Knowledge compilation for nondeterministic action languages

Sergej Scheck

Normandie Université

THÈSE

Pour obtenir le diplôme de doctorat

Spécialité INFORMATIQUE

Préparée au sein de l'Université de Caen Normandie

Knowledge Compilation for Nondeterministic Action Languages

Présentée et soutenue par
SERGEJ SCHECK

Thèse soutenue le 12/12/2022 devant le jury composé de		
M. STEFAN MENGEL	Chargé de recherche HDR, CNRS	Rapporteur du jury
M. BERNHARD NEBEL	Professeur des universités, UNIVERSITE FREIBURG ALLEMAGNE	Rapporteur du jury
MME HÉLÈNE FARGIER	Directeur de recherche au CNRS, Institut de Recherche en Informatique de Toulouse	Membre du jury
M. PETER JONSSON	Professeur des universités, Université de Linköping	Membre du jury
M. ALEXANDRE NIVEAU	Maître de conférences, Université de Caen Normandie	Membre du jury
M. ANDREAS HERZIG	Directeur de recherche au CNRS, Institut de Recherche en Informatique de Toulouse	Président du jury

Thèse dirigée par BRUNO ZANUTTINI (Groupe de recherche en informatique, image, automatique et instrumentation)

UNIVERSITÉ CAEN NORMANDIE

GREYC
Laboratoire de recherche en sciences du numérique

Contents

Résumé

Introduction

La *compilation de connaissances* (KC – knowledge compilation) est une branche de l'intelligence artificielle qui s'occupe du pré-traitement des représentations d'information pour pallier la difficulté calculatoire.

L'idée clé de la KC est que les données en entrée des requêtes peuvent souvent être partitionnées en deux parties : une grande partie qui est fixe pour la plupart des requêtes, appelée « base de données », et une petite partie variable appelée directement « requête ». En répondant à une requête, la base de données doit souvent être traitée de la même manière, on doit donc répéter le même calcul encore et encore. Pour éviter cela, on peut pré-traiter la base de données (en sachant à quelles requêtes on veut répondre le plus souvent ensuite) en faisant une grande partie de ce calcul une seule fois à l'avance. Plus précisément, on calcule d'abord (on appelle cela la phase « hors ligne ») une représentation plus convenable, pour répondre aux requêtes par la suite (dans la phase « en ligne »). Un exemple de pré-traitement d'information dans la vie réelle est le dictionnaire : pour chercher les mots plus efficacement, on trie d'abord les mots lexicographiquement. La différence avec la compilation de connaissances est que le tri de mots peut être effectué en temps polynomial, pour permettre la recherche des mots en temps logarithmique. Dans la compilation de connaissances, on est plutôt intéressé par un gain plus significatif : en investissant un calcul en temps éventuellement exponentiel au départ, on peut atteindre une réponse aux requêtes en temps polynomial par la suite, tandis que, avant le pré-traitement, les requêtes étaient NP-difficiles. Bien sûr, on veut dans le même temps éviter que la taille de la nouvelle représentation ne devienne trop grande : généralement on veut qu'elle reste polynomiale dans la taille de la représentation originale. On dit que la représentation reste succincte.

Typiquement, on est intéressé par deux types de tâches de calcul : les requêtes, où le résultat est une information extraite de la base de données, et les transformations, où le résultat est une autre représentation de la base de données. Ce n'est pas toujours possible de trouver une représentation qui soit succincte et permette de répondre efficacement à toutes requêtes, et d'effectuer toutes les transformations, efficacement : généralement il faut trouver un compromis entre la concision et l'efficacité. La KC étudie précisément l'interconnexion entre la taille de la représentation et la complexité de répondre aux requêtes ou d'effectuer les transformations. On appelle les ensembles de représentations avec une structure concrète les *langages* (de représentation) ; typiquement, ils sont définis par une grammaire. Le but final est d'élaborer un tableau (appelé *knowledge compilation map* – carte de compilation de connaissances) qui résume les résultats de complexité pour permettre de choisir le langage le plus convenable pour une tâche concrète.

Dans ce travail, nous appliquons l'approche de compilation de connaissances à la planification automatique, concrètement aux *langages d'actions*, i.e. les ensembles de formules qui permettent d'exprimer les actions. Un langage d'actions est un ensemble de formules/expressions, appelées *descriptions d'action*, avec une fonction d'interprétation qui associe les descriptions d'action à des actions concrètes.

La planification automatique est un champ de l'intelligence artificielle qui étudie les algorithmes pour atteindre un objectif à partir d'un état initial en exécutant des actions. En appliquant une action, on transforme un état du monde en un autre état. Un plan peut être une séquence d'actions ou une politique pour choisir les actions en fonction des observations. Nous nous concentrons sur les actions non déterministes, i.e. où le résultat de l'application de l'action dans un état n'est pas toujours certain. Enfin, nous travaillons avec des états propositionnels, encodés par des variables booléennes ; une action non déterministe est une fonction qui associe à chaque état un ensemble d'états. Notons qu'il est commun dans la littérature sur la planification de considérer les représentations booléennes.

Une notion centrale de la compilation de connaissances est celle de traduction : une traduction d'un langage L_1 à un autre langage L_2 est une fonction qui transforme une expression de L_1 en une expression de L_2 en satisfaisant une notion de préservation. La notion de préservation est définie par une propriété à laquelle on s'intéresse : par exemple, on peut vouloir traduire une description Π_1 d'une instance d'un problème en une autre description Π_2, de sorte que chaque plan π_2 pour Π_2 corresponde à un plan π_1 pour Π_1, et que pour chaque π_1 il y ait un π_2 qui soit de taille polynomiale en la taille de π_1 (en particulier, cela signifie qu'il existe un plan pour Π_2 si et seulement s'il existe un plan pour Π_1).

Travaux connexes

Il existe déjà des applications de la compilation de connaissances à la planification automatique, mais elles considèrent d'autres notions de compilation qui sont moins strictes, par exemple, la compilation peut être appliquée à un formalisme de planification entier (avec l'ensemble de variables, l'état initial et le but). Ainsi, une traduction permet d'introduire des nouvelles variables, et c'est l'existence d'un plan qui est préservée. Contrairement à ces travaux précédents, nous étudions une notion de traduction très stricte, appliquée aux descriptions d'actions (qui elles-mêmes peuvent être identifiées avec des fonctions booléennes). Nous interdisons que la traduction introduise de nouvelles variables ; de ce point de vue notre travail est similaire à la carte de compilation classique, et c'est l'interprétation de la description d'action qui est préservée par la traduction.

La différence avec les autres études de compilation de connaissances est la manière dont nous obtenons les langages (nos langages sont inspirés par les langages d'actions existants, et la plupart d'eux ne correspondent pas directement à un langage booléen), la richesse des langages que nous étudions (on va montrer que la plupart de ces langages peuvent être considérés comme des ensembles qui contiennent tous les langages de la carte de compilation classique), et les critères que nous utilisons pour construire notre carte de compilation. Nous nous concentrons sur les requêtes les plus typiques dans la planification : d'abord, tester si une action est applicable dans un état, et si un plan séquentiel atteint nécessairement un but. Les autres requêtes sont moins communes dans la littérature, mais elles semblent assez naturelles pour la planification non déterministe. Pour les transformations, nous considérons l'extraction d'une pré-condition en forme normale négative, l'expression du choix non déterministe, de l'exécution séquentielle et de la négation des actions. Finalement, nous comparons les langages du point de vue de leur *succinctness*, c'est-à-dire de leur concision.

Une caractéristique principale de notre approche est, d'abord, la distinction des langages de la façon dont ils décrivent les actions (les langages peuvent être « impératifs », ce qui signifie que les modifications sont explicitement mentionnées dans la description, ou « déclaratifs », ce qui signifie qu'ils décrivent les contraintes satisfaites par les états prédécesseurs et successeurs, et que les changements des valeurs de variables sont toujours possibles s'ils ne contredisent pas ces contraintes). L'autre caractéristique est la représentation des descriptions d'actions (et des formules logiques qu'elles contiennent), où nous distinguons deux possibilités : une représentation directe par un *arbre*, où la taille de la formule correspond au nombre de symboles dans l'expression, et une représentation par un graphe orienté acyclique, qui peut être plus efficace en stockant des sous-expressions isomorphes une seule fois.

Préliminaires

Les applications principales de nos résultats sont au problème générique de la planification non déterministe et totalement observable, que nous présentons dans le troisième chapitre, avec les notions de plan et de politique. Nos travaux utilisent également, pour les formules logiques, la forme normale négative (NNF), i.e. la représentation des formules logiques dans laquelle les négations ne peuvent apparaître que dans les littéraux, et les seuls connecteurs internes sont $\wedge$ (conjonction) et $\vee$ (disjonction) ; la forme normale conjonctive (CNF) et la forme normale disjonctive (DNF) sont des cas particulier de NNF. Par ailleurs, nous définissons les représentations des expressions en arbre et en circuit (graphe orienté acyclique, qui est la représentation standard dans la littérature de KC).

Les autres notions utiles sont des notions de complexité : nous définissons les classes de complexité que nous utilisons, et plus spécifiquement les classes non uniformes $\mathsf{P/poly}$ et $\mathsf{NP/poly}$ (c'est-à-dire, où l'algorithme qui résout le problème dépend de la taille de l'entrée). Nous présentons les hypothèses sur lesquelles nos résultats reposent, à savoir $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ et $\mathsf{coNP} \not\subseteq \mathsf{NP/poly}$. Nous montrons enfin deux lemmes qui sont utilisés par la suite : d'une part, décider si une formule CNF avec 3 littéraux par clause (appelée 3-CNF) et avec au maximum 3 occurrences de chaque variable et un problème NP-complet, et d'autre part, décider la validité d'une formule totalement quantifiée avec seulement un type de quantificateur (mais des négations pouvant porter sur des sous-formules quantifiées) est un problème PSPACE-complet.

Cadre formel

Nous donnons ensuite les définitions des objets centraux de notre travail. Nous définissons tout d'abord de manière formelle la notion de langage d'actions et de traduction ; une traduction d'un langage L_1 en un langage L_2 est une fonction qui associe à une description d'action dans L_1 et à l'ensemble des variables sur lesquelles elle porte, une description d'action dans L_2, qui a la même interprétation sur le même ensemble de variables. Par ailleurs, nous généralisons le concept de représentation en arbre ou circuit aux langages d'actions, et écrivons $L^{\mathbf{T}}$ et $L^{\mathbf{C}}$ pour les versions arbre et circuit, respectivement, du langage L,.

Les requêtes et transformations auxquelles nous nous intéressons, sont, tout d'abord, les problèmes de décider pour une action si elle est applicable dans un état (on note cette requête IS-APPLIC), et si une séquence d'actions résulte toujours en un état qui satisfait une certaine propriété, décrite par une formule NNF (on note cette requête ENTAILS). Ce sont les requêtes les plus basiques, qui apparaissent souvent dans la littérature (elles constituent ensemble la problématique du *belief tracking*). Nous donnons aussi une réduction simple du complément de IS-APPLIC (c'est-à-dire de la « non applicabilité ») à ENTAILS, montrant ainsi que décider la non applicabilité n'est pas plus difficile que ENTAILS au sens de la complexité. Les requêtes moins utilisées, mais qui ont toutefois certaines applications à la planification, sont IS-SUCC (décider si un état peut être atteint par une action appliquée dans un autre état), ST (« self-transition », cas particulier de IS-SUCC : décider si un état est un successeur possible de lui-même par une action), IS-MON (décider si une action est monotone dans un état, i.e. qu'elle ne modifie jamais la valeur d'une variable de vraie à fausse, ou qu'elle ne modifie jamais une valeur de fausse à vraie), et IS-DET (décider si une action a au plus un successeur dans un état).

Les transformations que nous étudions sont EXTRACT-PRECOND (pour une action, trouver une formule telle que l'action soit applicable dans un état si et seulement si l'état satisfait cette formule), SEQUENCE (pour deux actions, trouver une description d'action qui soit équivalente à leur exécution en séquence), CHOICE (exprimer l'action définie comme le choix non déterministe de deux autres actions), et NEGATION (exprimer le complément d'une action). Contrairement aux requêtes, pour lesquelles nous nous intéressons à la complexité en temps de calcul, pour les transformations nous cherchons à savoir si le résultat de la transformation (qui doit être dans le même langage que l'entrée) est de taille au maximum polynomiale. Nous définissons aussi le troisième critère pour la comparaison des langages : la *succinctness*, c'est-à-dire la concision d'un langage : un langage L_1 est appelé plus succinct qu'un autre langage L_2 si L_2 peut être traduit en L_1 et que la taille de la traduction est bornée par un polynôme (on dit que la

traduction est *de taille polynomiale*), mais que l'inverse n'est pas vrai.

Langages d'action minimalement complets

Nous commençons notre étude par les langages **O-PDDL**, **E-PDDL** et **NNFAT**, qui sont motivés par la littérature et les algorithmes de planification. Le *Planning Domain Definition Language* **PDDL** est un langage très utilisé par la communauté de planification, introduit originellement en 1998 pour la compétition internationale de planification, et dont nous étudions deux versions nondéterministes.

Les deux versions de **PDDL** non déterministe que nous introduisons diffèrent par la sémantique de l'application simultanée d'effets, qui donne lieu à deux langages d'actions : **O-PDDL** (« O » signifiant « original ») et **E-PDDL** (« E » signifiant « égalitaire »). Enfin, les théories d'actions, qui sont des formules logiques utilisées pour décrire les actions, nous donnent un langage déclaratif noté **NNFAT** (pour *NNF action theories*). Nous définissons les trois langages **O-PDDL**, **E-PDDL** et **NNFAT** par leur grammaire.

Nous appelons ces langages des « langages minimalement complets », car nous montrons que si l'on exclut un connecteur de la grammaire, le langage résultant devient incomplet, i.e. incapable d'exprimer *toutes* les actions non déterministes (sinon, le langage est dit complet). La différence entre **O-PDDL** et **E-PDDL** est la sémantique de l'exécution parallèle (& en **O-PDDL** et $\sqcap$ en **E-PDDL**). Pour définir la sémantique, nous définissons les effets des actions, une notion similaire aux *add-list* et *del-list* de **STRIPS**.

La différence entre & et $\sqcap$ est le traitement des effets qui sont en conflit : & transforme ces effets en un nouvel effet en appliquant le principe « affecter une variable à vrai l'emporte sur l'affectation à faux », tandis que pour $\sqcap$, l'exécution parallèle d'effets qui sont en conflit n'est pas définie.

Nous illustrons cette différence avec des exemples, qui montrent aussi la différence entre les langages impératifs (**O-PDDL** et **E-PDDL**) et le langage déclaratif **NNFAT** : pour décrire la persistance des valeurs des variables non modifiées par une action, on a besoin de nombreuses sous-expressions en **NNFAT**, tandis qu'en **O-PDDL** et **E-PDDL**, cette persistance est intégrée dans la sémantique.

Résultats pour les langages minimalement complets

Nous donnons une image complète au regard de tous les critères : la complexité des requêtes, les résultats de concision et les résultats pour les transformations.

Nous démontrons tout d'abord qu'il existe une traduction de **NNFAT** en **E-PDDL** (mais pas en **O-PDDL**) en temps polynomial. Ceci implique que les résultats de difficulté qui sont valides pour **NNFAT** sont valides également pour **E-PDDL**, et réciproquement, que les résultats d'appartenance (d'une requête à une classe de complexité) valides pour **E-PDDL** le sont aussi pour **NNFAT**.

Requêtes Nous montrons que la plupart des requêtes (sauf IS-SUCC et donc ST) sont déjà difficiles pour **NNFAT** : ENTAILS est coNP-complet pour tous les langages minimalement complets, et IS-SUCC est NP-complet pour **E-PDDL** et **O-PDDL**. La preuve de ces résultats utilise des descriptions d'actions notées $\alpha_n^{\text{sat},\sqcap}$ et $\alpha_n^{\text{sat},\&}$ (qui décrivent la même action), qui jouent un rôle fondamental dans notre travail et sont réutilisées plusieurs fois dans la thèse. L'intuition est que ces actions construisent de manière non déterministe toutes les formules 3-CNF satisfaisables.

Nous démontrons ensuite que IS-APPLIC est NP-complet pour **NNFAT** et **E-PDDL**, mais peut être résolu en temps polynomial pour **O-PDDL**. Pour ce dernier résultat, nous décrivons les procédures d'instanciation d'un circuit par un état, et de réduction d'un circuit : pour l'instanciation, il s'agit d'ajuster le circuit qui décrit une action à un état concret, ce qu'on utilise en prétraitement ; la réduction est un processus de simplification du circuit après élimination de branches non exécutables (ou dont on veut ignorer l'exécution). Ces processus d'instanciation et de réduction sont réutilisés plusieurs fois dans la thèse, pour élaborer des algorithmes efficaces.

Concernant les autres résultats, pour ENTAILS, nous obtenons que cette requête est coNP-complète pour les trois langages. En utilisant l'instanciation et la réduction, nous démontrons aussi que IS-MON

et IS-DET sont résolubles en temps polynomial pour **O-PDDL**. Nous montrons qu'ils sont en revanche coNP-complets pour **E-PDDL** et **NNFAT**. Enfin, nous démontrons que la requête ST est NP-complète pour **O-PDDL**, et résoluble en temps linéaire pour **E-PDDL** et **NNFAT**.

Concision En utilisant ces résultats, et parce que les preuves de difficulté de ST pour **O-PDDL** et de IS-SUCC pour **E-PDDL** utilisent des descriptions d'actions qui permettent de construire toutes les 3-CNF satisfaisables, nous montrons que sous l'hypothèse $\mathsf{NP} \not\subseteq \mathsf{P/poly}$, il n'existe pas de traduction de taille polynomiale de **O-PDDL** en **E-PDDL** ou **NNFAT**, ni de traduction dans le sens inverse. Il s'ensuit que **O-PDDL** est incomparable, du point de vue de la concision, avec **E-PDDL** et **NNFAT**; nous montrons en effet que **NNFAT** est moins succinct que **E-PDDL** sous l'hypothèse $\mathsf{NP} \not\subseteq \mathsf{P/poly}$.

Transformations Pour les transformations, nous remarquons que CHOICE est évidemment possible en temps linéaire pour tous les langages, parce qu'ils incluent un connecteur pour exprimer le choix. Nous montrons que ni SEQUENCE ni NEGATION ne peuvent être effectuées dans **O-PDDL** et **E-PDDL** sans explosion superpolynomiale, et que SEQUENCE ne peut pas l'être sans explosion dans **NNFAT**; en revanche, NEGATION peut être effectuée en temps linéaire dans **NNFAT**. Enfin, pour EXTRACT-PRECOND nous montrons que le résultat ne peut pas toujours être de taille polynomiale pour **NNFAT** et **E-PDDL**, mais que la transformation peut être effectuée en temps linéaire pour **O-PDDL**. Tous les résultats d'impossibilité tiennent sous sous l'hypothèse que $\mathsf{NP} \not\subseteq \mathsf{P/poly}$.

Conclusion sur les langages minimalement complets Il se trouve tout d'abord que tous les résultats de ce chapitre sont les mêmes pour les arbres et les circuits.

Par ailleurs, on peut conclure de nos résultats que la spécification d'actions directement en **NNFAT** (qui est souvent utilisée pour le raisonnement interne par les planificateurs) pourrait être plus raisonnable que la conversion depuis **E-PDDL**. On observe aussi que **O-PDDL** et **E-PDDL** ont des propriétés assez différentes du point de vue de la complexité, et qu'ils sont incomparables du point de vue de la concision. Cela signifie que le choix du langage concret à utiliser doit dépendre de l'algorithme de planification et de la notion de solution : par exemple, si on cherche un plan acyclique, on pourra être intéressé par un langage où ST est traitable efficacement (donc **E-PDDL**); sinon, il est en général souhaitable de pouvoir tester l'applicabilité en temps polynomial (et on choisira alors **O-PDDL**). Une autre observation intéressante est que **E-PDDL** permet d'effectuer une transformation qui ressemble au conditionnement des formules booleénnes, généralisé pour les descriptions d'actions. En revanche, **O-PDDL** ne permet pas d'effectuer cette transformation sans explosion superpolynomiale en taille; l'étude du conditionnement pour tous les langages pourra faire l'objet de travaux futurs.

