivalent to s satisfying α and s' being arbitrary, which is equivalent to $s' \in f(\alpha)(s)$ by the definition of $f(\alpha)$ and the semantics of **NPDDL**.
2. Now assume $\alpha = \alpha_1 \vee \alpha_2$ and $\mathrm{V}(\alpha_1) \subseteq P$. For $s' \in \alpha_1(s)$, we have $s' \in \alpha(s)$, and since we have $\mathrm{V}(\alpha_1) \subseteq P$, we have that s satisfies α_1, and hence $s' \in f(\alpha)(s)$ is equivalent to $s' \in \big(\&_{p\in P}(p \,|\, \neg p)\big)(s)$, which is true for all s'; hence both $s' \in \alpha(s)$ and $s' \in f(\alpha)(s)$ hold. Now for $s' \notin \alpha_1(s)$, we have $s' \in \alpha(s)$ if and only if $s' \in \alpha_2(s)$, which is equivalent to $s' \in f(\alpha_2)(s)$ by the induction hypothesis, and this in turn is equivalent to $s' \in f(\alpha)(s)$ by the definition of $f(\alpha_1 \vee \alpha_2)$ and the semantics of **NPDDL**.
3. Now assume $\alpha = \alpha_1 \wedge \alpha_2$ and $\mathrm{V}(\alpha_1) \subseteq P$. We have that $s \in \alpha(s)$ is equivalent to $s' \in \alpha_1(s) \wedge s' \in \alpha_2(s)$, and since we have $\mathrm{V}(\alpha_1) \subseteq P$, this is equivalent to $s \models \alpha_1 \wedge s' \in \alpha_2(s)$, which in turn is equivalent to $s \models \alpha_1 \wedge s' \in f(\alpha_2)(s)$ by the induction hypothesis, which is finally equivalent to $s' \in f(\alpha)(s)$ by the definition of $f(\alpha_1 \wedge \alpha_2)$ and the semantics of **NPDDL**.
4. Now let $\alpha = p'$. Then $s' \in \alpha(s)$ is equivalent to $p \in s'$ with s' otherwise arbitrary, which is clearly equivalent to $s' \in f(\alpha)(s)$.
5. The proof for $\alpha = \neg p'$ is symmetric to the previous case.
6. Let $\alpha = \alpha_1 \wedge \alpha_2$. Then $s' \in \alpha(s)$ is equivalent to $s' \in \alpha_1(s) \wedge s' \in \alpha_2(s)$, and by the induction hypothesis this is equivalent to $s' \in f(\alpha_1)(s) \wedge s' \in f(\alpha_2)(s)$. Now since the possible modifications of $f(\alpha_1)$ and $f(\alpha_2)$ are P-complete, it is easily seen from the definition of the semantics of **NPDDL** that the

set of possible modifications $M(f(\alpha_1) \,\&\, f(\alpha_2), s)$ is exactly $M(f(\alpha_1), s) \cap M(f(\alpha_2), s)$, so that $s' \in f(\alpha_1)(s) \wedge s' \in f(\alpha_2)(s)$ is equivalent to $s' \in \big(f(\alpha_1) \,\&\, f(\alpha_2)\big)(s)$, that is, to $s' \in f(\alpha)(s)$.

7. Finally let $\alpha = \alpha_1 \vee \alpha_2$. Assume first $s' \in \alpha(s)$, and by symmetry $s' \in \alpha_1(s)$; then by the induction hypothesis we have $s' \in f(\alpha_1)(s)$ and hence, $s' \in \big(f(\alpha_1) \mid f(\alpha_2)\big)(s) = f(\alpha)(s)$. Conversely, assume $s' \in f(\alpha)(s)$, then by the definition of $f(\alpha)$ and the semantics of $\mid$ we have $s' \in f(\alpha_1)(s)$ or $s' \in f(\alpha_2)(s)$. Assume by symmetry $s' \in f(\alpha_1)(s)$. Then by the induction hypothesis we have $s' \in \alpha_1(s)$ and hence, $s' \in (\alpha_1 \vee \alpha_2)(s)$, that is, $s' \in \alpha(s)$.