Restrictions

Restrictions à la STRIPS Le langage **STRIPS** peut être considéré une restriction de **PDDL** (plus précisément de sa version propositionnelle), et s'il est enrichi par un connecteur pour le choix non déterministe d'effets, on obtient deux langages, que nous notons **O-NCSTRIPS** et **E-NCSTRIPS**. On peut également voir ces langages comme obtenus en imposant des restrictions sur la profondeur et la structure des descriptions d'actions en **O-PDDL** et **E-PDDL**.

Nous étudions ces deux langages **O-NCSTRIPS** et **E-NCSTRIPS**; précisément, ces langages sont définis comme l'ensemble des descriptions d'actions de **O-PDDL** et **E-PDDL**, respectivement, qui ne permettent que des exécutions conditionnelles, parallèles, de choix non déterministes d'ensembles d'affectations élémentaires. Cette restriction de structure est de même nature que celle de la forme normale conjonctive pour les formules booléennes (une formule de profondeur bornée et avec un ordre fixe de connecteurs). Pour ces restrictions, il n'y a pas lieu de faire la distinction entre la représentation en arbre et la représentation en circuit, et nous montrons que les résultats de complexité de requêtes sont dans tous les cas les mêmes pour le langage « parent » et son sous-langage. Pour les transformations, les résultats se transfèrent également, sauf pour CHOICE pour lequel nous laissons le problème ouvert.

Un résultat intéressant est que le langage **NNFAT$^{\mathbf{C}}$** (version circuit de **NNFAT**) ne peut pas être traduit en **E-NCSTRIPS** sans explosion en taille. C'est le seul résultat de concision de la thèse qui n'est pas basé sur une hypothèse de complexité non démontrée. Les autres résultats de concision sont basés sur l'hypothèse $\mathsf{NP} \not\subseteq \mathsf{P/poly}$, et les preuves sont assez analogues aux preuves pour les langages parents : nous montrons que **O-NCSTRIPS** est incomparable, du point de vue de la concision, avec **NNFAT**, **E-NCSTRIPS** et **E-PDDL**, et que **E-NCSTRIPS** est incomparable avec **O-PDDL**. Nous n'avons pas de preuve que les langages parents sont (strictement) plus succincts que leurs restrictions, mais nous conjecturons que c'est le cas.

Restriction aux affectations positives Enfin, nous considérons aussi une troisième restriction des langages minimalement complets, obtenue en interdisant une sorte d'affectation des variables. Le langage **NPDDL$_{\mathbf{nf}}$** est ainsi défini comme la restriction de **E-PDDL** qui ne permet que les affectations positives aux variables. Par conséquent, ce langage est à la fois un sous-langage de **E-PDDL** et un sous-langage de **O-PDDL**, puisque la différence entre ces deux langages est le traitement de l'interaction des affectations positives et négatives par l'exécution simultanée. Même s'il s'agit d'un langage très restrictif (il est incomplet, c'est-à-dire incapable d'exprimer toutes les actions non déterministes), le langage a tout de même une complexité importante pour IS-SUCC, qui est NP-complet (car l'expression $\alpha_n^{\mathrm{sat},\sqcap}$ utilisée pour **O-PDDL** et **E-PDDL** est déjà une expression de **NPDDL$_{\mathbf{nf}}$**). Nous montrons que les requêtes et transformations sont toujours possibles en temps polynomial pour **NPDDL$_{\mathbf{nf}}$** lorsque c'est le cas pour un de ses langages parents, et difficiles sinon. Un résultat intéressant est que **E-PDDL** est plus succinct que **NPDDL$_{\mathbf{nf}}$** même pour les actions positivement monotones (c'est-à-dire les actions que peut représenter **NPDDL$_{\mathbf{nf}}$**).

Pour terminer, notons que nous ne considérons pas de restrictions de **NNFAT**, parce qu'elles sont bien étudiées dans la littérature de compilations des connaissances.

Extensions

O-PDDL et E-PDDL Notre étude du langage **E-PDDL** est notamment motivé par la sémantique de la logique des affectations propositionnelles parallèles **DL-PPA** de la littérature. Pour cette raison, nous considérons dans ce chapitre des extensions de **E-PDDL** et de **O-PDDL** avec un connecteur de séquence ; emprunté à **DL-PPA**, notées **E-PDDL$_{\mathrm{seq}}$** et **O-PDDL$_{\mathrm{seq}}$**, respectivement. De manière analogue, nous proposons d'autres extensions, obtenues en ajoutant le connecteur de négation $\neg_{\mathrm{min}}$ (langages notés **O-PDDL$_{\mathrm{not}}$** et **E-PDDL$_{\mathrm{not}}$**) et le connecteur de conjonction $\wedge$ (**O-PDDL$_{\mathrm{and}}$** et **E-PDDL$_{\mathrm{and}}$**).

Pour définir la sémantique pour ;, $\wedge$ et $\neg_{\mathrm{min}}$, il faut décider comment définir les effets. Pour ;, nous adaptons la définition de **DL-PPA**, et pour $\neg_{\mathrm{min}}$ et $\wedge$ nous fixons une sémantique ; d'autres sémantiques naturelles pourraient toutefois être étudiées dans des travaux futurs.

Les questions que nous étudions sont les mêmes que pour les langages précédents : complexité des requêtes et des transformations, concision. Toutefois, nous obtenons moins de résultats que pour les autres langages en ce qui concerne les transformations et la concision. La raison en est que pour prouver des résultats de séparation (donc pour montrer qu'il n'existe pas une traduction de taille polynomiale), nous utilisons le fait qu'une requête dans un langage est plus difficile que dans un autre. Mais pour les extensions de langages, nous démontrons que presque toutes les requêtes deviennent PSPACE-complètes. Pour les transformations, nous utilisons typiquement le même principe de preuve, donc nos résultats sont seulement fragmentaires.

Nous démontrons que pour l'extension par $\wedge$ et par ;, les langages avec la représentation en arbre permettent encore de vérifier la réponse aux requêtes en temps polynomial (et donc, les requêtes restent dans NP ou coNP). Pour les représentations en circuit et pour les autres extensions, presque toutes les requêtes (sauf ST pour les deux représentations de **E-PDDL$_{\mathrm{not}}$** et **E-PDDL$_{\mathrm{and}}$**) sont PSPACE-complètes. L'idée principale des preuves est typiquement basée sur l'utilisation d'une modification de la formule $\alpha_n^{\mathrm{sat},\sqcap}$ en une description d'action définie de manière récursive.

Dans des travaux futurs, il serait également intéressant d'étudier un « surlangage » de tous nos langages. Nous avons déjà une preuve que toutes les requêtes dans ce langage sont PSPACE-complètes pour la représentation en circuit. Nous conjecturons qu'il serait le plus succinct (strictement) de tous, mais si tel est le cas, ce sera probablement difficilement à démontrer avec les outils que nous utilisons dans cette thèse.

NNFAT Dans le dernier chapitre technique, nous étudions deux extensions de **NNFAT**. Nous l'enrichissons avec deux connecteurs permettant d'exprimer la persistance de variables. Le premier connecteur, F_X, vise à exprimer la propriété sémantique des langages impératifs : quand il est appliqué, une variable dans l'ensemble X peut changer sa valeur si et seulement si le changement apparaît explicitement dans la description d'action. Par conséquent, nous avons aussi besoin d'une définition d'effets, avec une distinction entre effets explicites et effets implicites. Nous qualifions F_X de *syntaxique*, parce que la sémantique d'une expression $\mathsf{F}_X(\alpha)$ dans le langage $\mathbf{NNFAT}_\mathsf{F}$ obtenu en autorisant F_X partout dans la formule, dépend toujours de la description d'action α, et pas seulement de l'action décrite par α.

L'autre connecteur, $\mathsf{C}_{X,V,F}$, utilise la notion de raisonnement par circonscription : parmi les successeurs obtenus via le même changement des valeurs des variables d'un ensemble F, il choisit ceux où le changement de valeurs des variables dans l'ensemble X est minimal. Ce connecteur est *sémantique*, parce que l'interprétation de $\mathsf{C}_{X,V,F}(\alpha)$ dépend seulement de l'action décrite par α, mais pas de la manière dont cette action est décrite. Le langage **NNFAT** enrichi par $\mathsf{C}_{X,V,F}$ est noté $\mathbf{NNFAT}_\mathsf{C}$.

Dans ce chapitre, nous étudions seulement la représentation en circuit, mais nous conjecturons que nos résultats sont également valides pour la représentation en arbre. Le résultat le plus important est probablement une traduction polynomiale de $\mathbf{NNFAT}_\mathsf{F}$ en **NNFAT**. Avec ce résultat, nous obtenons aisément tous les résultats de complexité pour $\mathbf{NNFAT}_\mathsf{F}$: IS-SUCC et ST sont possibles en temps polynomial, IS-DET et IS-MON sont coNP-complets, et IS-APPLIC est NP-complet. Pour $\mathbf{NNFAT}_\mathsf{C}$, nous montrons que toutes les requêtes sauf ST sont PSPACE-complètes. Il est intéressant de noter que ST reste soluble en temps polynomial même pour un langage si riche.

Conclusion

Nous concluons notre travail par un résumé des résultats, donné par tableaux de complexité et des diagrammes de concision (tableaux 1, 2, 3 ; diagrammes 1 et 2).

Parmi les perspectives ouvertes par nos travaux, figurent l'étude de nouvelles transformations et requêtes (qui peuvent aussi être utiles pour la planification), l'étude de nouveaux langages, et des applications de nos résultats, par exemple aux langages de représentation de préférences.

Language	IS-SUCC	IS-APPLIC	ENTAILS
Langages minimalement complets			
NNFAT$^{\mathbf{T}}$, **NNFAT$^{\mathbf{C}}$**	temps linéaire	NP-complet	coNP-complet
O-PDDL$^{\mathbf{T}}$, **O-PDDL$^{\mathbf{C}}$**	NP-complet	temps linéaire	coNP-complet
E-PDDL$^{\mathbf{T}}$, **E-PDDL$^{\mathbf{C}}$**	NP-complet	NP-complet	coNP-complet
Restrictions des langages impératifs			
O-NCSTRIPS	NP-complet	temps linéaire	coNP-complet
E-NCSTRIPS	NP-complet	NP-complet	coNP-complet
NPDDL$_{\mathbf{nf}}$	NP-complet	temps linéaire	coNP-complet
Extensions des langages impératifs			
O-PDDL$^{\mathbf{T}}_{\mathrm{seq}}$	NP-complet	NP-complet	coNP-complet
O-PDDL$^{\mathbf{C}}_{\mathrm{seq}}$	PSPACE-complet	PSPACE-complet	PSPACE-complet
O-PDDL$^{\mathbf{T}}_{\mathrm{and}}$	NP-complet	NP-complet	coNP-complet
O-PDDL$^{\mathbf{C}}_{\mathrm{and}}$	PSPACE-complet	PSPACE-complet	PSPACE-complet
O-PDDL$^{\mathbf{T}}_{\mathrm{not}}$, **O-PDDL$^{\mathbf{C}}_{\mathrm{not}}$**	PSPACE-complet	PSPACE-complet	PSPACE-complet
E-PDDL$^{\mathbf{T}}_{\mathrm{seq}}$	NP-complet	NP-complet	coNP-complet
E-PDDL$^{\mathbf{C}}_{\mathrm{seq}}$	PSPACE-complet	PSPACE-complet	PSPACE-complet
E-PDDL$^{\mathbf{T}}_{\mathrm{and}}$	NP-complet	NP-complet	coNP-complet
E-PDDL$^{\mathbf{C}}_{\mathrm{and}}$	PSPACE-complet	PSPACE-complet	PSPACE-complet
E-PDDL$^{\mathbf{T}}_{\mathrm{not}}$, **E-PDDL$^{\mathbf{C}}_{\mathrm{not}}$**	PSPACE-complet	PSPACE-complet	PSPACE-complet
Extensions de **NNFAT**			
NNFAT$^{\mathbf{C}}_{\mathsf{F}}$	temps polynomial	NP-complet	coNP-complet
NNFAT$^{\mathbf{C}}_{\mathsf{C}}$	PSPACE-complet	PSPACE-complet	PSPACE-complet

TABLE 1: Résultats de complexité pour IS-SUCC, IS-APPLIC et ENTAILS.

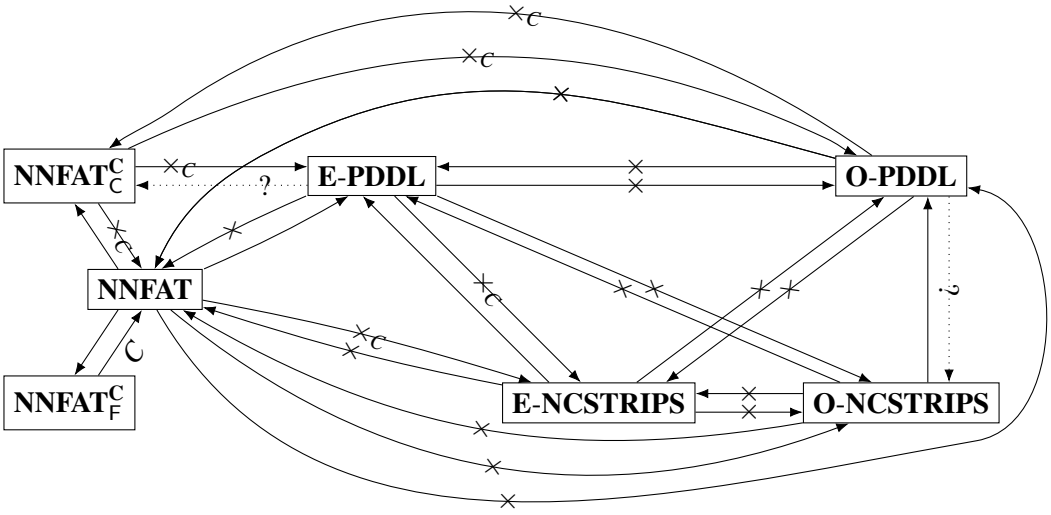
Language	ST	IS-DET	IS-MON
Langages minimalement complets			
NNFAT$^{\mathbf{T}}$, **NNFAT$^{\mathbf{C}}$**	temps linéaire	coNP-complet	coNP-complet
O-PDDL$^{\mathbf{T}}$, **O-PDDL$^{\mathbf{C}}$**	NP-complet	temps polynomial	temps polynomial
E-PDDL$^{\mathbf{T}}$, **E-PDDL$^{\mathbf{C}}$**	temps linéaire	coNP-complet	coNP-complet
Restrictions des langages impératifs			
O-NCSTRIPS	NP-complet	temps polynomial	temps polynomial
E-NCSTRIPS	temps linéaire	NP-complet	coNP-complet
NPDDL$_{\mathbf{nf}}$	temps linéaire	temps polynomial	trivial/temps polynomial
Extensions des langages impératifs			
O-PDDL$^{\mathbf{T}}_{\mathrm{seq}}$	NP-complet	coNP-complet	coNP-complet
O-PDDL$^{\mathbf{C}}_{\mathrm{seq}}$	PSPACE-complet	PSPACE-complet	PSPACE-complet
O-PDDL$^{\mathbf{T}}_{\mathrm{and}}$	NP-complet	coNP-complet	coNP-complet
O-PDDL$^{\mathbf{C}}_{\mathrm{and}}$	PSPACE-complet	PSPACE-complet	PSPACE-complet
O-PDDL$^{\mathbf{T}}_{\mathrm{not}}$, **O-PDDL$^{\mathbf{C}}_{\mathrm{not}}$**	PSPACE-complet	PSPACE-complet	PSPACE-complet
E-PDDL$^{\mathbf{T}}_{\mathrm{seq}}$	NP-complet	coNP-complet	coNP-complet
E-PDDL$^{\mathbf{C}}_{\mathrm{seq}}$	PSPACE-complet	PSPACE-complet	PSPACE-complet
E-PDDL$^{\mathbf{T}}_{\mathrm{and}}$	temps linéaire	coNP-complet	coNP-complet
E-PDDL$^{\mathbf{C}}_{\mathrm{and}}$	temps linéaire	PSPACE-complet	PSPACE-complet
E-PDDL$^{\mathbf{T}}_{\mathrm{not}}$, **E-PDDL$^{\mathbf{C}}_{\mathrm{not}}$**	temps linéaire	PSPACE-complet	PSPACE-complet
Extensions de **NNFAT**			
NNFAT$^{\mathbf{C}}_{\mathsf{F}}$	temps polynomial	coNP-complet	coNP-complet
NNFAT$^{\mathbf{C}}_{\mathsf{C}}$	temps linéaire	PSPACE-complet	PSPACE-complet

TABLE 2: Résultats de complexité pour ST, IS-DET et IS-MON.

Language	CHOICE	SEQUENCE	NEGATION	EXTRACT-PRECOND
Langages minimalement complets				
NNFAT$^{\mathbf{T}}$, **NNFAT$^{\mathbf{C}}$**	✓	○	✓	○
O-PDDL$^{\mathbf{T}}$, **O-PDDL$^{\mathbf{C}}$**	✓	○	○	✓
E-PDDL$^{\mathbf{T}}$, **E-PDDL$^{\mathbf{C}}$**	✓	○	○	○
Restrictions des langages impératifs				
O-NCSTRIPS	?	○	○	✓
E-NCSTRIPS	?	○	○	○
NPDDL$_{\mathbf{nf}}$	✓	○	○	✓
Extensions des langages impératifs				
O-PDDL$^{\mathbf{T}}_{\text{seq}}$	✓	✓	○	○
O-PDDL$^{\mathbf{C}}_{\text{seq}}$	✓	✓	?	○
O-PDDL$^{\mathbf{T}}_{\text{and}}$	✓	?	○	○
O-PDDL$^{\mathbf{C}}_{\text{and}}$	✓	?	?	○
O-PDDL$^{\mathbf{T}}_{\text{not}}$, **O-PDDL$^{\mathbf{C}}_{\text{not}}$**	✓	?	✓	○
E-PDDL$^{\mathbf{T}}_{\text{seq}}$	✓	✓	○	○
E-PDDL$^{\mathbf{C}}_{\text{seq}}$	✓	✓	?	○
E-PDDL$^{\mathbf{T}}_{\text{and}}$	✓	○	○	○
E-PDDL$^{\mathbf{C}}_{\text{and}}$	✓	○	?	○
E-PDDL$^{\mathbf{T}}_{\text{not}}$, **E-PDDL$^{\mathbf{C}}_{\text{not}}$**	✓	○	✓	○
Extensions de **NNFAT**				
NNFAT$^{\mathbf{C}}_{\mathsf{F}}$	✓	○	✓	○
NNFAT$^{\mathbf{C}}_{\mathsf{C}}$	✓	?	?	○

TABLE 3: Difficulté d'effectuer une transformation. "✓" signifie que la transformation est possible en temps polynomial. " ?" indique les questions ouvertes. ○ signifie que sous une hypothèse sur les classes de complexité, la taille du résultat de la transformation n'est pas polynomial en la taille de l'entrée.

FIGURE 1: Résultats de concision pour les langages minimalement complets, les langages **O-NCSTRIPS**, **E-NCSTRIPS**, et les représentations en circuit des extensions de **NNFAT** (i.e. $\mathbf{NNFAT}^{\mathbf{C}}_{\mathbf{C}}$ et $\mathbf{NNFAT}^{\mathbf{C}}_{\mathbf{F}}$). Un arc de L_1 vers L_2 signifie que L_1 peut être traduit en L_2 en temps polynomial. Un arc barré de L_1 vers L_2 signifie que sous une hypothèse sur les classes de complexité, il n'existe aucune traduction de taille polynomiale de L_1 en L_2, pour aucune des représentations. Un arc barré par $\times_C$ signifie que la séparation en termes de concision n'est démontrée que pour les circuits. Un arc étiqueté par C signifie que l'existence d'une traduction en temps polynomial n'est prouvée que pour les circuits.



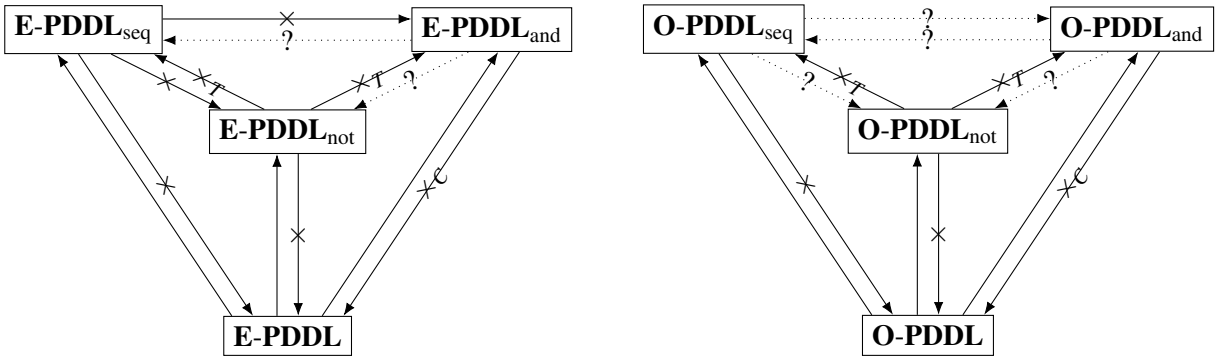


FIGURE 2: Résultats de concision pour les extensions de **E-PDDL** et **O-PDDL**. Un arc barré par $\times_T$ signifie que la séparation en termes de concision n'est démontrée que pour les arbres.

Acknowledgements

First I want to thank my advisors, Alexandre Niveau and Bruno Zanuttini, for their patience, kindness, and being always there when I had questions or needed help. Our work sessions were a lot of fun, as well as the social events we attended together. Of course all the results in this thesis are results of our common work, and I am grateful to them for giving me the freedom to choose research questions which we were working on together. Thank you also for motivating me to go to conferences and workshops where I met many interesting people and learned a lot about other research directions in AI, and for letting me try out teaching and even giving lectures in French, it was an awesome experience. In the end it seems impossible to list all the things Alex and Bruno have done for me, but the German term for "doctoral advisor" which is "Doktorvater" and literally means "doctoral father" is a good description of what they have been for me.

I am grateful to Stefan Mengel and Bernhard Nebel for agreeing to be the reviewers of my thesis and for carefully reading my dissertation and for spotting mistakes which seem so hard to notice. Also thank you to Hélène Fargier and Peter Jonsson for being in my defense jury, and to Andreas Herzig for being in both my defense jury and my follow-up committee and giving me numerous tips for my PhD.

I also would like to thank the supervisor of my Master's thesis Manuel Bodirsky, who encouraged me to do science after graduating from the TU Dresden, Maja Pech for being the supervisor of my Bachelor's thesis, and Reinhard Pöschel for being a member of my follow-up committee and the reviewer of both my Bachelor's and Master's thesis. You all made me love universal algebra and made me believe I am capable of doing research in mathematics, which determined my future life path.

Thank you my fellow PhD students Josselin, Mihail, Romain and Sébastien, with whom we shared our room at work, and all other colleagues from the GREYC lab, you told me a lot about the French culture and language.

I want to thank my mom Natalia and dad Wladimir for being awesome parents and for supporting me throughout my life. A special thank you to my dad and to my math teacher Thomas Riede for influencing my choice of profession by their example, and to my mom for helping me to prepare for exams during my studies.

I am also grateful to all my friends and relatives whose presence in my life is a reason to never give up. Especially I want to express my grattitude to Irina for doing rehearsals of my presentations over and over again, for showing genuine interest for my research and for supporting me during hard times in life, a "Dankeschön" to Cécile for checking my french summary for mistakes and a thank you to Raghad for giving me tips on teaching. I would like to thank my sports friend Tomaš for not breaking my finger this time, so that I could finish writing my thesis on time. Finally, thank you Lisa for your $\exists$-ence.

Part I

Introduction and Basics

CHAPTER 1

Introduction

1.1 Motivation

Knowledge Compilation

One of the key problems in computer science is the representation of information. When designing algorithms to solve computational problems we always want them to terminate in a reasonable time. Whether the time can be considered reasonable depends on the interconnection between the size of the input and the amount of time necessary to output a solution. Additionally we want that the computation never exceeds the amount of available resources like memory space. This raises the question about how to measure the size of the input, since the same information can be represented in many different ways. *Knowledge compilation (KC)* can be seen as a branch of computer science which studies the dependence between various representations of information in artificial intelligence and the complexity of computational tasks usually performed on these representations. The key idea of knowledge compilation is that many inputs to (repeated) computational tasks can be divided into two parts: the *fixed* part called *knowledge base* and the *varying* part called *query*. The knowledge base is often much bigger than the query and while performing the computation it must be treated in a certain way. To avoid the same computation taking place again and again one could think of performing it just once to adjust the knowledge base for answering a specific query task (which is called the *off-line reasoning*). Answering the query task with the preprocessed knowledge base is called *on-line reasoning* and can be performed faster than without preprocessing. This approach is especially useful if the original computational task was not tractable, but becomes tractable after the preprocessing, and the preprocessing does not exponentially increase the size of the knowledge base. Typically there is a trade-off between the *succinctness* of a representation and the computational efficiency of answering queries: the more compact a representation becomes, the more difficult is the extraction of information. More explanations and examples of this approach can be found in e.g. (Cadoli and Donini, 1997).

Originally this approach was developed to compare sets of propositional formulas called *(target) languages* (for example, Darwiche and Marquis (2002); Fargier and Marquis (2008)). These languages were studied from the perspective of the complexity of queries like *clausal entailment* (whether some information represented by a disjunction of literals is a logical consequence of the knowledge base) or *consistency* (whether the knowledge base is noncontradictory). The result of the research in this area can be summarized in a *knowledge compilation map*, like in Darwiche and Marquis (2002). According to this map the user chooses the most succinct target language which still allows for efficient answering of all queries the user is interested in, and then the knowledge base is automatically translated into the target language from a specification language which is more convenient for humans. Although originally knowledge compilation was studying languages with the same semantics (i.e. Boolean formulas with the standard interpretation), the framework can be extended to compare very different languages, for

example, like in the approach of heterogeneous compilation (Fargier et al., 2013).

Since we will use the word “language” very often, we would like to illustrate the concept with a real-life example: the English phrase “On Thursday, Alice went to work by tram, stayed there for nine hours, then she took the bus to a dancing club and danced tango for two hours, then she went home by foot and went to bed immediately after arriving home.” encodes some information and requires some memory space to store this expression. And a query task could be of the form “Did Alice dance jive on Thursday?”. A more concise representation could be obtained by dropping the vowels: “n Thrsd, lc wnt t wrk b trm, std thr fr tw hrs, thn sh tk th bs t dncng clb nd dncd tng fr tw hrs, thn sh wnt hm b ft nd wnt t bd mmdtl ftr rrvng hm”. This representation is still suitable to answer the query, but a human would have a hard time understanding the phrase and answering the question. Another option would be to use widely known abbreviations and digits and drop the prepositions and articles: "Alice, Th, tram: work (9 hrs), bus: dancing club, tango (2 hrs), foot: home, bed". This phrase is shorter than the original one, too, but it is also much easier to read than the one without vowels, and the question about whether she danced jive on Thursday is probably answered even quicker than when being given the original phrase. Thus, compiling knowledge in the second manner might be more practical when the expected queries will be of the kind as the example.

From the example with the human languages we see, that the idea of knowledge compilation is universal and can be applied whenever some information will be queried repeatedly. Actually the idea of sorting vocabulary in a dictionary or things at home is the same: we invest time before the actual work to save time or resources after. The key difference is that knowledge compilation is a technique to address intractability, and a knowledge compilation map usually is a table with “yes” or “no” entries depending on whether a language supports a query in polynomial time or not.

In this work we will apply techniques to *automated planning*. We will see later that even very basic queries for expressions with simple structure are often intractable and thus this is a good field for the application of the KC approach.

Planning

Automated planning is a research area of artificial intelligence which deals with achieving goals from an initial state. A concise definition from (Geffner and Bonet, 2013) sounds as follows: “Planning is the model-based approach to autonomous behavior where the agent selects the action to do next using a model of how actions and sensors work, what is the current situation, and what is the goal to be achieved.” “Model-based” means here, that we describe a model of the world and the actions that can be undertaken as an input to the planning algorithm (often called simply *planner*) which then searches for a solution. A planning model usually consists of a set S of states which can change after the application of actions $a \in A$. Executing actions is associated with positive costs which can depend on the applied action and the current state. Transitions from one state to another are achieved via the effects of actions. Depending on the knowledge about the current state, the properties of the actions and the action costs one can distinguish many variants of the tasks. The most basic one is *classical planning*, with only one agent, all actions being deterministic and durationless, action costs depending only on the action and not on the state, a known initial state and a fully observable world. Many problems can be efficiently reduced to classical planning. For example, if an action is “kind of deterministic” and has a “typical” effect (like pushing the door to open it), and all alternative effects are considered to be exceptions (the door breaks after pushing), we can plan in the classical setting an re-plan from the current state if something goes wrong. These assumptions, however, are often still insufficient to model the real-world situation (for example, if there is no “base case”, like when throwing the dice), and one of the obvious relaxations which can be made to achieve better realism of the model is dropping determinism. An action is said to be nondeterministic if the execution of it in a state can result in several successor states.

The notion of a solution to a planning problem depends on the setting and on the requirements. For example, in classical planning a natural notion of solution is a *linear plan*, which is just a sequence of actions, whereas for planning with nondeterminism it is more reasonable to consider *policies*, i.e. plans where the action to be undertaken next depends on the current state or observations made before.

The most natural decision problem in automated planning is to decide for a given model together with

the inputs (initial state and goal states) whether there exists at least one plan (i.e. this problem depends on what notion of plan we use). This is known as the *plan existence problem* (PLANEX) and is PSPACE-hard even for a simple STRIPS-formalism (Fikes and Nilsson, 1971) for classical planning (Bylander, 1994).

PLANEX is a decision problem which is asked for a whole planning domain, but while searching for a plan a planner asks many queries to actions and states on the way. The efficiency of such a planner depends on the complexity of these queries, which motivates the complexity study of various representations for actions. As we will see later, the complexity of reasoning with representations of nondeterministic actions stays in PSPACE even for quite sophisticated formalisms, but we will prove that these formalisms differ in the succinctness of representations. Thus it might be useful to choose more succinct representation knowing that the resulting efficiency of the planner might not change. We will deduce some practical consequences for the choice of representations by programmers from our results in the end of each chapter.

Actions

As we argued above, a central aspect of the description of planning 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 planners to operate on them while searching for a plan. The particular representation can be crucial for the performance of the planner. This can be seen from a (very informal representation of a) generic planning algorithm (Algorithm 1): we

```
Input: Initial state, actions, goal state
Output: Plan that always reaches goal
while Goal not ensured do
    if Actions available for choice then
        Choose action not yet chosen at this step;
        if Current plan guarantees precondition of chosen action then
            Append action to current plan;
        end
    end
    else
        Remove last action from current plan;
    end
end
return Current plan
```

Algorithm 1: Generic planning algorithm

observe that we need to check over and over again whether a chosen action is applicable after executing a partial plan. We will see later that there are several queries and/or transformations, which could be used to refine this algorithm, and thus the complexity of the algorithm depends on the complexity of those *repeated* queries.

In this thesis we consider different representation languages which can be used to describe nondeterministic actions within the formal framework of the knowledge compilation map from (Darwiche and Marquis, 2002), i.e. we study representation 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?). We focus on propositional domains in which states are assignments to a given set of Boolean variables. This is less general than studying relational descriptions as in **PDDL**, still the standard approach is reasoning with propositional representations (Helmert, 2006; Gnad et al., 2019).

1.2 Outline

We will consider three main (basic) representation languages with some of their possible modifications. The basic languages will be given by a Backus-Naur form (BNF) of the grammar defining the possible action descriptions, with *elementary/atomic actions* which can be combined to more complex action descriptions via *connectives*. For their modifications we will either extend the BNF or impose restrictions on the structure of the action descriptions.

One basic language is a grounded, nondeterministic version of **PDDL** (McDermott, 1998), and another one is a structurally similar language with an alternative semantics for the connective of parallel composition, which resembles that of the Dynamic Logic of Parallel Propositional Assignments **DL-PPA** (Herzig et al., 2019). These two languages are *imperative* in the sense that modifications are explicitly specified. This is useful when specifying actions by hand. The third main language is a logical representation of actions by NNF action theories (Cimatti and Roveri, 2000). It is *declarative* in the sense that it specifies how the states before and after the execution are related.

The modifications of the above languages which we are interested in are: syntactically restricting the imperative languages to obtain **STRIPS**-like structure Fikes and Nilsson (1971); enriching the imperative languages by various connectives: by adding the sequence connective ; with the interpretation of the operator of sequential composition from **DL-PPA**, with the negation connective $\neg_{\min}$ denoting action negation (Broersen, 2004) and with the conjunction connective $\wedge$ denoting a "common execution"/intersection; and by enriching NNF action theories by two different connectives C and F, which both formalize the notion of persistence (McCarthy, 1980) in different ways.

We chose to focus our study on those few languages because they use different constructs, different structural restrictions on expressions, and different representations of persisting values. Among them, the basic languages are minimally complete for representing nondeterministic actions, in the sense that removing one of their connectives would yield a language which is not fully expressive. For these we provide a systematic study. The other languages are well-studied restrictions (**STRIPS**-like languages) and natural extensions, and for them we provide a complete picture regarding the complexity of queries. The complexity of transformations and the relative succinctness of languages, however, is more difficult to determine, since the separation proofs are usually based on different complexities of queries in the languages, and if e.g. the queries are all PSPACE-complete (we will see that it is quite often the case for extensions) then this approach does not work.

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

To sum it up, our contributions are the following:

- we define a formal knowledge compilation framework for nondeterministic planning, including the notions of action languages and translations between them and queries and transformations which are the most natural to consider in this setting,
- then, we define three minimally complete languages for which we obtain a complete picture regarding their complexity and relative succinctness for both the tree and the circuit representation,
- finally, we study variants of these languages, and for all of them we give a complete picture for complexity of queries (usually for both tree and circuit representations, except for Chapter 7 where we do not make the distinction of representations for some languages, and Chapter 9 where all results are given only for circuits), and several succinctness results and results about the complexity of transformations (however, with gaps which are yet to be filled).

We will proceed as follows: in Chapter 2 we give an overview of related work and explain the differences to our study. In Chapter 3 we give some preliminaries from propositional logic and complexity theory. Then, in Chapter 4 we formally define the main concepts of our study: on the one hand, action

languages as the central object of the study, and on the other hand – queries and transformations, whose complexity will be used to construct the knowledge compilation map. In Chapter 5 we define the action languages which are basic for this work. For them we give the complexity results together with the detailed proofs which contain almost all important proof ideas in Chapter 6. In Chapter 7 we define variants of the basic action languages which are obtained by imposing some syntactic restrictions on them, and give the corresponding complexity results. In Chapters 8 and 9 we proceed to defining extensions of the basic action languages. As we said, reasoning about extensions is harder than about the basic languages, so that we have less proven results and more open questions in those chapters. In the end of each chapter with technical results we will give a brief summary with a table or diagram illustrating those results. We conclude by giving a summary of the work and commenting on perspectives in Chapter 10. Appendix A contains a table of the most important notions in this work and can be used to look up the definitions without the need to search for it inside the thesis.

1.3 Publications

Our results are partially published in the following conference papers:

1. S. Scheck, A. Niveau, and B. Zanuttini. Knowledge Compilation for Action Languages. In *Journées Francophones sur la Planification, la Décision et l'Apprentissage pour la conduite de systèmes (JFPDA 2020)*, 2020

2. S. Scheck, A. Niveau, and B. Zanuttini. Knowledge Compilation for Nondeterministic Action Languages. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, pages 308–316, 2021b

3. S. Scheck, A. Niveau, and B. Zanuttini. Explicit Representations of Persistency for Propositional Action Theories. In *Journées Francophones sur la Planification, la Décision et l'Apprentissage pour la conduite de systèmes (JFPDA 2021)*, 2021a

4. S. Scheck, A. Niveau, and B. Zanuttini. A KC Map for Variants of Nondeterministic PDDL. In *16es Journées d'Intelligence Artificielle Fondamentale (JIAF 2022)*, 2022

The chronological ordering of the articles goes hand in hand with the introduction of new action languages, but the results about a new action language also involve its succinctness relation to the previously introduced ones, and therefore it is not possible to say "the results from this article correspond to that chapter". This is especially so because we did not yet publish results about **O-PDDL** nor its variants, and these results are distributed over the whole work. There are also no published results about the action conjunction yet, these are given in Chapter 8. What can be related to articles are the languages **E-PDDL**$_{\text{seq}}$ (main results are in Scheck et al. (2021b)) and **E-PDDL**$_{\text{not}}$ (results published in Scheck et al. (2022)). This thesis is thus a mix of the above articles together with many yet unpublished results.

CHAPTER 2

Related Work

In this chapter we first give an overview of what we know about current research in related research areas, and then explain how our approach and results differ from those in the literature.

2.1 State of the art

The choice of an internal representation language has long been recognized by the planning community as one key to building efficient planners, starting with CMBP (Cimatti and Roveri, 2000), which uses binary decision diagrams for representing belief states and actions in conformant planning tasks. In the stochastic setting, SPUDD (Hoey et al., 1999) likewise uses algebraic decision diagrams for representing value functions and actions.

To et al. (2015) provide an abstract view of such *symbolic* approaches in the setting of nondeterministic actions, and consider other representation languages, namely CNF, DNF, and prime implicate representations. More recently, SYMPLE (Speck et al., 2018) uses edge-valued multi-valued decision diagrams for cost-optimal planning. At another extreme, CONFORMANT-FF (Hoffmann and Brafman, 2006) uses an implicit (hence very succinct) representation, but relies on satisfiability testing for querying the representation.

The question of representation languages for belief states and actions also arises beyond the classical, propositional setting. In the stochastic setting, apart from SPUDD cited above, RBAB (Lesner and Zanuttini, 2011) uses algebraic decision diagrams for value functions, but (probabilistic) **PDDL** for actions. Yet another domain in which efficient representations have been used is that of epistemic planning, with the use of (**DL-PA**) accessibility programs (Charrier and Schwarzentruber, 2017) or of binary decision diagrams (van Benthem et al., 2018) for representing epistemic actions ("event models") succinctly.

Despite these numerous approaches, as far as we know there has been no systematic study of the complexity of queries/transformations and succinctness for action languages. This is however an important problem, since many planners start by transforming the action specifications into some representation considered to be suitable for the particular algorithm (Hoey et al., 1999; To et al., 2015; van Benthem et al., 2018; Speck et al., 2018). This thesis is thus a first attempt at a systematic study of various action languages from the point of view of knowledge compilation. We start by studying nondeterministic actions, which lie at the core of fully observable nondeterministic planning and of conformant planning (Rintanen, 2004; Palacios and Geffner, 2009; Geffner and Bonet, 2013; Muise et al., 2014; To et al., 2015; Geffner and Geffner, 2018), because, as we will argue later in Chapter 6, the results for deterministic actions in our setting would be trivial. We also focus on propositional domains, in which states are assignments to a given set of propositions.

Works with related objectives do exist, but the focus has been on other aspects of planning, like the representation of plans (Bäckström and Jonsson, 2012), axioms (Thiébaux et al., 2005), or action

costs (Speck et al., 2021). The closest line of work which we are aware of is the one pioneered by Bäckström (1995), who study what features of an action language (e.g., negative preconditions) can be emulated by another language for variants of **STRIPS**. However, this study considers a less demanding form of translations between languages than we do, allowing translations, in particular, to introduce new variables. Nebel (2000) studies similar questions, up to preservation of plan length (up to a polynomial increase, typically). Contrastingly, as explained in Subsection 4.1.3, we are interested in translations which preserve the exact semantics of actions, including the set of variables involved. The difference of our approach and that of Nebel resembles that between the Tseitin transformation and equivalent transformations: while the first ensures equisatisfiability, the latter ensure that set of models remains the same.

Farther from our goal, but also taking the point of view of knowledge compilation about planning, Bäckström and Jonsson (2012) study representations of plans with respect to their size and to the complexity of retrieving the individual actions which they prescribe at each step. Another class of objects related to planning for which a knowledge compilation perspective has been taken is the class of action costs (Speck et al., 2021). Also related is the translation of HTN specifications into **PDDL** proposed by Alford et al. (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 at a given level.