□

The following propositions are quite intuitive, because $\mathbf{NPDDL}_{\text{seq}}$ and restricted $\mathbf{DL\text{-}PPA}$ are essentially the same: $\varphi \rhd \ldots$ is analogous to $\varphi?$, $\mid$ is analogous to $\cup$, and $\&$ is analogous to $\sqcap$. However, we must pay attention to two facts. The first difference between the languages is that in $\mathbf{NPDDL}_{\text{seq}}$, if φ is not true in $\varphi \rhd \alpha$, then the action just does not change the current state whereas in $\mathbf{DL\text{-}PPA}$, φ being false results in a failure. The other difference is that formulas in $\mathbf{NPDDL}_{\text{seq}}$ must be in $\mathbf{NNF}$, while $\mathbf{DL\text{-}PPA}$ does not have restrictions on the occurence of $\neg$ but does not have the $\wedge$ connective.

Proposition 17. *There is a polynomial-time translation of* $\mathbf{NPDDL}_{\text{seq}}$ *into restricted* $\mathbf{DL\text{-}PPA}$.

PROOF. Consider an $\mathbf{NPDDL}_{\text{seq}}$ action description α. The translation f first replaces each subformula of the form $\varphi \wedge \psi$ in α with $\neg(\neg\varphi \vee \neg\psi)$, then it computes an action description in restricted $\mathbf{DL\text{-}PPA}$ as follows:

$$\begin{array}{lcl}
f(\epsilon) & = & \top? \\
f(p) & = & +p \\
f(\neg p) & = & -p \\
f(\alpha_1 \,\&\, \alpha_2) & = & f(\alpha_1) \sqcap f(\alpha_2) \\
f(\varphi \rhd \alpha) & = & \big(\varphi?; f(\alpha)\big) \cup (\neg\varphi?) \\
f(\alpha_1 \mid \alpha_2) & = & f(\alpha_1) \cup f(\alpha_2) \\
f(\alpha_1; \alpha_2) & = & f(\alpha_1); f(\alpha_2)
\end{array}$$

It is easy to check that this translation can be computed in polynomial time and that it is correct. In particular, φ is duplicated in the fifth line but it involves no recursive call of f, hence preserving polynomial size (the rewriting of $\neg\varphi$ into a $\mathbf{DL\text{-}PPA}$ formula can be done in linear time), and $\neg\varphi?$ in the same line ensures that the action does nothing but does not fail when φ is not satisfied. □

The proof of the converse is completely symmetric.

Proposition 18. *There is a polynomial-time translation of restricted* $\mathbf{DL\text{-}PPA}$ *to* $\mathbf{NPDDL}_{\text{seq}}$.

PROOF. Consider a restricted $\mathbf{DL\text{-}PPA}$ action description $\langle \alpha, P \rangle$. The translation f first replaces each subformula of the form $\neg(\varphi \vee \psi)$ with $(\neg\varphi \wedge \neg\psi)$, ending up with a description in which all formulas are in $\mathbf{NNF}$, then it computes an action description in $\mathbf{NPDDL}_{\text{seq}}$ as follows:

$$\begin{array}{lcl}
f(p \leftarrow \varphi) & = & (\varphi \rhd p) \,\&(\neg\varphi \rhd \neg p) \\
f(\varphi?) & = & (\varphi \rhd \varepsilon) \,|(\neg\varphi \rhd \bot) \\
f(\alpha_1; \alpha_2) & = & f(\alpha_1); f(\alpha_2) \\
f(\alpha_1 \cup \alpha_2) & = & f(\alpha_1) \mid f(\alpha_2) \\
f(\alpha_1 \sqcap \alpha_2) & = & f(\alpha_1) \,\&\, f(\alpha_2)
\end{array}$$

It is easy to check that this translation can be computed in polynomial time and that it is correct. In particular, φ is duplicated in the first and second lines but it involves no recursive call of f, hence preserving polynomial size, and $\neg\varphi \rhd \bot$ in the second line ensures that the action fails when φ is not satisfied. □

5 Complexity of Deciding Successorship

We now turn to studying the complexity of *queries* to action descriptions. In this paper, we concentrate on the most natural query, which is formally defined by the following computational problem.