Finally, in the knowledge compilation literature, a lot of work has focused on languages for representing Boolean functions (Darwiche and Marquis, 2002; Oztok and Darwiche, 2015; Capelli et al., 2021). We also consider Boolean functions (over variables before and after the action), but our study departs from these since we consider languages which are not natural for representing standard Boolean functions, namely, **O-PDDL**, **E-PDDL** (defined in Chapter 5) and their variants. These languages have a "built-in" distinction into predecessor- and successor variables and thus for the size of e.g. an **E-PDDL** representation of a given Boolean function it might be crucial which variables have been put into which set, which relates this representation to the influence of the chosen ordering of variables on the size of the minimal OBDD (Meinel and Theobald, 1998, Chapter 8). There have been other studies of settings which likewise depart from standard Boolean functions. In particular, (Fargier and Mengin, 2021) take a knowledge compilation perspective on languages for representing preferences. Some of these languages bear important similarities with action languages (in particular, a preference relation also relates two propositional states to each other, and the *ceteris paribus* assumption resembles persistency of fluents), however, the natural problems to study for preferences and for action languages are different.

A general difference of our approach to many others in knowledge compilation is that we consider very rich languages which are fully expressive. A language is said to be more expressive than another if the first one can decribe everything the other can, but not vice versa (an example of a study involving not fully expressive languages is (Fargier and Marquis, 2008)). The advantage of not fully expressive languages is their higher computational efficiency. In our work, however, we concentrate on languages which are fully expressive, i.e. able to represent every nondeterministic action. This relates our work with the classical knowledge compilation map of Darwiche and Marquis (2002), where only complete languages were studied.

What is different from (Darwiche and Marquis, 2002) is the manner in which the languages under consideration are obtained: in the classical setting there is one "superlanguage" of Boolean formulas in negation normal form, and all languages are its subclasses obtained by some syntactic restrictions: for example, a bounded depth of the formula (CNF, DNF), restrictions on sets of variables shared by subformulas (DNNF), ordering of the variables (OBDD) etc., thus the basic language NNF is "above" the rest. By imposing such restrictions tractability of certain queries can be ensured. We, however, take basic languages which are "in-between", and then consider their variants obtained by adding connectives taken from literature in knowledge representation (we will call languages obtained in this manner "extensions") or restricting the syntax to resemble "real-life" languages like STRIPS (these languages will be called "restrictions"). This resembles the approach by Nebel (2000) where the basic language of STRIPS is made "richer" and more expressive.

2.2 Novelty

Most complexity studies in planning restrict themselves to the propositional setting, first because planners usually operate on grounded representations (Gnad et al., 2019), second – because it is easier to reason in the propositional setting, but the results are already general enough because the grounding process is usually straightforward.

We start by defining three basic languages, of which one is inspired by the existing KC literature and at the same time generalizes usual internal representations, namely formulas in negated normal form. The choice of concretely this language as a typical representative of a declarative language is of secondary importance, and the particularly chosen NNF action theories are suitable to illustrate all differences of these two approaches, in particular: the problem of circumscription/ramifications (McCarthy, 1980; Kartha and Lifschitz, 1994), the notion of translation for languages with a different semantics, and the succinctness separation proofs. Since the complexity of Boolean formulas is a very well studied domain, we do not need to put much effort into determining the complexity (most of the results in our work appear either directly in Darwiche and Marquis (2002) and Lang et al. (2003) or can be deduced from there). The novelty of our research regarding action theories is the extension of the language by connectives which aim at adding an imperative flavour to a declarative language and thus could contribute to avoid facing the frame problem when specifying actions. One of these connectives, $\mathsf{C}_{X,V,F}$, aims at formalizing the idea of the frame as in Kartha and Lifschitz (1994) in a way that allows to reuse already existing action descriptions without any change. Contrary to other approaches we just define a simple grammar where the corresponding connectives can appear anywhere in the formula, which generalizes already existing formalisms, and again, general results about the very rich language studied in this work can be transferred to special cases which are or will be studied in the literature. The other connective is not inspired by any existing notion but by direct intuition.

As for imperative languages, the original motivation for the basic action languages which we will consider is a formal propositional abstraction of the nondeterministic extension **NPDDL** of **PDDL** proposed in Bertoli et al. (2003). It turns out that a straightforward formalization of the grammar leads to a definition of action descriptions which matches the general definition of effects by Rintanen (2003), where compound effects are defined as "conjunctions" and "disjunction" of other effects. These effects are nondeterministic and the definition is general enough to ensure that each possible nondeterministic action can be represented. Moreover, this setting is even more general than ours because the author considers probabilities for nondeterministic choices, which we do not do. In his work, Rintanen shows that for each nondeterministic effect there are equivalent effects in different normal forms. The syntax of those effects is very close to our definition of the syntax of **O-PDDL** and **E-PDDL** (Chapter 5), but we do not make the assumption that variables can be set only by one occurence of this variable in an effect at once. Two different ways to relax this condition are the reason why we consider two different nondeterministic variants of **PDDL**.

The first solution is inspired by the semantics of STRIPS (Fikes and Nilsson, 1971; Lifschitz, 1987), when the transition from a predecessor state to a successor state happens by first deleting the "Delete list" and then adding the "Add list". Thus we can say that "addition overrides deletion" and whenever a conflict between effects occurs it is resolved in favor of positive effects.

Another solution is the one used e.g. by Nebel (2000), which is to say that an application of an action in a state is undefined if the positive and the negative effects are inconsistent. This is in the spirit of the parallel composition $\sqcap$ from the Dynamic Logic of Parallel Propositional Assignments **DL-PPA** (Herzig et al., 2019). **DL-PPA** is an extension of **DL-PA** (Balbiani et al., 2013), a logic originally invented to provide a framework for reasoning about imperative programs.

A particular result from Herzig et al. (2019) is that every program π in **DL-PPA** can be translated into an equivalent **DL-PA** program in polynomial time, which implies that the connective of parallel composition can be eliminated without explosion in size. This translation, however, uses additional variables which does no harm because of the semantics of **DL-PA** which assumes the set of what we call state variables to be infinite. Our approach is more restrictive since we compare expressions analogous to Boolean formulas and assume the models to be assignments to finite sets of variables, and thus this translation is not possible in our setting. From this point of view our study can be seen as "which

connectives can be expressed by other connectives without the use of additional variables and without a super-polynomial blow-up in size", and whether the parallel composition can be compiled away using other connectives is an example for a natural question in our setting. The inspiration by **DL**-**PPA** and the very similar semantics are the reason why we have chosen the **DL**-**PA**-like notation instead of that which is more typical for the planning literature.

In the formal framework which will be defined in Chapter 4 we give a formal meaning to actions: an action in our propositional setting is simply a collection of *state transitions* (s, s') and thus for a set of propositional state variables P a P-action corresponds to a Boolean function over $P \cup P'$ where P' stands for P after the execution of an action. Therefore we can say that we study classical knowledge compilation for languages inspired by planning, but contrastingly to many other works on knowledge compilation for representing Boolean formulas we do not pay much attention to trying to find sublanguages which allow for many tractable queries, but rather concentrate on expressivity of connectives and their interaction. Thus our most important and practically relevant results will be the *succinctness separation results* which show that some formalism is much richer than the other. Also in contrast with many studies is that except for Chapter 7 we do not impose any restrictions on the depth of the formula, ordering of the variables or ordering of the connectives, i.e. the grammar will be as general as possible.

We remark that many of the presented results are interesting on their own, even without a relation to a concrete action language that exists "in real life". For example, a particular succinctness result could contribute to the design of some "real" action language in future, or a proof idea can be used in other knowledge compilation studies not directly related to planning.

CHAPTER 3

Preliminaries

Contents

For the sake of completeness we will give a brief, but necessary background in nondeterministic planning, logic and computational complexity.

3.1 Model-based Domain-independent Planning

Since our study is mainly motivated by the operations on action descriptions performed by a planner, we first give the formal definition of a planning task.

The following notation is standard in the literature (with little variations, concretely this notation follows that by (Geffner and Bonet, 2013, p. 65)): a general non-deterministic (fully observable) planning model without action costs is a tuple $\Sigma := \langle S, S_0, S_G, A, F \rangle$ where

- S is a finite set of possible states
- $S_0, S_G \subseteq S$ are the set of possible initial and goal states respectively
- A is a set of actions, with $A(s) \subseteq A$ the set of actions applicable in $s \in S$
- F is a nondeterministic transition function with $F(a,s) \subseteq 2^S$ for all $a \in A, s \in S$

In our work, for the sake of simplicity, we will write $a(s)$ instead of $F(a,s)$ and not distinguish between action symbols and actions, because the subject of the study are computational properties of action representations and not the planning formalisms themselves.

For a sequence of actions $\langle a_1, \ldots, a_k \rangle$ and $0 \leq i \leq k-1$ we define its *execution* (with S_0 as above): $S_{i+1} := \{s_{i+1} \in a_i(s_i) \mid s_i \in S_i\}$. Then we can define various notions of solutions. For example, a sequence of actions $\langle a_1, \ldots, a_k \rangle$ is a *strong linear plan* for Σ if each a_{i+1} is applicable in S_i for all $0 \leq i \leq k-1$ (i.e. the action sequence is *executable*) and $a_k(S_{k-1}) \subseteq S_G$. The problem of checking whether a given sequence of actions is indeed a plan is known as *belief tracking*.

A more flexible concept than linear plans is the notion of policies: a *policy* is a mapping $\pi : S \to A$ which defines which action is to be chosen in which state. An execution of a policy π in state s_0 is a sequence $s_0, \pi(s_0), s_1, \pi(s_1), s_2, \pi(s_2), \ldots$ with $s_{i+1} \in (\pi(s_i))(s_i)$. A policy is an *acyclic safe solution* (this term is used for instance by (Ghallab et al., 2016)) to Σ if each of its executions in any $s_0 \in S_0$ is acyclic (i.e. there is no repetition of states) and necessarily reaches a goal state $s_G \in S_G$ at some point.

In the propositional setting, initial and goal states are often assumed to be represented by a Boolean formula, i.e. the set of goal states is defined to be the set of all states which have a given (desired) property described by the formula.

3.2 Negation Normal Form

In this thesis we will use the usual logical connectives for Boolean formulas: $\wedge$ (conjunction), $\vee$ (disjunction), $\neg$ (negation), $\to$ (implication), $\leftrightarrow$ (equivalence) and $\oplus$ ("exclusive or") with the usual interpretation. For the Boolean values we will use $\top$ and $\bot$ for "true" and "false" respectively.

The first KC map (Darwiche and Marquis, 2002) was designed to compare languages of Boolean formulas in negation normal form.

Definition 3.1 (NNF)**.** Let Q be a set of variables. A *literal* over Q is a variable $q \in Q$ or its negation $\neg q$. A Boolean formula φ over Q is said to be 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. The variables which occur in φ will be denoted by $\mathrm{V}(\varphi)$.

Example 3.2. *Let $Q = \{p, q, r\}$. The formula $\varphi = p \vee (q \wedge \neg r)$ is an NNF formula, but $\psi = q \to \neg(p \wedge r)$ is not an NNF formula, because it contains a negation of a non-atomic subformula and an implication operator.*

The set of NNF formulas is complete, i.e., any Boolean function can be expressed by an NNF formula. In particular, a Boolean formula that contains only the operators $\vee$, $\wedge$, $\to$ and $\neg$ can be transformed into an equivalent NNF formula in linear time. $\top$ and $\bot$ are formally not part of the grammar we defined above, but they can be expressed as $p \vee \neg p$ and $p \wedge \neg p$, respectively, and therefore we allow for NNF formulas to contain $\top$ and $\bot$-symbols. We will assume that all logical formulas in our work are in negated normal form unless stated otherwise. Since $\varphi \leftrightarrow \psi$ is equivalent to $(\varphi \to \psi) \wedge (\psi \to \beta)$, a formula with a bounded depth d of nesting of $\leftrightarrow$ can be transformed into an equivalent NNF formula in linear time (the time required for the transformation is exponential in d, but d is bounded), too. Therefore we will sometimes use formulas containing $\leftrightarrow$ for the sake of simplicity and compactness, but keeping in mind that we actually mean an equivalent 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 a set of variables Q and an assignment t to the variables in Q, we write $t \models \varphi$ ("t satisfies φ") if φ evaluates to true under the assignment t. A special case of this notation is $t \models q$ for a variable q, which means that $t(q) = \top$. The set of all models of φ will be denoted by $\|\varphi\|$. A formula φ is called *satisfiable* if $\|\varphi\| \neq \emptyset$, i.e. there exists at least one model of φ, and *unsatisfiable* otherwise. Two formulas φ and ψ are *equivalent*, denoted by $\varphi \equiv \psi$ if $\|\varphi\| = \|\psi\|$.

A formula is *(partially) quantified* if some of its variable occurences are bound by a quantifier $(\exists, \forall)$. If all variables are bound, it is called *fully quantified*. A quantified Boolean formula is in *prenex normal form* if it consists of two parts: the quantifier-free second part called *matrix* (which is a normal Boolean formula) and the first part containing only quantifiers.

Throughout the whole document we will use the greek letters $\varphi, \psi, \Phi, \Psi, \ldots$ for logical formulas and capital latin letters $A, B, S, T, Q, \ldots$ for sets of variables.

3.2.1 Boolean Languages

In the following we will consider several special classes of NNF formulas which play an important role in knowledge compilation (but not only there). In the knowledge compilation literature these classes are called *languages* (Darwiche and Marquis, 2002). The basic Boolean language in the literature is that of NNF formulas, and its subsets will be called *(sub)languages*. By NNF (respectively CNF, DNF, ...) we might refer to the language or to just one formula, if the meaning is clear from the context, but usually we will say "NNF formula" to avoid misunderstandings.

3.2.2 Conjunctive Normal Form

Definition 3.3 (CNF)**.** Let Q be a set of variables. A *clause* over Q is a disjunction of literals. A Boolean formula φ over Q is said to be in *conjunctive normal form* (CNF) if it is a conjunction of clauses.

Example 3.4. *Let $Q = \{p,q,r\}$. $\gamma_1 = p \vee \neg p \vee q$ and $\gamma_2 = r \vee \neg q$ are clauses, and their conjunction $\varphi = \gamma_1 \wedge \gamma_2 = (p \vee \neg p \vee q) \wedge (r \vee \neg q)$ is a CNF-formula.*

The set of all CNF formulas is complete, because any NNF formula can be transformed into an equivalent CNF formula using the distributivity law, De Morgan's laws and the elimination of double negations.

A formula is said to be a 3-CNF if it is a CNF with at most three literals per clause.

3.2.3 Disjunctive Normal Form

Definition 3.5. Let Q be a set of variables. A *term* over Q is a conjunction of literals. A Boolean formula φ over Q is said to be in *disjunctive normal form* (DNF) if it is a disjunction of terms.

Disjunctive normal form is the dual concept of CNF.

Example 3.6. *The set of DNF formulas is complete because every Boolean formula ψ is equivalent to the disjunction of terms t_s^ψ, where s ranges over all models of ψ and $t_s^\psi = (\bigwedge_{s \models q} q) \wedge (\bigwedge_{s \not\models q} \neg q)$. I.e. any Boolean function can be represented by a disjunction of terms, of which each one represents a tuple which is mapped to $\top$ by this function.*

3.2.4 Representations

One of the key aspects studied in knowledge compilation is the succinctness of languages which are equally expressive. In order to reason about succinctness, it is crucial to define the *size* of formulas. There are two natural variants of measuring the size.

The first variant corresponds to a representation of the formulas φ in the language by their syntactic *rooted tree*. In this case, the size of a formula is just the number of symbols occuring there (excluding the parentheses, which are actually not necessary, but rather used for better readability).

The second variant, which is the one used in the literature about knowlegde compilation (Darwiche and Marquis, 2002), corresponds to the representation of these formulas φ by a rooted directed acyclic graph, or *circuit*, where a root of a subformula can be the end node of many arcs. This allows to store isomorphic subformulas one time independently of the number and place of their occurences in the formula. A circuit can be *reduced* (this reduction process is similar to the one underlying binary decision diagrams (Bryant, 1992)) by iteratively identifying the roots of two isomorphic subformulas to each other, until no more reduction is possible. It is important to emphasize that we consider the tree representation to be a special case of the circuit representation.

For example, when recursively defining a function f which maps formulas to other formulas, we might use $f(\varphi \wedge \psi) := (f(\varphi) \vee f(\psi)) \wedge f(\varphi)$. In the tree representation the size of the image of a formula would in general be exponential because of the double occurence of $f(\varphi)$, but in the circuit representation we can store the circuit of $f(\varphi)$ once and refer to it twice. A concrete example of a reduced circuit representation and the tree representation of a formula can be seen in Figure 3.1.

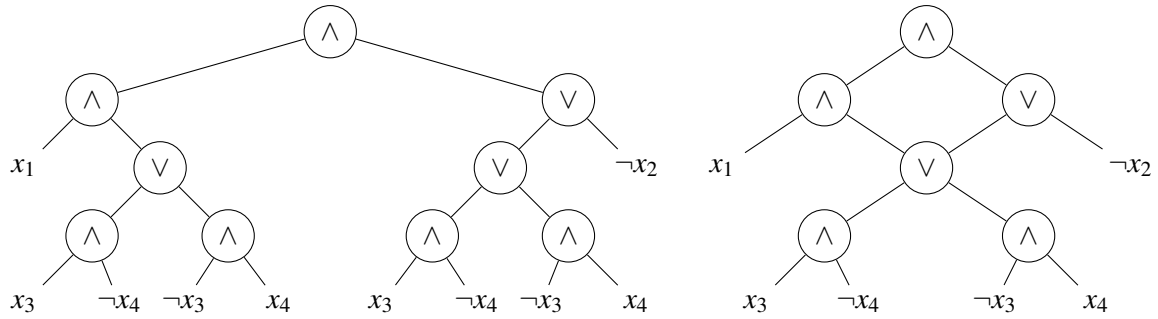


Figure 3.1: A tree (on the left) and circuit (on the right) NNF representation of the propositional (non-NNF) formula $(x_1 \wedge (x_3 \oplus x_4)) \wedge ((x_3 \oplus x_4) \vee \neg x_2)$. We see that the subformula induced by the $\vee$-node at the third level occurs twice, but in the circuit representation it is stored just once and then referenced two times.

Thus, the circuit representation of formulas with an arbitrary depth might be exponentially shorter than the tree representation of the same formula, but for formulas of bounded depth this is not the case.

For all languages and representations, we write $|\alpha|$ for the size of an formulas φ in the representation, namely the number of nodes and edges in the tree or DAG. For a set M we denote by $|M|$ the cardinality of M. Since we consider all formulas (and other syntactic expressions) in this work as graphs (including trees), we might refer to subformulas (resp. subexpressions) as (internal) *nodes* and *leaves*.

3.3 Background in Complexity Theory

In this section we briefly give some basics from complexity theory which will be sufficient to understand the complexity proofs in this work. We will refrain from giving formal definitions like in Arora and Barak (2009) because the details of the theory are not important for our results.

Definition 3.7. A *decision problem* is a class of questions (called *instances*) which require a "yes" or "no" answer. A decision problem is in the complexity class

- P if there exists an algorithm which outputs the answer in time polynomial in the size of the input (we say it is a *polynomial-time* algorithm),
- NP if the answer "yes" can be verified by a polynomial-time algorithm when provided a *certificate* whose size is polynomial in the size of the input,
- coNP if the answer "no" can be verified by a polynomial-time algorithm when provided a certificate whose size is polynomial in the size of the input,
- PSPACE if the answer can be computed by an algorithm that uses an amount of memory space which is polynomial in the size of the input.

An alternative definition for NP involves nondeterministic Turing machines, that is, machines which are the "best-possible guesser". Then NP is the set of problems solvable by a nondeterministic Turing machine in polynomial time, which explains the name (**N**ondeterministic **P**olynomial-time). Using this definition one can define NPSPACE to be the class of problems which can be solved by a nondeterministic Turing machine using a polynomial amount of space. Since a well-known implication from Savitch's theorem is that NPSPACE = PSPACE (Arora and Barak, 2009, Theorem 4.12), we won't introduce the notion of Turing machines which is otherwise not relevant for this work.