Definition 19 (IS-SUCC)**.** *Let L be an action language. The decision problem* IS-SUCC *is defined by:*

- *input: an action description $\langle \alpha, P \rangle \in L$ and two states $s, s' \subseteq P$,*
- *question: is s' an α-successor of s?*

Proposition 20. *The problem* IS-SUCC *is polynomial-time solvable for $L =$* $\mathbf{NNFAT}$.

PROOF. From the semantics of $\mathbf{NNFAT}$ it follows that deciding $s' \in \alpha(s)$ amounts to deciding whether the assignment to $P \cup \{p' \mid p \in P\}$ induced by s, s' satisfies α, which can clearly be done in linear time. □

Proposition 21. *The problem* IS-SUCC *is in* $\mathbf{NP}$ *for $L =$* $\mathbf{NPDDL}_{\text{seq}}$.

PROOF. We define a witness for a positive instance to be composed of either α_1 or α_2 for each subexpression $\alpha_1 \mid \alpha_2$ of α, and of a state t for each subexpression $\alpha_1; \alpha_2$ (representing the guessed intermediate state of the execution). Such a witness is clearly of polynomial size. Now verifying it amounts to verifying that when the nondeterministic choices are those encoded by the witness and the execution of sequence constructs go through the encoded intermediate states, s' is indeed an α-successor of s. This can clearly be done in polynomial time since there remains only to evaluate conditions of $\rhd$ constructs in given states and applying effects of the form p or $\neg p$. □

For showing hardness, we build a specific action able to "produce" all and only satisfiable 3-CNF formulas. For this we first define an encoding of any 3-CNF formula φ over n variables as an assignment to a polynomial number of variables.

Notation 22. *Let $n \in \mathbb{N}$ and X_n be the set of variables $\{x_1, \dots, x_n\}$. Observe that there are a cubic number N_n of clauses of length 3 over X_n (any choice of 3 variables with a polarity for each). We fix an arbitrary enumeration $\gamma_1, \gamma_2, \dots, \gamma_{N_n}$ of all these clauses, and we define Q_n to be the set of variables $\{q_1, q_2, \dots, q_{N_n}\}$. Then we identify an assignment s to Q_n to the 3-CNF formula over X_n, written $\varphi(s)$, which for all i contains the clause γ_i if and only if $q_i \in s$ holds.*
We also write $s(\varphi)$ for the assignment to Q_n which encodes a 3-CNF formula φ over X_n. By $\ell \in \gamma_i$ we mean that the literal ℓ occurs in the clause γ_i.

Example 23. *Let $n = 2$, and consider an enumeration of all clauses over variables $X_2 = \{x_1, x_2\}$ which starts with $\gamma_1 = (x_1 \vee x_1 \vee x_2), \gamma_2 = (x_1 \vee x_1 \vee \neg x_2), \gamma_3 = (x_1 \vee \neg x_1 \vee x_2), \gamma_4 = (x_1 \vee \neg x_1 \vee \neg x_2), \gamma_5 = (\neg x_1 \vee \neg x_1 \vee x_2), \gamma_6 = (\neg x_1 \vee \neg x_1 \vee \neg x_2), \dots$. Then the 3-CNF $\varphi = (x_1 \vee x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_1 \vee x_2)$ is identified to the state $s(\varphi) = \{q_1, q_5\}$.*

Using Notation 22, for all $n \in \mathbb{N}$ we define the $\mathbf{NPDDL}$ action description $\langle \beta_n, Q_n \rangle$ by

$$\beta_n = \mathop{\&}_{x \in X_n} \left(\big(\mathop{\&}_{\gamma_i : x \in \gamma_i} (q_i \,|\, \varepsilon) \big) \;\Big|\; \big(\mathop{\&}_{\gamma_i : \neg x \in \gamma_i} (q_i \,|\, \varepsilon) \big) \right)$$

Intuitively, β_n chooses an assignment ($\bot$ or $\top$) to each variable in X_n (outermost nondeterministic choices). Whenever it chooses one, it chooses nondeterministically some clauses which are satisfied by it, and adds them to the result. Hence it builds a satisfiable formula (which is satisfied precisely by—at least—the assignment made of its choices over each variable).

Lemma 24. *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 a β_n-successor of the state $\emptyset$.*

PROOF. If φ is satisfiable, let s_X be an assignment to X_n which satisfies it. Consider the execution of β_n in which for each x, when the subexpression corresponding to x is executed, the left (resp. right) subexpression of the non-deterministic choice is executed if $x \in s_X$ holds (resp. if $x \notin s_X$ holds). Finally, consider the execution of this expression $\&(q_i \mid \varepsilon)$ in which for all i, q_i (resp. ε) is executed when φ contains (resp. does not contain) the clause γ_i. Clearly, this execution reaches $s(\varphi)$. Conversely, if an execution reaches a state s, then a model of $\varphi(s)$ can be built by considering each variable $x \in X_n$, and including (resp. not including) x if and only if the execution went through the left (resp. right) of the nondeterministic choice when the subexpression corresponding to x was executed. $\square$

Since β_n can clearly be built in polynomial time given a set of variables X_n, Lemma 24 directly gives a reduction from the 3-SAT problem to the problem IS-SUCC for $\mathbf{NPDDL}$. Hence the latter problem is $\mathbf{NP}$-hard, and since we have shown IS-SUCC to be in $\mathbf{NP}$ for $\mathbf{NPDDL}_{\mathrm{seq}}$ (Proposition 21), we have the following.

Proposition 25. *The problem* IS-SUCC *is $\mathbf{NP}$-complete for $L = \mathbf{NPDDL}$ and for $L = \mathbf{NPDDL}_{\mathrm{seq}}$.*

Finally, since $\mathbf{NPDDL}_{\mathrm{seq}}$ and restricted $\mathbf{DL\text{-}PPA}$ are translatable into each other in polynomial time (Propositions 17 and 18), we have the following.

Corollary 26. *The problem* IS-SUCC *is $\mathbf{NP}$-complete when L is restricted $\mathbf{DL\text{-}PPA}$.*

We finally turn to the complexity of IS-SUCC for $\mathbf{DL\text{-}PPA}$.

Proposition 27. *The problem* IS-SUCC *is $\mathbf{PSPACE}$-complete for $L = \mathbf{DL\text{-}PPA}$.*

PROOF. It is known that model checking for $\mathbf{DL\text{-}PPA}$ is $\mathbf{PSPACE}$-complete [4]. This problem is the one of checking whether a given state s is in $\|\varphi\|$ for a given $\mathbf{DL\text{-}PPA}$ formula φ. We reduce it to IS-SUCC for $\mathbf{DL\text{-}PPA}$ as follows.
Suppose that we are given a $\mathbf{DL\text{-}PPA}$ formula φ over the set $P = \{p_1, \dots, p_n\}$, and let without loss of generality $s = \{p_1, \dots, p_k\}$. Let r (standing for "result") be a fresh variable, and build the $\mathbf{DL\text{-}PPA}$ action description $\langle \alpha, P \cup \{r\} \rangle$ with

$$\begin{aligned} \alpha \;=\; & (\langle +p_1; \dots; +p_k; -p_{k+1}; \dots; -p_n \rangle \varphi?; +r) \\ & \cup (\neg \langle +p_1; \dots; +p_k; -p_{k+1}; \dots; -p_n \rangle \varphi?; -r) \end{aligned}$$