For a complexity class $C \in \{\mathsf{NP}, \mathsf{coNP}, \mathsf{PSPACE}\}$ we say that a decision problem S is *C-hard* if every decision problem T in C can be reduced to S in polynomial time, i.e. there is a polynomial-time algorithm which computes for an instance t of T an instance s of S such that the answer for s is "yes" if and only if it was "yes" for t. A C-hard problem is *C-complete* if it is in C and at the same time C-hard.

In this work, we will also use nonuniform complexity classes.

Definition 3.8. A *polynomial advice function* is a function a with the codomain $\{0,1\}^*$ (i.e. the set of all strings with entries 0 or 1) such that for two inputs x, y with $|x| = |y|$ (i.e. with the same length) it holds $a(x) = a(y)$ and the length of $a(x)$ is bounded by a polynomial in $|x|$. Let C be a complexity class. Then C/poly is the class of problems which are in C if the instances x are extended by a second input $a(x)$ with some polynomial advice function a.

Informally, for a decision problem S being in C/poly means that every instance s of S can be answered by some C-algorithm (e.g. nondetermnistic Turing machine for $C = \mathsf{NP}$), but the concrete algorithm is chosen depending on the size of s.

The simplest illustration of this concept is the complexity class P/poly of problems which are solvable in polynomial time if the input is extended by an advice string which depends on the size of the instance. In words this means that for the inputs of size n there exists a polynomial-time algorithm a_n solving the problem, and a_n can be different for each n, thus a problem S in P/poly can be answered by a family $\{a_n\}_{n\in\mathbb{N}}$ of polynomial-time algorithms.

Inclusions and Assumptions Some relations between complexity classes are well-known (Arora and Barak, 2009): for example, P is contained in $\mathsf{NP}, \mathsf{coNP}$ and P/poly, and NP and coNP themselves are contained in PSPACE. PSPACE itself is equal to $\mathsf{coPSPACE}$ (and thus $\mathsf{coPSPACE} = \mathsf{NPSPACE}$).

In our work we will use yet unproven assumptions especially about the nonuniform complexity classes P/poly, $\mathsf{NP}/\mathsf{poly}$ and $\mathsf{coNP}/\mathsf{poly}$. Although it is unproven, it is widely believed that all of the complexity classes defined in Definitions 3.7 and 3.8 are distinct: in particular, it is believed that $\mathsf{NP} \not\subseteq \mathsf{P}/\mathsf{poly}$ and $\mathsf{coNP} \not\subseteq \mathsf{NP}/\mathsf{poly}$. $\mathsf{NP} \subseteq \mathsf{P}/\mathsf{poly}$ would imply a collapse of the polynomial hierarchy at the second level (Karp-Lipton theorem) and $\mathsf{coNP} \subseteq \mathsf{NP}/\mathsf{poly}$ would imply a collapse at the third level (Yap, 1983) (for more details, and especially for the definition of the polynomial hierarchy, we refer to Arora and Barak (2009)). Especially we assume $\mathsf{PSPACE} \not\subseteq \mathsf{NP}/\mathsf{poly}$.

Hard problems For a decision problem S (like deciding whether an action is applicable in a state) we want to find the "smallest" complexity class C which contains it. Usually this means that S is among the hardest problems in C, i.e. it is C-complete. Thus a complexity result for some query which is harder than polynomial-time solvable normally consists of two parts: the membership and the completeness part. In the membership part we prove that the problem is indeed in C, and in the hardness part we give a polynomial-time reduction to S from some other problem T for which it is already known that it is C-hard. C-hardness of S then follows by transitivity of reductions. There are large lists of problems which are already proven to be complete (for example, many NP-complete problems are listed by Garey and Johnson (1979)), but we will usually deal with the few classic ones, which are worth defining at this point:

- Deciding the satisfiability of a 3-CNF is an NP-complete problem (known as 3-SAT),
- Deciding the unsatisfiability of a 3-CNF formula is a coNP-complete problem,
- Deciding the validity of a fully quantified Boolean formula with alternating quantifiers of the form $\exists x_1 : \forall x_2 : \exists x_3 : \ldots \forall x_n : \varphi$ with a 3-CNF φ is a PSPACE-complete problem.

These problems are still quite general, and we are going to use more specific versions in proofs.

Lemma 3.9. *Let $k \geq 3$, then deciding the satisfiability of a* 3-CNF φ *is polynomial-time reducible to deciding the satisfiability of another* 3-CNF ψ *with at most k occurences of each variable.*

PROOF. Let φ be a 3-CNF over $\{x_1, \ldots, x_n\}$. Let the variable x_i occur $k + j$ times in φ. We introduce the variables $y_i^1, \ldots, y_i^j$ (and for notational convenience we write y_i^0 for x_i) and for $m = 1, \ldots, j$ we replace the m-th occurence of x_i in φ with y_i^m, and then append the clauses $(y_i^{m-1} \vee \neg y_i^m)$ and $(\neg y_i^{m-1} \vee y_i^m)$ to the modified φ (the conjunction of these two clauses is equivalent to $y_i^{m-1} \leftrightarrow y_i^m$). We obtain ψ by doing

this for all x_i. This construction can be performed in linear time because for each "excessive" occurence we need to add a constant number of symbols. ϕ and ψ are equisatisfiable by construction, because a model of ϕ can be extended to a model of ψ by assigning the value of x_i to the $y_i^1, \ldots, y_i^j$, and for every assignment satisfying ψ the restriction to the x_i-variables is already a model of φ. □

Lemma 3.10. *Deciding the validity of fully quantified formulas* $\neg(\exists x_1 : \neg(\exists x_2 : \neg(\ldots \neg(\exists x_n : \varphi)\ldots)))$ *and* $\neg(\forall x_1 : \neg(\forall x_2 : \neg(\ldots \neg(\forall x_n : \varphi)\ldots)))$ *with* φ *being a* 3-CNF *or a* 3-DNF *is a* PSPACE*-complete problem.*

PROOF. First we observe that we can always introduce dummy variables y and z to obtain any desired sequence of operators, for example: $\exists x : \chi \equiv \forall y : \exists x : \forall z : \chi$ with $y, z \notin \mathrm{V}(\chi)$. Furthermore, for all formulas χ it holds $\exists x : \chi \equiv \neg(\forall x : \neg\chi)$ and $\forall x : \chi \equiv \neg(\exists x : \neg\chi)$.

We know that deciding whether $\exists x_1 : \forall x_2 : \exists x_3 : \ldots \forall x_n : \varphi$ with a 3-CNF φ is true is a PSPACE-complete problem, and by applying the above equivalences we can completely eliminate one sort of quantifier in a formula. For example: $\exists x_1 \forall x_2 \exists x_3 \ldots \varphi$ with a 3-CNF φ is equivalent to $\neg(\forall x_1 : \neg(\forall x_2 : \exists x_3 : \ldots \forall x_n : \varphi))$. In the same manner we continue for the subformula $\exists x_3 : \forall x_4 : \ldots \forall x_n : \varphi$, and repeat the procedure until there are no more existential quantifiers. Then the transformed formula equivalent to the original one is $\neg(\forall x_1 : \neg(\forall x_2 : \neg(\ldots \neg(\forall x_n : \varphi)\ldots)))$. Observe that φ itself is not negated here because the last variable x_n was universally quantified. If x_n is existentially quantified, after performing the above transformations we obtain $\neg(\forall x_1 : \neg(\forall x_2 : \neg(\ldots \neg(\forall x_n : \neg\varphi)\ldots)))$, and $\neg\varphi$ is equivalent to a 3-**DNF**.

Dually, by replacing the universal quantifiers with negated existential ones we can transform every fully quantified formula into the form $\neg(\exists x_1 : \neg(\exists x_2 : \neg(\ldots \neg(\exists x_n : \varphi)\ldots)))$. Since we can introduce dummy variables, we can always get an appropriate prefix of quantifiers such that after transforming the quantifiers using negations we obtain the desired form with only one sort of quantifiers and a 3-CNF or a 3-**DNF** matrix. □

CHAPTER 4

A Formal Framework for Comparing Action Languages

Contents

In this chapter we will formally define and illustrate the main notions of our study: states, actions, action languages and translations between them, and give the criteria which we will use to design our KC map. As we said in the introduction, these criteria can be roughly divided into three sorts: queries, which in our case will all be decision problems, whose complexity is of interest, transformations, where the output is a formula or an action description and where we are interested in the size of the output, and succinctness relations between languages.

4.1 Action Languages

The object of study of this work are the properties of action languages. The notion of an action language is similar to that of Boolean languages from Chapter 3, but a little more general. We recall that while in the original KC map all languages had the same interpretation and semantics and the only difference was the structure of the formulas, in our setting we will consider two different types of languages: the ones inspired by **STRIPS** and **PDDL**, which are "imperative" because changes of variable values are explicitly written in form of commands/variable assignments, and the "declarative" ones which specify relations between predecessors and successors and are based on Boolean formulas, and where variables can change their values if it is not explicitly prohibited. Recall that we study propositional representations.

4.1.1 States

Throughout our work we will consider a countable set of propositional *state variables* $\mathbb{P} := \{p_i \mid i \in \mathbb{N}\}$. Sometimes we will use other variables like $q_i, r_i, \ldots$, $x_i, y_i, \ldots$ or more sophisticated state variables like window_open for better clarity.

Let $P \subset \mathbb{P}$ be a nonempty 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. Analogously to logical formulas we write $\mathrm{V}(\alpha)$ for the set of variables occuring in an action description α; note that we will consider expressions involving both variables in $\mathbb{P}$ and variables not in $\mathbb{P}$, so in general we do *not* have $\mathrm{V}(\alpha) \subseteq \mathbb{P}$. If we write $s \models \varphi$ for a state s and formula φ we mean that the assignment corresponding to s is a model of φ.

4.1.2 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, in contrast with deterministic actions (which map states to states), and 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 4.1 (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 and P is called the *scope* of a.

Note that $a(s)$ is defined for all states s. We say that a is *applicable* in s if and only if $a(s) \neq \emptyset$. Every action a can be identified with a binary *transition relation* $\|a\|$ on states which is defined as $\|a\| := \{(s, s') \mid s' \in a(s)\}$. Elements of $\|a\|$ are called *state transitions*.

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

Definition 4.2 (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 from $L \times 2^{\mathbb{P}}$ to the set of all actions such that, when defined on $\alpha \in L$ and $P \subset \mathbb{P}$, $I(\alpha, P)$ is a P-action.

If $\langle L, I\rangle$ is an action language and $L' \subseteq L$ then we call $\langle L', I|_{L' \times 2^{\mathbb{P}}}\rangle$ a *sublanguage* of $\langle L, I\rangle$ and $\langle L, I\rangle$ a *superlanguage* of $\langle L', I|_{L' \times 2^{\mathbb{P}}}\rangle$. An action language $\langle L, I\rangle$ is *complete*, if for every P and every P-action a there exists an $\alpha \in L$ with $I(\alpha, P) = a$.

We call the expressions in L *action descriptions*, and call I the *interpretation function* of the language. In this work action descriptions will be generated by a simple grammar similar to that of propositional logic, which will be given in Bachus-Naur form. Complex expressions will be generated from *atomic actions* using *connectives* (for example, the **O-PDDL** action description $+p \,\&\, -q$ consists of the atomic actions $+p$ and $-q$ which are connected via the &-connective). An action language then will be either directly defined as the set of all expressions generated by a grammar, or as a subset of this set satisfying some additional criteria. Sometimes we will use the letter L to denote a generic action language and not just its action descriptions. In such cases we will use I_L to denote its interpretation function.

Observe that those sets P such that $I(\alpha, P)$ is defined are *a priori* not related to $\mathrm{V}(\alpha)$: α may involve extra symbols (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. As an illustration of the role of P, for the NNF action theory $\alpha := (p \wedge \neg p') \vee (\neg p \wedge p')$, which can be read "switch the value of p", the semantics of such theories (defined in Section 5.4) implies that all other variables take an arbitrary value after the action. Hence the argument P in $I(\alpha, P)$ specifies which variables we consider and hence, which variables are "reinitialized" in this manner. For example, if $P = \{p, q\}$ then both states $s' = \{p\}$ and $s'' = \{p, q\}$ are $I(\alpha, P)$-successors of $s = \emptyset$, and $P \not\subseteq \mathrm{V}(\alpha)$. This example also illustrates the use of extra symbols: $\mathrm{V}(\alpha) = \{p, p'\} \not\subseteq \mathbb{P}$.

If the language $\langle L, I\rangle$ and the set 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. We might also say "action α", meaning the action described by the action description α.

For an action description α we write $\|\alpha\|$ in the meaning of $\|I(\alpha, P)\|$ if I is clear from the context and P is fixed. Recall that for logical formulas φ the notation $\|\varphi\|$ stood for the set of models of φ, and analogously, $\|\alpha\|$ can be seen a set of models of α where each model is a state transition (s, s'). For actions α and β in a fixed language $\langle L, I\rangle$, we write $\alpha \equiv \beta$ if for all P such that $I(\alpha, P)$ and $I(\beta, P)$ are defined, $\|I(\alpha, P)\| = \|I(\beta, P)\|$ holds.

As a general rule, we will use notation $s, t, \ldots$ for states, $a, b, \ldots$ for actions and $\alpha, \beta, \ldots$ for action descriptions.

4.1.3 Translations

Definition 4.3 (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 $f\colon L_1 \times 2^{\mathbb{P}} \to L_2$ 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. For instance, considering $\alpha := (p \wedge \neg p') \vee (\neg p \wedge p')$ again and the set $P := \{p, q\}$, it can be translated into the following expression in the nondeterministic variant of PDDL defined by Bertoli et al. (2003):

```
(and (when (p) (not p)) (when (not p) (p)) (oneof q (not q)))
```

Again, when P is clear from the context, we write $f(\alpha)$ for $f(\alpha, P)$.

We insist that a translation is *not* allowed to introduce auxiliary variables. For instance, a translation of a P-action α could not set a fresh proposition je_α representing "α has just been executed" in the resulting state s', since the translation would then be from a P-action to a $P \cup \{\mathsf{je}_\alpha\}$-action. This is in contrast with many studies of compilation for planning (see Chapter 2). The reason why we focus on this setting is that we are interested in translating the *actions*, and not only the solutions to a planning problem. Our strict notion guarantees that translating the actions of a domain will preserve many properties: the existence of solution plans of course, but also their length, their number, and measures of complexity like many notions of width (Palacios and Geffner, 2009; Lipovetzky and Geffner, 2012, for instance).

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. We emphasize that the computational process is not part of the definition of "polynomial-size", i.e. the existence of a polynomial-size translation f does not guarantee that f can be computed using a polynomial amount of space. This remark is a general one, because in our work all positive results will be about an existence of a polynomial-time translation, and all negative results – about the non-existence of polynomial-size translations.

We note that the cardinality of P needs to be taken into account for the notions of polynomial-time and polynomial-size translations. For instance, we will define a translation which is polynomial-time in $|\alpha| + |P|$ but in the worst case unbounded in only $|\alpha|$ (Proposition 6.2, page 38). ¸

Representations

We recall from Subsection 3.2.4 that the size of a formula depends on the representation. This notion can be generalized to action descriptions in the obvious manner: we speak of a *tree representation* of action descriptions if they are represented by the underlying syntactic tree, and of the *circuit representation* if the representing graph is just a directed acyclic rooted graph. Both action descriptions and Boolean formulas will be referred to as *expressions*. Since the grammars of our languages will use propositional NNF formulas as parts of action descriptions, the chosen representation of action descriptions will also affect the representation of the contained Boolean formulas. Analogously to formulas we will denote by $|\alpha|$ the size of the expression α if the chosen representation is clear from the context.

We recall that the space efficiency of representations may significantly (superpolynomially) differ only if the depth of the expressions is not bounded. Therefore we will give complexity and succinctness results for both representations of languages L which allow for unbounded depth (except for Chapter 9, where we emphasize that we study only circuit representations). These representations will be denoted by $L^{\mathbf{T}}$ and $L^{\mathbf{C}}$. If we make a statement about a language without specifying the representation, this means that the statement holds for both representations.

4.2 Criteria for Comparison

In this section we introduce the criteria which we will use to compare the action languages. In knowledge compilation we aim at identifying representations which enable storing information efficiently while

allowing to answer queries and perform transformations in reasonable time. Therefore we investigate the complexity of queries and transformations most natural in planning and the relative succinctness of the action descriptions (Darwiche and Marquis, 2002). Since we said in the introduction that queries are the varying parts of the input to a computational problem, we should probably use the term *query task* to denote the decision problems whose complexity is the key criterion which we study here. But since in knowledge compilation literature the word query is often used to denote the query task, too, we will say "query" instead of "query task" in the rest of this work for the sake of simplicity.

4.2.1 Queries

Applicability and Entailment

Remark 4.4. We note that all queries defined below take as input an action description α and at least one state s, i.e. the question asked is usually "is it true for α and s that ...?". By universally quantifying over s we obtain a derived query "is ... true for α and for all s (all s satisfying some property)?". In this work we do not concentrate on these derived queries, but we note that their complexity shouldn't be very hard to derive from the complexity of the "original" queries.

A large body of planners frame planning problems as search in the space of reachable states or belief states (i.e., sets of states) or, equivalently, in the space of partial plans. Most essential to such approaches is the ability to answer two queries at a given node/partial plan: (1) what actions are applicable, or equivalently, what are the successors of this node, and (2) whether the current partial plan achieves the goal (whether this is a terminal node). This motivates us to introduce two main queries to an action language.

In this section let $\langle L, I\rangle$ be a fixed action language, α, $\alpha_1, \ldots, \alpha_k$ be action descriptions in L, $P \subset \mathbb{P}$ be a set of variables such that $I(\alpha, P)$ and $I(\alpha_1, P), \ldots, I(\alpha_k, P)$ are defined, and s, s' be P-states.

When given action descriptions $\alpha_1, \ldots, \alpha_k$ and a state s, we write $(\alpha_1; \ldots; \alpha_k)(s)$ for the set of states s' such that there are $s_0, s_1, \ldots, s_k$ with $s = s_0$, $s_i \in \alpha_i(s_{i-1})$ for $i = 1, \ldots, k$, and $s_k = s'$.

Definition 4.5 (IS-APPLIC)**.** The query IS-APPLIC asks for given α, P, s, whether $\alpha(s)$ is non-empty.

Definition 4.6 (ENTAILS)**.** The query ENTAILS asks for given $\alpha_1, \ldots, \alpha_k$, P, s, and an **NNF** formula φ over P, whether $s' \models \varphi$ for all $s' \in (\alpha_1; \ldots; \alpha_k)(s)$.

Note that if for $(\alpha_1; \ldots; \alpha_k)$ and s there is no such chain $s_0, \ldots, s_k$ as defined above then it automatically entails any formula φ, especially any unsatisfiable formula, for instance $\varphi := p \wedge \neg p \equiv \bot$ for an arbitrary p.

These two queries together define *belief tracking*, whose centrality to planning has been put forward in the more general case of partially observable domains (Bonet and Geffner, 2014; Brafman and Shani, 2016). Recall from Section 3.1 that in order to verify for a sequence of actions that it is a strong linear plan which reaches a goal we need to ensure that each action a_i is applicable after a_{i-1} and that the goal in the end is ensured, and this is essentially belief tracking. We wish to emphasize that a more general formulation of these queries would involve a *belief state* rather than a state s, and the questions would be whether all states in the belief state allow the action to be applied or whether they all satisfy the goal. Since considering various representations of such belief states is out of the scope of this work (see Remark 4.4), we stick to a single state s. However, upper bounds can already be derived from our results; for instance, if for a language $\langle L, I\rangle$ the query IS-APPLIC is polynomial-time solvable for a single input state s, then the more general query is in coNP for any "reasonable" representation of belief states (i.e., representations of S for which $s \in S$ can be decided efficiently): indeed, it suffices to guess such an $s \in S$ that witnesses that the action is not applicable.

We also wish to emphasize that complexity results about these queries not only apply to planners which directly work with partial plans, as Conformant-FF (Hoffmann and Brafman, 2006). They also apply to planners in which an explicit belief state is maintained at each node (Cimatti and Roveri, 2000), even if such belief states are not explicit in the definitions of the queries. For instance, if ENTAILS is coNP-complete for some action language, then this means that entailment can be decided efficiently from

such an explicit representation only if that representation is (in the worst case) exponential in $\alpha_1, \ldots, \alpha_k$, P, s, otherwise maintaining the belief state and checking whether the belief state after the last execution always satisfies φ would give a polynomial-time algorithm.