Clearly, α can be built in time polynomial in the size of φ, and α does nothing except setting r to $\top$ if $s \in \|\varphi\|$ holds, and to $\bot$ otherwise. It follows that s is a model of φ if and only if the state $\{r\}$ is an α-successor of the state $\emptyset$.[2]
Hence IS-SUCC is $\mathbf{PSPACE}$-hard for $L = \mathbf{DL\text{-}PPA}$.
For membership, we reduce it to the satisfiability problem for $\mathbf{DL\text{-}PPA}$ formulas, which is in $\mathbf{PSPACE}$ [4]. Given an action description $\langle \alpha, P \rangle$ with $P = \{p_1, \dots, p_n\}$, a state $s = \{p_1, \dots, p_k\}$ (without loss of generality) and a state s', we define φ to be the $\mathbf{DL\text{-}PPA}$ formula

$$\langle +p_1; \dots; +p_k; -p_{k+1}; \dots; -p_n; \alpha \rangle \Big(\bigwedge_{p \in s'} p \wedge \bigwedge_{p \in P \setminus s'} \neg p \Big)$$

Clearly, φ can be built in polynomial time, and it is satisfiable if and only if the program "go to state s and then execute α" can lead to the state s', which is just a rephrasing of s' being an α-successor of s. $\square$

[2] The choice of $\emptyset$ is arbitrary, since the variables of φ are all set by the modalities and are used only there and hence, their initial and final values do not matter.

6 Succinctness

In this section, we study the relative succinctness of action languages. Succinctness is formally defined as follows [9].

Definition 28 (succinctness)**.** *An action language L_1 is said to be* at least as succinct as *an action language L_2, denoted by $L_1 \preceq L_2$, if there exists a polynomial-size translation from L_2 to L_1. If $L_1 \preceq L_2$ and $L_2 \not\preceq L_1$ hold, then L_1 is said to be* strictly more succinct than L_2, *written $L_1 \prec L_2$. If $L_1 \preceq L_2$ and $L_2 \preceq L_1$ hold, then L_1 and L_2 are said to be* equally succinct.

The succinctness relation $\preceq$ is reflexive and transitive, hence it is a preorder. However, it is not antisymmetric and thus not an order.
Clearly, if there is a polynomial-time translation from L_2 to L_1 then $L_1 \preceq L_2$ holds. Hence we have the following as a direct consequence of Propositions 17 and 18.

Proposition 29. *The languages* $\mathbf{NPDDL}_{\mathrm{seq}}$ *and restricted* $\mathbf{DL}$-$\mathbf{PPA}$ *are equally succinct.*

Our next results rely on assumptions about *nonuniform* complexity classes. Recall that $\mathbf{P/poly}$ (resp. $\mathbf{NP/poly}$) is the class of all decision problems such that for all $n \in \mathbb{N}$, there is a polynomial-time algorithm (resp. a nondeterministic polynomial-time algorithm) which decides the problem for all inputs of size n [2]. The assumptions $\mathbf{NP} \not\subseteq \mathbf{P/poly}$ and $\mathbf{PSPACE} \not\subseteq \mathbf{NP/poly}$ which we use are standard ones; in particular, $\mathbf{NP} \subseteq \mathbf{P/poly}$ would imply a collapse of the polynomial hierarchy at the second level (Karp-Lipton theorem), and $\mathbf{PSPACE} \subseteq \mathbf{NP/poly}$ would imply a collapse at the third level, since already $\mathbf{coNP} \subseteq \mathbf{NP/poly}$ would do so [21].