Another important point is that in many practical languages, actions come with an *explicit* precondition (Nebel, 2000; Nau, 2007, for instance), namely, a formula φ such that the action is applicable at s if and only if s satisfies φ. In that case, query IS-APPLIC is polynomial-time for any "reasonable" representation of φ. Such practical languages are essentially encompassed by our language **O-PDDL** (Chapter 5), however, for **E-PDDL** and **NNFAT** (Chapter 5) the picture is different: indeed, as we will show later (Chapter 6), representing preconditions explicitly might be (exponentially) less space-efficient than representing them implicitly.

The complexities of the two queries are related as follows.

Lemma 4.7. *For any action language, the complement of* Is-Applic *is polynomial-time reducible to* Entails*, even if the input sequence of* Entails *is restricted to contain only one action.*

PROOF. An action described by α is *not* applicable in s (i.e., it has no successors in s) if and only if all α-successors s' of s satisfy $\bot \equiv p \wedge \neg p$ for an arbitrary p, i.e. if and only if α entails $\varphi \coloneqq \bot$ in s. □

Successorship

IS-APPLIC and ENTAILS, although being practically relevant, are not the most fundamental ones. Both IS-APPLIC and ENTAILS ask only about a part of the information captured by an action description. For example, knowing the answers for IS-APPLIC of α for all s does not allow to uniquely determine the action described by α. Therefore we define another query, which, while not used directly in practice, is still very helpful for our studies, because it is very basic and can be seen as model checking for action languages.

Definition 4.8 (IS-SUCC)**.** IS-SUCC: given α, P, s, s', decide whether s' is an α-successor of s.

We will see that knowing the complexity of IS-SUCC is usually sufficient to easily deduce the complexity of other queries.

Other queries

Definition 4.9. An action a is said to be *deterministic* in state s if $|a(s)| \leq 1$.

An action a is said to be *positively monotone* in state s if for all $s' \in a(s)$ it holds that $s \subseteq s'$. Dually, an action is *negatively monotone* in s if $s' \subseteq s$ for all $s' \in a(s)$.

In words, an action a is deterministic in s if s has at most one a-successor. An action is positively monotone in s, if it can only change variable values to $\top$ (to $\bot$ if it is negatively monotone) for each state transition from s. We emphasize that these definitions depend on a given state, e.g. an action can be deterministic in one state and not deterministic in another.

The following questions are quite natural for nondeterministic planning. For instance, knowing that all actions are deterministic is useful because one can first compile the action descriptions into a formalism for classical planning and then apply all the numerous algorithms implemented in this area. And checking monotony could be useful with respect to computing the delete-free relaxation, which plays an important role in classical planning (Geffner and Bonet, 2013, page 28).

Definition 4.10. Let $\langle L, I \rangle$ be a fixed action language, α an action description in L, $P \subset \mathbb{P}$ be a set of variables such that $I(\alpha, P)$ is defined, and s be a P-state. We consider the following queries:

- IS-DET: given α, P, s, decide whether α is deterministic in s.
- IS-MON: given α, P, s, decide whether α is positively monotone in s.

For IS-MON we can also consider the dual variant, asking whether an action is negatively monotone in a given state. We study both of these queries, but we use the same name for them, because the reasoning is almost always the same for symmetry reasons. At one place in our work (we will explicitly state, where) there is a little asymmetry, which doesn't affect the results, though.

The next query may seem a little artificial, and in this work we consider it only to use it in succinctness proofs later.

Definition 4.11. The query ST asks for a given action description α, state variables P and $s \subseteq P$ with $I(\alpha, P)$ defined, whether $s \in \alpha(s)$.

However, ST might still be interesting on its own because answering this query is easy for many of the languages we are going to study, and a positive answer to ST means that we are not progressing towards the goal if our policy is using α in s.

Remark 4.12. Obviously, since trees are a special case of circuits, membership complexity results for queries for the circuit representation also hold for the tree representation of a language, and hardness results for trees also apply to circuits.

4.2.2 Succinctness

When choosing an internal representation of actions for a planner, we might e.g. want the complexity of IS-APPLIC not to be harder than NP and that of ENTAILS not to be harder than coNP. Since there are several languages with these properties, a complementary criterion is to choose a representation which consumes the least memory, i.e. can express the actions more concisely. This motivates the next definition (Darwiche and Marquis, 2002); recall that by a complete action language we mean one in which any nondeterministic action has at least one representation. We will see later that this is the case of all languages studied in this work.

Definition 4.13 (succinctness)**.** A complete action language L_1 is *at least as succinct* as a complete action language L_2 (written $L_1 \preceq L_2$) if there exists a polynomial-size translation from L_2 into L_1. If in addition there is no polynomial-size translation from L_1 into L_2 then L_1 is *strictly more succinct* than L_2, written $L_1 \prec L_2$. If neither $L_1 \preceq L_2$ nor $L_2 \preceq L_1$, we say that they are *succinctness-incomparable*.

Clearly, if L_2 is translatable into L_1 in polynomial *time* then L_1 is at least as succinct as L_2.

Apart from choosing internal representations, succinctness also tells to what extent a planner designed for input descriptions of actions in a given language L, can be used to handle descriptions in another language L'. Indeed, if L is at least as succinct as L', then this means that an input in L' can be translated into one in L, for use by the planner, without an explosion in size. For instance, our results will show that a planner designed for $L = \textbf{E-PDDL}$ can be used with action descriptions in $L' = \textbf{NNFAT}$, but not necessarily vice-versa. In our work we will only compare languages in terms of succinctness for the same representation (i.e. we will not compare languages with the tree representations, of the form $L_1^{\mathbf{T}}$, to languages with the circuit representation, of the form $L_2^{\mathbf{C}}$).

4.2.3 Transformations

The third main object of interest in knowledge compilation is the complexity of performing transformations of given expressions. Transformations can be used by planners for reasoning about several actions at once; for instance, in the probabilistic setting, one version of SPUDD (Hoey et al., 1999) starts by computing a disjunction of all available actions, so that the current belief state is progressed only once (by all actions) at each subsequent step.

A natural example of a transformation in knowledge compilation is computing the CNF representation of $\neg\varphi$ for a CNF formula φ. In our article the main difference between queries and transformations will be the answer we are interested in. For queries we are looking for the membership and hardness results (e.g. showing that IS-SUCC in a language is both NP-hard and in NP), whereas for transformations we are interested whether the representation $f(\alpha)$ for some transformation has size bounded by some polynomial. In this thesis we concentrate on the complexity of the following four transformations.

Definition 4.14 (Transformations)**.** Let $\langle L, I\rangle$ be a fixed action language, α, α_1 and α_2 be action descriptions in L, $P \subset \mathbb{P}$ be a set of variables such that $I(\alpha, P)$ and $I(\alpha_1, P), I(\alpha_2, P)$ are defined, and s and s' be P-states.

- EXTRACT-PRECOND: given α and P, compute an NNF formula φ such that for all s: $\alpha(s) \neq \emptyset \iff s \models \varphi$.
- SEQUENCE: given $\alpha_1, \alpha_2 \in L$, compute an action description $\beta \in L$ such that for all s: $\beta(s) = \{s' \mid \exists t$ such that $t \in \alpha_1(s)$ and $s' \in \alpha_2(t)\}$.
- CHOICE: given $\alpha_1, \alpha_2 \in L$, compute an action description $\beta \in L$ such that for all s: $\beta(s) = \alpha_1(s) \cup \alpha_2(s)$.
- NEGATION: given $\alpha \in L$, compute an action description $\beta \in L$ such that for all s: $\beta(s) = \{s' \mid s' \notin \alpha(s)\}$.

A transformation is said to be *polynomial-time* if the output action description β (resp. formula φ) can be computed in time polynomial in the size of the inputs, including the scope P. It is said to be *polynomial-size* if the size of the output is bounded by a polynomial in the size of the inputs.

We insist that for the three latter transformations it is required that the output is in the same language as the input description(s), because they are about transforming action descriptions into other action descriptions, whereas EXTRACT-PRECOND is rather about extracting information from an action description. This relates it to queries, but we still see it as a transformation because we are mainly interested whether representing a precondition explicitly is possible without a loss of succinctness, while for queries we are interested in the time complexity. Another reason to treat it separately from queries is that in our work queries always can be formulated in a simple form which requires a state as input, whereas for EXTRACT-PRECOND there is no such natural formulation, it's always performed with only an action description as input. The other direction of EXTRACT-PRECOND, "merging" the precondition and the description of effects into one expression, is always possible in linear time for the languages which we study, as we will see later.

As we already discussed in Subsection 4.2.1, the tractability of EXTRACT-PRECOND matters if we want applicability checking to boil down to satisfaction of a given NNF formula (hence in polynomial time). The other transformations are useful for planners which reason about higher-level actions. For example, a language which supports polynomial-size SEQUENCE is a good candidate for a target language if the action specification involves expressions similar to Hierarchical Task Networks (Georgievski and Aiello, 2015) or macro operators (Botea et al., 2005). Similarly, if the actions are specified in a dynamic logic which allows for action negation (Broersen, 2004), a target language supporting polynomial-size NEGATION is a good choice if it is computationally more efficient from some point of view. CHOICE is an obvious transformation to consider when working with nondeterminism.

Somewhat counterintuitive about transformations is that, contrary to queries where we can often deduce information about complexity from the sub/superlanguage relation (e.g., IS-SUCC is NP-hard for **E-PDDL**$_{\text{not}}$ since it is NP-hard already for its sublanguage **E-PDDL**), it might be that a transformation is easy for a language L, hard for a superlanguage L' of L, and again easy for a superlanguage L'' of L'.

The last remark we make in this section will be about SEQUENCE: the action β satisfying $\beta(s) = \{s' \mid \exists t$ such that $t \in \alpha_1(s)$ and $s' \in \alpha_2(t)\}$ for all s will be denoted $\alpha_1 ; \alpha_2$ because later in our work we will introduce the sequence connective ;.

Part II

Basic Languages

CHAPTER 5

Minimally Complete Languages

Contents

In this chapter we formally define three action languages, which are minimally complete in the sense that removing any of the operators from the grammar makes the resulting language incapable of expressing all possible nondeterministic actions. Two of these languages are imperative and one is declarative (see Chapter 4).

In Chapter 2 we already mentioned that Rintanen defined a formalism which allows to represent every nondeterministic action (actually this formalism is in some sense much more general than ours, for example, nondeterministic choices there come together with probabilities). However, all effect (effects there roughly correspond to our action descriptions) descriptions there are assumed to satisfy Property 2 which states, that in each state only one occurence of a variable can contribute to the determination of a successor. This property is "semantic" in the sense that cheking it requires computational effort to find out whether the given expression is indeed a valid effect. For example, the effect $(\top \rhd p) \wedge (\top \rhd \neg p)$ (expressed with the notation from there) is formed according to the rules of the grammar, but it does not satisfy Property 2, because the variable p is assigned twice, by the left and the right "conjunct".

We want to define a semantics which provides an interpretation to every well-formed formula. To do so we choose two natural ways to relax this assumption. The two (imperative) languages obtained in this way will be called **O-PDDL** and **E-PDDL**, because they can be seen as abstract variants of a nondeterministic extension (Bertoli et al., 2003) of the Planning Domain Definition Language **PDDL** (McDermott, 1998). Since the different solutions can be expressed by the semantics of just one connective, we will use the same notation for everything but these connectives, i.e. the only difference between the grammars of **O-PDDL** and **E-PDDL** will be the connective of the *parallel composition* (& and $\sqcap$, respectively). We found the notation of **DL-PA** (Balbiani et al., 2013) and **DL-PPA** (Herzig et al., 2019) (especially $+p$ and $-p$ for atomic actions and $\sqcap$ for the parallel composition) quite intuitive and therefore we use it instead of the notation more typical to the planning literature, but we decided to keep the notation for the conditional execution $\rhd$ because its semantics is quite different from that of the formula test $\varphi?$, which is specific to dynamic logics. The intuitiveness is not the only reason, though, later we will again refer to **DL-PPA** in Chapter 8.

The third (declarative) language **NNFAT** is that of NNF action theories, which is a representation of actions via Boolean formulas in negated normal form.

We will argue that these languages are indeed minimally complete and thus a good starting point for our study. This chapter only contains definitions, explanations and illustrations, and aims at helping the reader to get familiar with the notation and "style" of the technical part of this work.

We will illustrate the action languages by expressing the following running examples.

Example 5.1. *Consider injecting a drug for anesthesia to a patient and then operating her. The whole process can be described as an action "operating the patient", and there are several possible results: the anesthesia might work well, or not, and the surgery itself can be successful or not. If the surgery is not successful, the patient dies, thus having no pain. If the surgery is successful, the patient is supposed to have pain, which, however, can be suppressed by the anesthesia. This situation can be described via the variables* $P = \{\text{no\_pain}, \text{dead}, \text{cured}\}$.

Example 5.2. *Consider a self-driving car on a right highway lane with boundary fences (so that there is no way to get off the road). Suddenly there is a child on the road directly in front of the car. The car is programmed to always avoid deaths of people outside the car, which in this case can be achieved by turning left to another lane or by stopping abruptly (or both). On the other hand, it is programmed to prevent severe damage for the passengers. Therefore, when going left in an emergency situation the car necessarily launches the airbags to minimize potential damage from the cars on the left lane, and the other option of saving the passengers is simply staying on the current lane (i.e. not going left). The situation can be modeled with the variables* $P = \{\text{child\_on\_road}, \text{left}, \text{moving}, \text{airbag}\}$.

5.1 Effects

In order to define the interpretation functions for the imperative languages we first define the following notion, similar to that of add- and del-lists of STRIPS.

Definition 5.3 (effect)**.** An *effect* over a set of variables $P \subset \mathbb{P}$ is an ordered pair $\langle Q^+, Q^- \rangle$ with $Q^+, Q^- \subseteq P$ and $Q^+ \cap Q^- = \emptyset$. The set Q^+ (resp. Q^-) is called a *positive* (resp. *negative*) effect. An effect $\langle A^+, A^- \rangle$ is a *subeffect* of an effect $\langle Q^+, Q^- \rangle$ iff $A^+ \subseteq Q^+$ and $A^- \subseteq Q^-$.

We emphasize that the positive and negative parts of subeffects are disjoint by definition.

In the following we are going to define the set $\mathrm{E}(\alpha, P, s)$ of *explicit effects of* α *in* s for **O-PDDL**, **E-PDDL** and their extensions and restrictions, but we already use it to define the interpretation functions, which simply formalize the fact that variables not explicitly set by the action retain their value. For all action descriptions α in a language L and sets of variables P with $\mathrm{V}(\alpha) \subseteq P \subset \mathbb{P}$ such that $\mathrm{E}(\alpha, P, s)$ is defined, we set:

$$\forall s \subseteq P\colon\ I_L(\alpha, P)(s) \coloneqq \{(s \setminus Q^-) \cup Q^+ \mid \langle Q^+, Q^- \rangle \in \mathrm{E}(\alpha, P, s)\}$$

An effect $\langle Q^+, Q^- \rangle$ is said to *witness* a transition (s, s') if $(s \setminus Q^-) \cup Q^+ = s'$.

We emphasize that for a transition (s, s') there may exist several effects witnessing it. For example, given $P \coloneqq \{p_1, p_2\}$, the two effects $\langle \{p_2\}, \emptyset \rangle$ and $\langle \{p_1, p_2\}, \emptyset \rangle\}$ cause the same transition from $s \coloneqq \{p_1\}$ to $s' \coloneqq \{p_1, p_2\}$. Note that the second effect "reassigns" p_1 to the same value. In fact, it is easy to see that for two given P-states s, s', the effects which witness (s, s') are all $\langle Q^+, Q^- \rangle$ satisfying $s' \setminus s \subseteq Q^+ \subseteq s'$ and $s \setminus s' \subseteq Q^- \subseteq P \setminus s'$. This motivates the following definition.

Definition 5.4 (minimal, maximal effect)**.** An effect $\langle Q^+, Q^- \rangle$ is called *minimal in* s if $Q^+ \cap s = \emptyset$ and $Q^- \subseteq s$ hold (i.e. no positive or negative effect is useless in s), and *P-maximal* if $Q^+ \cup Q^- = P$ holds.

In words, an effect $\langle Q^+, Q^- \rangle$ is minimal in s if every variable which occurs in Q^+ or Q^- indeed changes the value during the transition defined by this effect in s, and it is P-maximal if every variable in P is "assigned" by either the positive or the negative effect during the transition.

5.2 Nondeterministic PDDL

We start with the well-known *planning domain definition 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 we abstract away from the precise syntax of the specification language. Note that **PDDL** enables specifications at the relational level, but we consider grounded specifications, since this thesis focuses on the propositional setting.

We first define the syntax of **O-PDDL**. The "O" in our notation means "original", because later in Subsection 5.3 we will consider an alternative language with a slightly different semantics for simultaneous execution, which we will denote by **E-PDDL**. In the same time the "O" stands for "overriding" because it is the language where "addition overrides deletion".

Definition 5.5 (**O-PDDL**)**.** An **O-PDDL** *action description* is an expression α generated by the grammar

$$\alpha ::= +p \mid -p \mid \varphi \rhd \alpha \mid (\alpha \cup \alpha) \mid \alpha \,\&\, \alpha \mid \bot$$

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

Intuitively,

- $+p$ (resp. $-p$) is the action which sets p true (resp. false),
- $\rhd$ denotes conditional execution,
- $\cup$ denotes (exclusive) nondeterministic choice,
- & denotes simultaneous execution, where addition overrides deletion,
- $\bot$ denotes failure,

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. $+p$ and $-p$ are called *atomic actions*. Later we will often write $\pm p$ meaning "$+p$ or $-p$".

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

$$\texttt{when}\ x\ \texttt{(oneof}\ y\ \texttt{(and (not}\ y\texttt{)}\ z\texttt{))}$$

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

Intuitively, a state transition via an **O-PDDL** action can be seen as an *execution* of the action, which is a top-down execution of the nodes according to their semantics. We execute a $\cup$-node by choosing the left or the right side of it, and then executing in parallel the left and the right side of each &-node which has been chosen. In order to define the interpretation function of **O-PDDL**, we need to define how to execute two effects simultaneously (via &).

Recall that by definition, the positive and negative part of an effect do not intersect. The semantics of & will define how such effects are associated to expressions such as $+p \,\&\, +q \,\&\, -p$.

The effects $\mathrm{E}(\alpha, P, s)$ of **O-PDDL** action descriptions, and thus the semantics of **O-PDDL**, are defined as follows:

- $\mathrm{E}(\bot, P, s) = \emptyset$,
- $\mathrm{E}(+p, P, s) := \{\langle \{p\}, \emptyset \rangle\}$, and $\mathrm{E}(-p, P, s) := \{\langle \emptyset, \{p\} \rangle\}$,
- $\mathrm{E}(\varphi \rhd \alpha, P, s) := \begin{cases} \mathrm{E}(\alpha, P, s) & \text{if } s \models \varphi, \\ \{\langle \emptyset, \emptyset \rangle\} & \text{otherwise,} \end{cases}$

- $\mathrm{E}(\alpha \cup \beta, P, s) \coloneqq \mathrm{E}(\alpha, P, s) \cup \mathrm{E}(\beta, P, s)$,
- $\mathrm{E}(\alpha \,\&\, \beta, P, s) \coloneqq \left\{ \langle Q_\alpha^+ \cup Q_\beta^+, (Q_\alpha^- \cup Q_\beta^-) \setminus (Q_\alpha^+ \cup Q_\beta^+) \rangle \;\middle|\; \begin{array}{l} \langle Q_\alpha^+, Q_\alpha^- \rangle \in \mathrm{E}(\alpha, P, s), \\ \langle Q_\beta^+, Q_\beta^- \rangle \in \mathrm{E}(\beta, P, s) \end{array} \right\}$

We emphasize here that the failure symbol $\bot$ is a necessary construct to ensure completeness of the language. If $\bot$ was not part of the language, actions would always be applicable because every atomic action ($\pm p$) has at least one effect and combinations of two actions having at least one effect each result in an action having at least one effect, too. We also highlight that $\rhd$ is not necessarily expressing a precondition. If φ is not satisfied in a state s then $\varphi \rhd \alpha$ simply modifies no variable in s. If we want to express that φ is a precondition for α being applicable we need to write $(\varphi \rhd \alpha) \sqcap (\neg\varphi \rhd \bot)$. On the other hand, the action ε which has only the empty effect $\langle \emptyset, \emptyset \rangle$ in all states can be expressed via $(p \wedge \neg p) \rhd +p$ (or any other expression with a contradiction as condition), and therefore we do not include it into the grammar, in order to keep it minimal.