Proposition 30. *There is no polynomial-size translation from* $\mathbf{NPDDL}$ *into* $\mathbf{NNFAT}$ *unless* $\mathbf{NP} \subseteq \mathbf{P/poly}$ *holds.*

PROOF. We use the action description β_n that was introduced in Section 5; the size of β_n is clearly polynomial in n.
Assume that for every $\mathbf{NPDDL}$ action description α_n there is an equivalent $\mathbf{NNFAT}$ action description α'_n of size polynomial in that of α_n. In particular, there is an $\mathbf{NNFAT}$ action description β'_n of size polynomial in n which is equivalent to β_n. Then the following is a nonuniform polynomial-time algorithm for the 3-SAT problem; given a formula φ in 3-CNF over n variables:

1. encode φ into a state $s(\varphi)$ over the set of variables Q_n as in Notation 22;
2. decide whether $s(\varphi)$ is a β'_n-successor of $\emptyset$;
3. claim that φ is satisfiable if the answer is positive, otherwise claim that φ is unsatisfiable.

All steps are polynomial-time (Proposition 20), the algorithm is correct (Lemma 24), and the algorithm depends only on the number of variables in φ (which is polynomially related to the size of φ), hence this is indeed a nonuniform polynomial time algorithm for 3-SAT. Since 3-SAT is $\mathbf{NP}$-complete, we get $\mathbf{NP} \subseteq \mathbf{P/poly}$. $\square$

We finally consider the relative succinctness of $\mathbf{DL}$-$\mathbf{PPA}$ and $\mathbf{NPDDL}_{\mathrm{seq}}$. Since model checking in $\mathbf{DL}$-$\mathbf{PPA}$ is $\mathbf{PSPACE}$-complete, there can be no polynomial time translation from $\mathbf{DL}$-$\mathbf{PPA}$ to $\mathbf{NPDDL}_{\mathrm{seq}}$ unless $\mathbf{PSPACE} = \mathbf{NP}$. However, we will prove a stronger result.
For this, we use the problem of deciding whether a QBF formula is valid, for QBFs restricted to be of the form $\Phi = \forall x_1 \exists x_2 \dots \forall x_{2n-1} \exists x_{2n} \varphi$, with φ a 3-CNF formula and $\mathrm{V}(\varphi) \subseteq X_{2n} = \{x_1, \dots, x_{2n}\}$; clearly, deciding validity is as hard for such formulas (hereafter called "normalized QBFs") as for unrestricted QBFs, and hence it is $\mathbf{PSPACE}$-complete.
For all $n \in \mathbb{N}$, we define the $\mathbf{DL}$-$\mathbf{PPA}$ action description $\langle \delta_{2n}, X_{2n} \cup Q_{2n} \cup \{r\} \rangle$, where Q_{2n} is as in Notation 22, r is a fresh variable (standing for "result"), and δ_{2n} is defined to be

$$\begin{array}{rcl} r & \leftarrow & \big([+x_1 \cup -x_1]\langle +x_2 \cup -x_2 \rangle \\ & & \dots \\ & & [+x_{2n-1} \cup -x_{2n-1}]\langle +x_{2n} \cup -x_{2n} \rangle \psi_{2n}\big) \end{array}$$

with $\psi_{2n} = \bigwedge_{q_i \in Q_{2n}} \big(q_i \rightarrow (\bigvee_{x \in \gamma_i} x \vee \bigvee_{\neg x \in \gamma_i} \neg x)\big)$ (rewritten without $\wedge$ nor $\rightarrow$ in polynomial time). Observe that the size of δ_{2n} is polynomial in n.

Lemma 31. *Let Φ be a normalized QBF over the set of variables $X_{2n} = \{x_1, \dots, x_{2n}\}$. Then Φ is valid if and only if $s(\varphi) \cup \{r\}$ is a δ_{2n}-successor of $s(\varphi)$.*

PROOF. By the semantics of $\mathbf{DL}$-$\mathbf{PPA}$, the modality $[+x_i \cup -x_i]$ mimicks exactly the quantification $\forall x_i$, and $\langle +x_i \cup -x_i \rangle$ mimicks exactly $\exists x_i$. On the other hand, it is easy to see that an assignment s_X to X_{2n} is a model of φ if and only if $s_X \cup s(\varphi)$ is a model of ψ_{2n}. It follows that Φ is valid if and only if $[+x_1 \cup -x_1]\langle +x_2 \cup -x_2 \rangle \dots [+x_{2n-1} \cup -x_{2n-1}]\langle +x_{2n} \cup -x_{2n} \rangle \psi_{2n}$ is true in $s(\varphi)$ and hence, that Φ is valid if and only if δ_{2n} assigns r to $\top$ when run in $s(\varphi)$, which finishes the proof. $\square$

Using δ_{2n}, the proof of the following result is parallel to that of Proposition 30.

Proposition 32. *There is no polynomial-size translation from* $\mathbf{DL}$-$\mathbf{PPA}$ *into* $\mathbf{NPDDL}_{\mathrm{seq}}$ *unless* $\mathbf{PSPACE} \subseteq \mathbf{NP/poly}$ *holds.*

PROOF. Assume that there is a polynomial-size translation from $\mathbf{DL}$-$\mathbf{PPA}$ into $\mathbf{NPDDL}_{\mathrm{seq}}$, and for all n, let δ'_{2n} be an $\mathbf{NPDDL}_{\mathrm{seq}}$ description equivalent to δ_{2n} and of size polynomial in that of δ_{2n}, hence in n. Then the following is a nonuniform nondeterministic polynomial-time

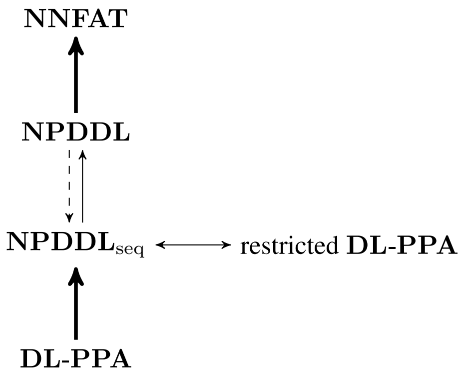

Figure 7.1: Succinctness relations between the languages. A thick arrow from L to L' means $L \prec L'$, a thin line means $L \preceq L'$, and a dashed arrow means that it is still unknown whether $L \preceq L'$.

algorithm for the problem of deciding the validity of a normalized QBF; given a normalized QBF over $2n$ variables, with matrix φ:

1. encode φ into $s(\varphi)$,
2. decide whether $s(\varphi) \cup \{r\}$ is a δ'_{2n}-successor of $s(\varphi)$,
3. claim that Φ is valid if the answer is positive, otherwise claim that Φ is not valid.

The steps are all feasible in deterministic or nondeterministic polynomial time (Proposition 21), the algorithm is correct by Lemma 31, and δ_{2n} depends only on the number of variables of Φ, hence this is indeed a nonuniform nondeterministic polynomial-time algorithm for deciding the validity of a normalized QBF. Since this is a $\mathbf{PSPACE}$-complete problem, we conclude $\mathbf{PSPACE} \subseteq \mathbf{NP/poly}$. $\square$

7 Conclusion

We have studied the complexity of deciding whether a state is a successor of another one through a given action, and the relative succinctness of three languages which are suitable for specifying planning tasks and actions. We have shown that deciding successorship is polynomial-time solvable for $\mathbf{NNFAT}$, $\mathbf{NP}$-complete for $\mathbf{NPDDL}$, $\mathbf{NPDDL}_{\text{seq}}$, and restricted $\mathbf{DL\text{-}PPA}$, and $\mathbf{PSPACE}$-complete for $\mathbf{DL\text{-}PPA}$. The succinctness results agree with the intuition that the languages which are more succinct also have harder queries; the relationships which we have shown are represented on Figure 7.1.