Example 5.6. *The running example 5.1 is naturally expressed as follows:*

$$\text{surgery} \coloneqq \left(\begin{array}{c} \overbrace{(+\text{no\_pain} \cup \varepsilon)}^{\textit{effect of anesthesia}} \\ \,\&\, \underbrace{((-\text{no\_pain} \,\&\, +\text{cured}) \cup (+\text{no\_pain} \,\&\, +\text{dead}))}_{\textit{effect of surgery}} \end{array} \right)$$

The description is natural because it can be written directly, decomposed into the effect of the anesthesia and the effect of the surgery. If the anesthesia works, its effect $+$no_pain *"overrides" the effect* $-$no_pain *of the surgery itself. The variables keep their value if not assigned in this manner explicitly, so e.g. if the patient was dead before, he remains dead afterwards, because there is no* $-$dead *assignment in the action description.*

The other one, example 5.2, is not that natural to express with **O-PDDL** *because it needs to explicitly list all possible consistent combinations of decisions which save the people on the road with decisions made to save the passengers:*

$$\text{save\_all} \coloneqq \text{child\_on\_road} \rhd \left(\begin{array}{l} (+\text{left} \,\&\, +\text{airbag}) \\ \cup (+\text{left} \,\&\, +\text{airbag} \,\&\, -\text{moving}) \\ \cup (-\text{left} \,\&\, -\text{moving}) \end{array} \right)$$

We emphasize that the variables which are not assigned explicitly in the action description retain their value of the state where they are applied. For example, the action of saving the people does not affect the position of the people on the road, so they stay on the road. The $-$left *in the last line can be left out since it is assumed that we are already on the right lane.*

Proposition 5.7. **O-PDDL** *is indeed minimally complete, i.e. excluding any of its constructs from the grammar makes it unable to express all nondeterministic actions.*

PROOF. Excluding e.g. $-p$ makes the language uncapable of expressing the transition $\{p\} \to \emptyset$, and for the same reason we cannot exclude $+p$. Without the condition operator $\rhd$ the language can only express actions which are either applicable everywhere or nowhere. Actions defined without the choice operator $\cup$ are necessarily deterministic, and expressions without & cannot describe actions which have a transition where two or more variables change their value. Finally, $\bot$ is necessary to allow expressing actions which are not always applicable. □

Note that explicit effects (of subactions) matter only for &, in the sense that for other constructs, the transition relation $\|\alpha\|$ of α only depends on the transition relation of its subactions. Still the effects are necessary for the semantics because of the intended interpretation of &. For example, we want to capture the property that for $P = \{p\}$, $\alpha = p \rhd +p$ and $\alpha' = \varepsilon$ it holds $(\alpha \,\&\, -p)(\{p\}) = \{\{p\}\}$ and $(\alpha' \,\&\, -p)(\{p\}) = \{\emptyset\}$. Therefore in general it can happen that $\alpha \equiv \alpha'$, but $\alpha \,\&\, \beta \not\equiv \alpha' \,\&\, \beta$ and hence it is not possible to define the semantics of & using only the transition relation.

5.3 Egalitarian PDDL

The operator & in **O-PDDL** has an asymmetry with respect to positive and negative assignments. This asymmetry, on the one hand, makes non-applicability more explicit due to the mandatory failure symbol, but on the other hand this has some unnatural implications; for instance, if we want to rename the variable no_pain into a variable pain, simply replacing $+$no_pain by $-$pain and $-$no_pain by $+$pain in Example 5.6 results in a different action, since this time $+$pain has priority over $-$pain. Therefore we introduce a language in which the operator & is replaced by an operator $\sqcap$ which treats $+p$ and $-p$ literals symmetrically; as a consequence, executing $+p$ and $-p$ simultaneously will not be possible at all, resulting in the action with no transition. Another justification for this operator is for modelling actions which are composed of effects occurring in parallel (via different subprocesses); whenever the subprocesses need to set the value of a variable to different values, the action fails. This is indeed the view of the Dynamic Logic of Parallel Propositional Assignments (Herzig et al., 2019), from which we borrow the operator $\sqcap$.

We denote the resulting language by **E-PDDL**, with "E" standing for "egalitarian".

Definition 5.8 (E-PDDL). An **E-PDDL** *action description* is an expression α generated by the grammar

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

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

For the interpretation function, we need to define an additional property for effects: we say that two effects $e_1 := \langle Q_1^+, Q_1^- \rangle$, $e_2 := \langle Q_2^+, Q_2^- \rangle$ *agree* or *are (mutually) consistent* if $Q_1^+ \cap Q_2^- = Q_2^+ \cap Q_1^- = \emptyset$ holds; in other words, if one effect assigns $\top$ to a variable, the other does not assign $\bot$ to it. Then the *parallel combination* of e_1 and e_2 is defined to be the effect $\langle Q_1^+ \cup Q_2^+, Q_1^- \cup Q_2^- \rangle$. It can be interpreted as simultaneously executing the assignments of e_1 and e_2.

Example 5.9. *Let $P := \{p_1, p_2, p_3\}$, $e_1 := \langle \{p_1, p_2\}, \{p_3\} \rangle$, $e_2 := \langle \{p_1\}, \emptyset \rangle$, and $e_3 := \langle \{p_3\}, \{p_2\} \rangle$. Then e_1 and e_3 do not agree, because the positive effect of e_1 has a nonempty intersection with the negative effect of e_3. But e_2 and e_3 agree, and their combination is $\langle \{p_1, p_3\}, \{p_2\} \rangle$. e_1 and e_2 agree, too, and their combination is e_1 again, because e_2 is a subeffect of e_1.*

Then we define the effects of $\alpha \sqcap \beta$ in s to be all *possible* combinations of effects of α and β in s:

$$\mathrm{E}(\alpha \sqcap \beta, P, s) := \left\{ \langle Q_\alpha^+ \cup Q_\beta^+, Q_\alpha^- \cup Q_\beta^- \rangle \;\middle|\; \begin{array}{c} \langle Q_\alpha^+, Q_\alpha^- \rangle \in \mathrm{E}(\alpha, P, s), \langle Q_\beta^+, Q_\beta^- \rangle \in \mathrm{E}(\beta, P, s), \\ Q_\alpha^+ \cap Q_\beta^- = Q_\alpha^- \cap Q_\beta^+ = \emptyset \end{array} \right\}$$

The interpretation function is otherwise defined exactly as for **O-PDDL**.

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 $\mathrm{E}(\alpha, P, s) = \{\langle \{p_1\}, \{p_2\} \rangle, \langle \{p_1, p_2\}, \emptyset \rangle, \langle \{p_3\}, \{p_2\} \rangle\}$, since in the last combination, $-p_2 \sqcap +p_3$ and $+p_2$ do not agree.

Note that the expression $+p \sqcap -p$ (for an arbitrary $p \in \mathbb{P}$) defines an action with no successor. We use $\bot$ as a shorthand for it, but we do not need to include it into the grammar, contrary to **O-PDDL**. The void action ε can be defined in **E-PDDL** in the same way as in **O-PDDL**, via $(p \wedge \neg p) \rhd +p$.

Example 5.10. *Our running example 5.1 is expressed in* **E-PDDL** *as follows:*

$$\begin{array}{rl} & (+\text{no\_pain} \sqcap +\text{cured}) \\ \text{surgery} := & \cup (+\text{no\_pain} \sqcap +\text{dead}) \\ & \cup (-\text{no\_pain} \sqcap +\text{cured}) \end{array}$$

The description is not very natural because we have to write all possible combinations of effects of anesthesia and surgery explicitly. If we replaced & by $\sqcap$ in the expression from Example 5.6, the combination of the choices $+$no_pain *and* $(-$no_pain $\sqcap +$cured$)$*, i.e. the situation where the anesthesia worked and the surgery went well, would not be possible.*

Example 5.2, however, is expressed in **E-PDDL** *more easily and naturally than in* **O-PDDL**:

$$\text{save\_all} ::= \text{child\_on\_road} \rhd \left(\begin{array}{c} \overbrace{(+\text{left} \cup -\text{moving})}^{\textit{save the pedestrians}} \\ \sqcap \underbrace{(-\text{left} \cup (+\text{left} \sqcap +\text{airbag}))}_{\textit{save the passengers}} \end{array} \right)$$

Due to the semantics of $\sqcap$ *it is impossible to choose going left to save the pedestrians and choose not to go left to save the passengers, in contrast with the semantics of* & *where this would result in simply going left. This captures the intuition of the impossibility to break the laws of nature.*

It is easy to see, with exactly the same proof as for Proposition 5.7 (with $\sqcap$ instead of &), that **E-PDDL** is also minimally complete.

5.4 NNF Action Theories

We finally define the language **NNFAT** 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 expressions (Cimatti and Roveri, 2000; Bryce et al., 2006; To et al., 2015). Note that such planners in fact use more efficient representations, like OBDDs or DNFs, which are defined by further restrictions on NNF. Then a sublanguage of NNF to work with can be chosen using the knowledge compilation map for Boolean functions (Darwiche and Marquis, 2002). The general results from our work can be used for sublanguages in the usual way: separation results showing that some language L cannot be translated into **NNFAT** without a superpolynomial increase in size and complexity upper bounds for queries can be directly transferred to all sublanguages of **NNFAT**, whereas for transformations in general we can not draw conclusions.

To prepare the definition we associate an auxiliary variable $p' \notin \mathbb{P}$ to each variable $p \in \mathbb{P}$; p' is intended to denote the value of p *after* the action took place, while p denotes the value *before*. For any set $P \subseteq \mathbb{P}$, we write $(P)'$ for the set $\{p' \mid p \in P\}$.

Definition 5.11 (NNFAT). An **NNFAT** *action description* is a Boolean formula α in NNF over $P \cup (P)'$ for some set of state propositions $P \subset \mathbb{P}$.

The interpretation of an **NNFAT** action description α is defined for all $P \subset \mathbb{P}$ such that $\mathrm{V}(\alpha) \subseteq P \cup (P)'$ by

$$\forall s_1 \subseteq P\colon\ I(\alpha, P)(s_1) = \{s_2 \subseteq P \mid s_1 \cup (s_2)' \models \alpha\}$$

where $s_1 \cup (s_2)'$ is the assignment to $P \cup (P)'$ corresponding to the transition from s_1 to s_2. For P such that $\mathrm{V}(\alpha) \not\subseteq P \cup (P)'$, $I(\alpha, P)$ is not defined. In words, an **NNFAT** action description 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 $\mathbb{P} \cup (\mathbb{P})'$. Thus **NNFAT** is a declarative language, that describes state transitions via conditions on the predecessor and successor states together.

Importantly, **NNFAT** does not assume persistency of values, so that if, for example, a variable does not appear at all in an **NNFAT** action description, then this means that its value after the execution of the action can be arbitrary. For instance, the expression $p_1' \vee (\neg p_2 \vee p_3')$, when interpreted over $\{p_1, p_2, p_3, p_4\}$, represents an action which sets p_4 to any value and either (1) sets p_1 to true and p_2, p_3 to any value (nondeterministically), or (2) sets p_3 to true and p_1, p_2 to any value, in case p_2 is true in the initial state, and otherwise sets each variable to any value, or (3) performs any consistent combination of (1) and (2). The non-persistency of variables is illustrated by expressing our running examples in **NNFAT**.

Example 5.12. *We first remark that although in this example we use* $\rightarrow$ *and* $\leftrightarrow$ *which are not part of the grammar of* **NNFAT**, *we do it only for the sake of better comprehensibility. Since* $\alpha \leftrightarrow \beta \equiv (\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta)$ *and negation can be pushed to the leaves in linear time, and we do not have nested*

equivalences in our example, the equivalent **NNFAT** *representation is not much longer, but by far less comprehensible than the given expression.*

For the example with the surgery we need to express that variables which are not affected by a certain effect retain their value. This is ensured by the conjunction with the equivalences stating exactly this in the end of each line.

$$\text{surgery} \coloneqq \begin{pmatrix} (\text{no\_pain}' \wedge \text{cured}' \wedge (\text{dead} \leftrightarrow \text{dead}')) \\ \vee(\text{no\_pain}' \wedge \text{dead}' \wedge (\text{cured} \leftrightarrow \text{cured}')) \\ \vee(\neg\text{no\_pain}' \wedge \text{cured}' \wedge (\text{dead} \leftrightarrow \text{dead}')) \end{pmatrix}$$

As for the second example 5.2:

$$\text{save\_all} \coloneqq \begin{pmatrix} \text{child\_on\_road} \rightarrow \begin{pmatrix} (\text{left}' \vee \neg\text{moving}') \\ \wedge(\neg\text{left}' \vee (\text{airbag}' \wedge \text{left}')) \\ \wedge(\text{left}' \leftrightarrow \text{airbag}') \\ \wedge(\text{left}' \rightarrow \neg\text{moving}') \end{pmatrix} \end{pmatrix} \wedge \begin{pmatrix} (\text{child\_on\_road} \leftrightarrow \text{child\_on\_road}') \\ \wedge \begin{pmatrix} (\text{left} \not\leftrightarrow \text{left}') \\ \vee(\text{moving} \not\leftrightarrow \text{moving}') \\ \vee(\text{airbag} \not\leftrightarrow \text{airbag}') \end{pmatrix} \rightarrow \text{child\_on\_road} \end{pmatrix}$$

Even though we use additional symbols, we need a lot of space to describe the fact that a change of a variable must have an explicit reason mentioned in the description of the effects, which means here: if a variable changes a value via save_all *there must be a situation that requires a reaction causing this change, i.e. there are people on the road.*

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

Remark 5.13. **NNFAT** is minimally complete in the sense that since negation is only allowed to occur at leaves, we cannot exclude neither $\wedge$ nor $\vee$ from the grammar while keeping full expressivity. In other words, in order to be able to express all actions with e.g. only atomic actions p and p' and operators $\wedge$ and $\neg$, we would need to allow expressions of the form $\neg\alpha$ with non-atomic α, and then $\neg\alpha$ would not be an **NNF** anymore.

Observe that the logical $\vee$ in **NNFAT** allows it to express nondeterminism naturally and acts in exactly the same way as $\cup$ in **E-PDDL** and **O-PDDL**: for two actions α and β we have $(\alpha \vee \beta)(s) = (\alpha \cup \beta)(s) = \alpha(s) \cup \beta(s)$. Therefore it is important to emphasize that although at first sight the logical $\wedge$ and the parallel execution $\sqcap$ operators seem to be the same, too (for example, $+p \sqcap +q$ sets p and q true similarly to $p' \wedge q'$), they are actually quite different, and in general none of them can be seen as a generalisation of the other. The key difference is that while $\alpha \wedge \beta(s) = \alpha(s) \cap \beta(s)$, the operator $\sqcap$ can produce new successors (for example, $\{p,q\} \notin (+p)(\emptyset)$ and $\{p,q\} \notin (+q)(\emptyset)$, but $\{p,q\} \in (+p \sqcap +q)(\emptyset)$).

Finally, note that an **NNF** precondition φ for applicability of an action described by α can be included into the action description directly in **NNFAT** via $\varphi \wedge \alpha$.

CHAPTER 6

Complexity Results for Minimally Complete Languages

Contents

In this chapter we will give a complete KC map for the minimally complete languages **O-PDDL**, **E-PDDL** and **NNFAT**. Recall that for each of them we study both the tree and circuit representation, i.e. we will give complexity results for **O-PDDL**$^{\mathbf{T}}$, **O-PDDL**$^{\mathbf{C}}$, **E-PDDL**$^{\mathbf{T}}$, **E-PDDL**$^{\mathbf{C}}$, **NNFAT**$^{\mathbf{T}}$ and **NNFAT**$^{\mathbf{C}}$. We will summarize the results in form of a table and a diagram in the end of the chapter.

But before we turn to the subject of the chapter, we would like to justify why we study (truly) nondeterministic action languages: this is because removing the connectives which are responsible for nondeterminism (i.e. $\cup$ in **O-PDDL** and **E-PDDL** and $\vee$ in **NNFAT**) makes everything trivial. For example, for an action description α with scope P generated by the grammar

$$\alpha ::= +p \mid -p \mid \varphi \rhd \alpha \mid \alpha \sqcap \alpha$$

deciding applicability in $s \subseteq P$ is easy (contrary to real **E-PDDL**, see Proposition 6.13): first replace each subexpression $\varphi \rhd \beta$ with ε if $s \not\models \varphi$ and with β if $s \models \varphi$ (we will later give a name to this procedure in Definition 6.10). Then α is applicable if and only if there is no variable $p \in P$ such that $+p$ and $-p$ are both atomic subactions of α.

Before we finally start with proving our complexity results we will prove some nontrivial statements about the imperative languages.

Lemma 6.1. *For all action descriptions α, β, γ in* **E-PDDL** *and* **O-PDDL** *(with respect to whether & or $\sqcap$ is used) it holds:*

1. *In general, $(\alpha \sqcap \alpha) \not\equiv \alpha$ and $(\alpha \,\&\, \alpha) \not\equiv \alpha$*
2. *If $s' \in \alpha(s) \cap \beta(s)$ then $s' \in (\alpha \sqcap \beta)(s)$ and $s' \in (\alpha \,\&\, \beta)(s)$*
3. *If all the effects of α and β in s are P-maximal, then $(\alpha \sqcap \beta)(s) = \alpha(s) \cap \beta(s)$, but in general $\alpha(s) \cap \beta(s) \subsetneq (\alpha \,\&\, \beta)(s)$*

4. *For all s,* $\mathrm{E}(\alpha \sqcap (\beta \cup \gamma), P, s) = \mathrm{E}((\alpha \sqcap \beta) \cup (\alpha \sqcap \gamma), P, s)$ *(and hence in particular* $\alpha \sqcap (\beta \cup \gamma) \equiv (\alpha \sqcap \beta) \cup (\alpha \sqcap \gamma)$*), but in general,* $\alpha \cup (\beta \sqcap \gamma) \not\equiv (\alpha \cup \beta) \sqcap (\alpha \cup \gamma)$.
The same is true for & *instead of* $\sqcap$*, i.e.* $\mathrm{E}(\alpha \,\&\, (\beta \cup \gamma), P, s) = \mathrm{E}((\alpha \,\&\, \beta) \cup (\alpha \,\&\, \gamma), P, s)$*, but in general* $\alpha \cup (\beta \,\&\, \gamma) \not\equiv (\alpha \cup \beta) \,\&\, (\alpha \cup \gamma)$.

5. *Conditions can be distributed over other operators:* $\mathrm{E}(\varphi \rhd (\alpha \sqcap \beta), P, s) = \mathrm{E}((\varphi \rhd \alpha) \sqcap (\varphi \rhd \beta), P, s)$*,* $\mathrm{E}(\varphi \rhd (\alpha \,\&\, \beta), P, s) = \mathrm{E}((\varphi \rhd \alpha) \,\&\, (\varphi \rhd \beta), P, s)$ *and* $\mathrm{E}(\varphi \rhd (\alpha \cup \beta), P, s) = \mathrm{E}((\varphi \rhd \alpha) \cup (\varphi \rhd \beta), P, s)$*, and* $\mathrm{E}(\varphi \rhd (\psi \rhd \alpha), P, s) = \mathrm{E}(\varphi \wedge \psi \rhd \alpha, P, s)$.

PROOF.

1. For $P \coloneqq \{p, q\}$, $\alpha \coloneqq +p \cup +q$ and $s \coloneqq \emptyset$, observe that $s' \coloneqq \{p, q\}$ is not an α-successor of s, but it is an $(\alpha \sqcap \alpha)$ and $(\alpha \,\&\, \alpha)$-successor of s (choosing $+p$ once, and $+q$ once).