An examination of the proof of Proposition 32 reveals that the reasons for $\mathbf{DL\text{-}PPA}$ being strictly more succinct than $\mathbf{NPDDL}_{\text{seq}}$ are the modal operators. Our mid-term goal is to investigate complexity of queries and succinctness in a more systematic way, for languages constructed using combinations of features like the sequence operator, modalities, Kleene star, parallel execution, etc. For example, we want to try to find out whether $\mathbf{DL\text{-}PPA}$ without the Kleene star is strictly less succinct than $\mathbf{DL\text{-}PPA}$ (because until now the only elimination of $*$ that we know requires exponential space). Another interesting question is whether $\mathbf{NPDDL}_{\text{seq}}$ is strictly more succinct than $\mathbf{NPDDL}$, because IS-SUCC is $\mathbf{NP}$-complete for both of them.

Acknowledgements

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

References

[1] Alexandre Albore, Héctor Palacios, and Hector Geffner. Compiling uncertainty away in nondeterministic conformant planning. In *Proc. 19th European Conference on Artificial Intelligence (ECAI 2010)*, volume 215, pages 465–470, 2010.

[2] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach.* Cambridge University Press, 2009.

[3] Christer Bäckström and Peter Jonsson. Algorithms and limits for compact plan representations. *Journal of Artificial Intelligence Research*, 44:141–177, 2012.

[4] Philippe Balbiani, Andreas Herzig, François Schwarzentruber, and Nicolas Troquard. DL-PA and DCL-PC: model checking and satisfiability problem are indeed in PSPACE. *arXiv preprint arXiv:1411.7825*, 2014.

[5] Philippe Balbiani, Andreas Herzig, and Nicolas Troquard. 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)*, pages 143–152, 2013.

[6] Piergiorgio Bertoli, Alessandro Cimatti, Ugo Dal Lago, and Marco Pistore. Extending PDDL to nondeterminism, limited sensing and iterative conditional plans. In *Proc. ICAPS 2003 Workshop on PDDL*, 2003.

[7] Daniel Bryce, Subbarao Kambhampati, and David E. Smith. Planning graph heuristics for belief space search. *Journal of Artificial Intelligence Research*, 26:35–99, 2006.

[8] Alessandro Cimatti and Marco Roveri. Conformant planning via symbolic model checking. *Journal of Artificial Intelligence Research*, 13:305–338, 2000.

[9] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[10] Maria Fox and Derek Long. The third international planning competition: Temporal and metric planning. In *Proc. 6th International Conference on Artificial Intelligence Planning Systems (AIPS 2002)*, pages 333–335, 2002.

[11] Maria Fox and Derek Long. PDDL2. 1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[12] Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers, 2013.

[13] Tomas Geffner and Hector Geffner. Compact policies for fully observable non-deterministic planning as SAT. In *Proc. 28th International Conference on Automated Planning and Scheduling (ICAPS 2018)*, pages 88–96, 2018.

[14] Andreas Herzig, Frédéric Maris, and Julien Vianey. Dynamic logic of parallel propositional assignments and its applications to planning. In *Proc. 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, pages 5576–5582, 2019.

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

[16] Christian J. Muise, Sheila A. McIlraith, and Vaishak Belle. Non-deterministic planning with conditional effects. In *Proc. 24th International Conference on Automated Planning and Scheduling (ICAPS 2014)*, pages 370—374, 2014.

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

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

[19] Jussi Rintanen. Complexity of planning with partial observability. In *Proc. 14th International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pages 345–354, 2004.

[20] Son Thanh To, Tran Cao Son, and Enrico Pontelli. A generic approach to planning in the presence of incomplete information: Theory and implementation. *Artificial Intelligence*, 227:1–51, 2015.

[21] Chee K Yap. Some consequences of non-uniform conditions on uniform classes. *Theoretical computer science*, 26(3):287–300, 1983.