2. Every effect $\langle Q^+, Q^- \rangle$ witnessing a transition (s, s') satisfies $Q^+ \subseteq s'$ and $Q^- \cap s' = \emptyset$. If s' is both an α- and β-successor of s, then there are effects $\langle Q^+_\alpha, Q^-_\alpha \rangle$ of α and $\langle Q^+_\beta, Q^-_\beta \rangle$ of β witnessing this. Because of the previous statement it follows that $Q^+_\alpha \cap Q^-_\beta = Q^+_\beta \cap Q^-_\alpha = \emptyset$ and thus $\langle Q^+_\alpha \cup Q^+_\beta, Q^-_\alpha \cup Q^-_\beta \rangle$ is an effect of $\alpha \sqcap \beta$ and $\alpha \,\&\, \beta$ in s witnessing (s, s').

3. Two P-maximal effects agree only if they are equal, and a $\sqcap$-combination of an effect with itself is itself. As for $\alpha \,\&\, \beta$, consider $P = \{p\}$ and $\alpha = +p$, $\beta = -p$, $s = \emptyset, s' = \{p\}$. Then $s' \in (\alpha \,\&\, \beta)(s)$ but $\alpha(s) \cap \beta(s) = \emptyset$. But since an &-combination of an effect with itself is still the same effect, it holds that $\alpha(s) \cap \beta(s) \subseteq (\alpha \,\&\, \beta)(s)$.

4. The first part follows directly from the definitions. The second part is similar to Item 1 of the lemma: let $P \coloneqq \{p, q, r\}$, $\alpha \coloneqq +p \cup +q$, $\beta \coloneqq +r$, $\gamma \coloneqq +r$, and $s \coloneqq \emptyset$, then we have $\{p, q\} \in (\alpha \cup \beta) \sqcap (\alpha \cup \gamma)(s)$ as a result of choosing α twice, once with $+p$ and once with $+q$, while $\{p, q\}$ is obviously not an $\alpha \cup (\beta \sqcap \gamma)$-successor of s. The proof is exactly the same for & in place of $\sqcap$.

5. Follows directly from the definitions.

□

Proposition 6.2. *There is a polynomial-time translation from* **NNFAT$^{\mathbf{T}}$** *to* **E-PDDL$^{\mathbf{T}}$***, and one from* **NNFAT$^{\mathbf{C}}$** *to* **E-PDDL$^{\mathbf{C}}$**.

PROOF. Let f be the function defined inductively as follows, for all **NNFAT** action descriptions α, α_1, α_2 and sets of variables P such that $P \cup (P)' \supseteq \mathrm{V}(\alpha)$:

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

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

3. for $p \in P$, $f(p') \coloneqq +p \sqcap \left(\bigsqcap_{q \in P, q \neq p} (+q \cup -q) \right)$;

4. for $p \in P$, $f(\neg p') \coloneqq -p \sqcap \left(\bigsqcap_{q \in P, q \neq p} (+q \cup -q) \right)$;

5. $f(\alpha_1 \wedge \alpha_2) \coloneqq f(\alpha_1) \sqcap f(\alpha_2)$;

6. $f(\alpha_1 \vee \alpha_2) \coloneqq f(\alpha_1) \cup f(\alpha_2)$.

We show by induction that all the effects of $f(\alpha)$ (in any state) are P-maximal and that f is indeed a translation from **NNFAT** to **E-PDDL**, i.e. $f(\alpha)$ and α describe the same action. Assume that the claim is proven for α_1 and α_2.

For the base cases, Items 1–4 take care of the fact that there is no implicit persistency of values in **NNFAT**. Since they explicitly assign all variables, all the effects of $f(\alpha)$ are P-maximal in the sense of Definition 5.4.

Now 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 by the induction hypothesis $f(\alpha_1)$ and $f(\alpha_2)$ have only P-maximal effects, it follows together with Item 3 of Lemma 6.1 that $\mathrm{E}(f(\alpha_1) \sqcap f(\alpha_2), P, s) = \mathrm{E}(f(\alpha_1), P, s) \cap \mathrm{E}(f(\alpha_2), P, s)$, so that $s' \in f(\alpha_1)(s) \wedge s' \in f(\alpha_2)(s)$ is equivalent to $s' \in \big(f(\alpha_1) \sqcap f(\alpha_2)\big)(s)$, that is, to $s' \in f(\alpha)(s)$, and $f(\alpha_1) \sqcap f(\alpha_2)$ has only P-maximal effects again.

Now 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, $s' \in \big(f(\alpha_1) \cup f(\alpha_2)\big)(s) = f(\alpha)(s)$. Conversely, assume $s' \in f(\alpha)(s)$, then by the definition of $f(\alpha)$ and the semantics of $\cup$ 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, $s' \in \alpha(s)$. It is easy to see that $f(\alpha_1) \cup f(\alpha_2)$ has only P-maximal effects, since $f(\alpha_1)$ and $f(\alpha_2)$ had.

Thus we have shown that α and $f(\alpha)$ describe the same action. Since f rewrites each node of α independently from the others, it is polynomial-time both when applied to an expression in **NNFAT**$^{\mathbf{T}}$ (with a result in **E-PDDL**$^{\mathbf{T}}$) and to an expression in **NNFAT**$^{\mathbf{C}}$ (with a result in **E-PDDL**$^{\mathbf{C}}$). □

The reason why the proof works only for **E-PDDL**, but not for **O-PDDL** is that because of Item 3 from Lemma 6.1 $\sqcap$ can naturally express $\wedge$, but & cannot (Item 3 of Lemma 6.1). In fact, we will show later that **O-PDDL** $\not\preceq$ **NNFAT**.

6.1 Queries

We recall that we are interested in the complexity of the queries IS-SUCC, IS-APPLIC, ENTAILS, ST, IS-DET and IS-MON for tree and circuit representations. The complexity proofs for queries are the key to the other proofs in our work, namely to hardness results for transformations and to succinctness separation results.

6.1.1 Successorship

As we mentioned before, IS-SUCC is not the most widely used, but the most basic query in the sense that it is analogous to model checking for Boolean formulas. One can say that the answers to IS-SUCC give all the information which is stored in an action description.

Proposition 6.3. Is-Succ *can be solved in linear time for* **NNFAT**$^{\mathbf{T}}$ *and* **NNFAT**$^{\mathbf{C}}$.

PROOF. By definition of **NNFAT**, $s' \in \alpha(s)$ holds if and only if the assignment to $P \cup (P)'$ 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 (Darwiche and Marquis, 2002). □

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

Notation 6.4. Let $n \in \mathbb{N}$, and let $X_n := \{x_1, \ldots, x_n\}$ be a set of variables. Let Γ_n be the set of clauses of length 3 over X_n. Observe that there are a cubic number N_n of such clauses (any choice of 3 variables with a polarity for each). 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 $\{p_1, p_2, \ldots, p_{N_n}\}$. 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. For a literal ℓ and clause γ we write $\ell \in \gamma$ meaning that ℓ occurs in γ at least once.

Example 6.5. *Let $n := 2$, and consider an enumeration of all clauses over $X_2 := \{x_1, x_2\}$ which starts with $\gamma_1 := (x_1 \vee x_1 \vee x_2), \gamma_2 := (x_1 \vee x_1 \vee \neg x_2), \gamma_3 := (x_1 \vee \neg x_1 \vee x_2), \gamma_4 := (x_1 \vee \neg x_1 \vee \neg x_2), \gamma_5 := (\neg x_1 \vee \neg x_1 \vee x_2), \gamma_6 := (\neg x_1 \vee \neg x_1 \vee \neg x_2), \ldots$. 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 encoded by the state $s(\varphi) = \{p_1, p_5\}$.*

Notation 6.6. Using Notation 6.4, for all $n \in \mathbb{N}$ we define the **E-PDDL$^{\mathbf{T}}$** action description $\alpha_n^{\mathrm{sat},\sqcap}$ by

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

Analogously we obtain the **O-PDDL** action description $\alpha_n^{\mathrm{sat},\&}$ by replacing $\sqcap$ by $\&$:

$$\alpha_n^{\mathrm{sat},\&} := \bigotimes_{x \in X_n} \left(\left(\bigotimes_{\gamma_i \colon x \in \gamma_i} (+p_i \cup \varepsilon) \right) \cup \left(\bigotimes_{\gamma_i \colon \neg x \in \gamma_i} (+p_i \cup \varepsilon) \right) \right)$$

Since there are no negative assignments $-p_i$ in the above action descriptions, $\sqcap$ and $\&$ have the same interpretation and thus $\alpha_n^{\mathrm{sat},\sqcap}$ and $\alpha_n^{\mathrm{sat},\&}$ describe the same nondeterministic action. In the following we will prove statements for $\alpha_n^{\mathrm{sat},\sqcap}$, which are therefore also true for $\alpha_n^{\mathrm{sat},\&}$.

Intuitively, $\alpha_n^{\mathrm{sat},\sqcap}$ (and $\alpha_n^{\mathrm{sat},\&}$) 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. The following lemma is then a direct consequence.

Lemma 6.7. *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 $\alpha_n^{\mathrm{sat},\sqcap}$-successor of the state $\emptyset$. Analogously, φ is satisfiable if and only if $s(\varphi) \in \alpha_n^{\mathrm{sat},\&}(\emptyset)$*

Corollary 6.8. Is-Succ *is* NP*-hard for the tree and circuit representations of* **O-PDDL** *and* **E-PDDL**.

Proposition 6.9. Is-Succ *is* NP*-complete for both tree and circuit representations of* **O-PDDL** *and* **E-PDDL**.

PROOF. Hardness follows from Corollary 6.8, thus it remains to show NP-membership of IS-SUCC for **E-PDDL$^{\mathbf{C}}$** and **O-PDDL$^{\mathbf{C}}$**. These results then apply to the tree representations (Remark 4.12, page 24).

We first give the proof for **E-PDDL$^{\mathbf{C}}$**. The intuition is to guess a branch to be chosen for each $\cup$-node in the circuit. 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.

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'. Precisely, we define a *labelling* to be a function assigning to each node and edge in the circuit a number in $[0..2^{|\alpha|}]$ (using $O(|\alpha|)$ bits), and satisfying the following conditions.

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,

Clearly, the size of a labelling is polynomial in the size of the circuit, and by construction, there is a given labelling of the circuit if and only if there is an execution of the action which executes each node as many times as its label indicates.

Consequently, $s' \in \alpha(s)$ holds if and only if there is a labelling such that the action $\ell_1 \sqcap \cdots \sqcap \ell_k$, where $\ell_1, \ldots, \ell_k$ 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). This can be verified in polynomial time, so the problem is indeed in NP.

For **O-PDDL$^{\mathbf{C}}$** the proof is exactly the same, but with $\&$ instead of $\sqcap$, and for $\ell_1 \& \cdots \& \ell_k$ we do not need consistency, but that the negative assignments necessary for the transition (s, s') are not overwritten by the positive ones, and that no $\bot$-node is executed. $\square$

6.1.2 Applicability and Entailment

The proof of the following statement follows a method which we will repeatedly use later to prove NP- or P-membership of queries. Since the principal idea of those proofs is the same, we will present it at this point in all details, and in those proofs which make use of this idea later we will refer to the detailed version. First we need to introduce the procedure which we will call *reducing and instantiating* a circuit.

Definition 6.10. An **E-PDDL** or **O-PDDL** circuit α is *instantiated* by a state s as follows: for each node n in α that is of the form $\varphi \rhd \beta$ redirect all arcs pointing to n to β if $s \models \varphi$, and to ε if $s \not\models \varphi$, and then remove the $\varphi\rhd$.

For a circuit of an **E-PDDL** or **O-PDDL** action description α we can remove some nodes $N = \{n_1, \ldots, n_k\}$ and arcs $L = \{l_1, \ldots, l_m\}$ and then obtain the *reduced circuit with respect to N and L* as follows:

1. for each node C which has been removed remove also the arcs which used to point at C
2. for each operator node which has no more children: remove this node
3. for each $\cup$-node $\gamma \cup \beta$ whose child γ has been removed, but β is still there, redirect all arcs directly to β and remove the $\cup$-node
4. for each $\sqcap$-node (&-node respectively), if at least one child has been removed – remove the $\sqcap$ (resp. &)-node itself
5. if a subformula is not connected to the root, remove the whole subformula

All steps of the reduction algorithm need to be repeated as long as they can be applied. The remainder will be a rooted connected graph.

In words, an instantiation by s of a circuit is a simplification which removes the conditional constructs from a circuit without changing the effects for a particular state. Therefore, if $\hat{\alpha}$ is an instantiation of α by s then for all s' it holds that $s' \in \alpha(s) \Leftrightarrow s' \in \hat{\alpha}(s)$. Thus we can assume for any procedure of answering a query about $\alpha(s)$ for **O-PDDL** or **E-PDDL** that the first step will be an instatiation by the input state.

Removing some nodes and arcs from a circuit disturbs the semantics of other nodes of the circuit. By reducing the graph with respect to this removal we “rescue what can be recued", i.e. we eliminate all parts of the circuit which cannot be executed anymore. Thus the reduced circuit allows for exactly those executions which are still possible after removing the nodes and arcs.

An application of this procedure is for example removing all nodes which definitely cannot be executed (like $\bot$) and then checking whether the reduced graph still allows for a sensible execution. We will do exactly this in the proof of the next proposition.

Proposition 6.11. Is-Applic *is linear-time for both representations of* **O-PDDL**.

PROOF. Again, it is enough to prove the claim for circuits by Remark 4.12 (page 24).

The only reason for an **O-PDDL**$^{\mathbf{C}}$ action α to fail in state s can be an execution of a $\bot$-leaf. Since a $\bot$-leaf can never be executed due to its semantics, it's an "if and only if"-situation: an **O-PDDL**$^{\mathbf{C}}$-action is non-applicable if and only if in each possible execution there necessarily exists an executed $\bot$-node. Therefore in order to check applicability we can remove all $\bot$-leaves and see whether the reduced circuit still allows for an execution. The detailed algorithm looks as follows:

1. Instantiate α by s.
2. Remove all $\bot$-nodes.
3. Reduce the circuit with respect to this removal.
4. If the root has been removed then α is non-applicable in s; if the root has not been removed, α is applicable in s.

It is important to instantiate the circuit and not directly start with removing the $\bot$-leaves because otherwise we might remove a $\varphi \rhd \bot$-node with $s \not\models \varphi$, although the failure should actually be replaced by an ε.

For the correctness of the algorithm we observe that if we removed the root node it means that we necessarily removed the rest of the graph because it was not connected to the root anymore. Therefore the executable remainder of the formula is simply non-existent. On the other hand, if the root has not been removed by the reduction algorithm, there exists a possible execution of α without the need to execute any of the $\bot$-nodes, which is therefore a valid **O-PDDL**$^{\mathbf{C}}$-execution.

We see that the algorithm is executable in polynomial time (actually even in linear time) because the decision whether to remove a node or an arc or not is made in constant time, and we need to make this decision a linear number of times, in a bottom-up fashion. □

Proposition 6.12. Is-Applic *is* NP*-complete for* **NNFAT**$^{\mathbf{T}}$ *and* **NNFAT**$^{\mathbf{C}}$.

PROOF. IS-APPLIC is in NP, because applicability of α in s can be witnessed by a particular successor, and successorship can be verified in linear time for both **NNFAT**$^{\mathbf{T}}$ and **NNFAT**$^{\mathbf{C}}$ by Proposition 6.3. For hardness, let φ be a propositional NNF formula over variables $p_1, \ldots, p_n$. Let the **NNFAT**$^{\mathbf{T}}$ action description φ' be obtained from φ by replacing each p_i with p'_i. Then φ' is applicable in any state (say in $s := \emptyset$) if and only if φ is satisfiable, because applicability of φ' means the existence of a model (s, s') of φ', where the unprimed variables are fictional for the Boolean function described by φ'. Hence IS-APPLIC is NP-hard for **NNFAT**$^{\mathbf{T}}$ (and hence for **NNFAT**$^{\mathbf{C}}$ as well). □

Proposition 6.13. Is-Applic *is* NP*-complete for* **E-PDDL**$^{\mathbf{T}}$ *and* **E-PDDL**$^{\mathbf{C}}$.

PROOF. Hardness follows from the existence of a polynomial-time translation from **NNFAT** into **E-PDDL** (Proposition 6.2). For membership we observe that a certificate for applicability of α in s is a successor $s' \in \alpha(s)$ with the corresponding successorship certificate, which can be verified in polynomial time because IS-SUCC in in NP for **E-PDDL**$^{\mathbf{C}}$ (Proposition 6.9). □

We now turn to the query ENTAILS. Since model checking for Boolean NNF formulas is linear-time, and IS-SUCC is in NP for all our minimally complete languages, the following proposition is straightforward, too.

Proposition 6.14. Entails *is in* coNP *for both representations of* **NNFAT**, **O-PDDL** *and* **E-PDDL**.

PROOF. Suppose we are given a sequence $\alpha_1, \ldots, \alpha_k$ of action descriptions in any of the above languages together with a state s and a formula φ. A witness for non-entailment would be a sequence of states $s_0 = s, s_1 \in \alpha_1(s_0), s_2 \in \alpha_2(s_1), \ldots, s_k \in \alpha_k(s_{k-1})$ such that $s_k \not\models \varphi$ together with the k corresponding successorship witnesses. □

We observe that since by Lemma 4.7 (page 23) the complement of IS-APPLIC is reducible to ENTAILS we deduce the following result from Propositions 6.12 and 6.13.

Corollary 6.15. Entails *is* coNP*-hard for* **NNFAT**$^{\mathbf{T}}$, **NNFAT**$^{\mathbf{C}}$, **E-PDDL**$^{\mathbf{T}}$ *and* **E-PDDL**$^{\mathbf{C}}$.

It remains to show hardness for **O-PDDL**.

Proposition 6.16. Entails *is* coNP*-hard for* **O-PDDL**$^{\mathbf{T}}$ *and* **O-PDDL**$^{\mathbf{C}}$.

PROOF. Consider the following **O-PDDL** action description over $X_n = \{x_1, \ldots, x_n\}$: $\alpha := \&_{x_i \in X_n}(+x_i \cup -x_i)$. This action is always applicable and the successors of any state are all possible assignments to the x_i variables. If now α entails a DNF φ over X_n this means that φ is satisfied by any assignment, i.e. it is valid. This is a reduction from a coNP-complete problem (checking the validity of a DNF formula) to ENTAILS. □

From Corollary 6.15 and Propositions 6.14 and 6.16 we conclude:

Corollary 6.17. Entails *is* coNP*-complete for both the tree and circuit representations of* **NNFAT**, **O-PDDL** *and* **E-PDDL**.

6.1.3 Other queries

We start with determining the complexity of IS-DET and IS-MON. Our argumentation in proofs will be similar, therefore the two queries are considered in the same subsection. First we observe that they are both in coNP.

Corollary 6.18. Is-Det *is in* coNP *for both representations of* **NNFAT**, **E-PDDL** *and* **O-PDDL**.

PROOF. A witness for "real" nondeterminism of α in s can be given by two α-successors s' and s'' of s with $s' \neq s''$ together with the corresponding witnesses for successorship, which can be verified in polynomial time by Propositions 6.3 and 6.9. □

Corollary 6.19. Is-Mon *is in* coNP *for both representations of* **NNFAT**, **E-PDDL** *and* **O-PDDL**.

PROOF. A witness for positive nonmonotony of α in s can be given by a successor s' with $s \not\subseteq s'$ ($s' \not\subseteq s$ for negative nonmonotony) together with a corresponding witness for successorship. □

Proposition 6.20. Is-Det *can be answered in polynomial time for* **O-PDDL**$^{\mathbf{T}}$ *and* **O-PDDL**$^{\mathbf{C}}$.

PROOF. It is enough to show the claim for the circuit representation of **O-PDDL**. Let α be an **O-PDDL**$^{\mathbf{C}}$ action and s a state.

The algorithm resembles that from the proof of Proposition 6.11 and even makes use of it. To check whether α is deterministic in s we go through all conditions which would ensure nondeterminism, and argue that if none of them is satisfied then α is deterministic in s.

We first instantiate α by s and remove the $\bot$-leaves since they do not contribute to any possible nonfailing execution, and we reduce the graph. I.e. we check whether α is applicable in s, and if it is not, then it is deterministic by definition. If α is applicable in s, we call the instantiated and reduced graph $\hat{\alpha}$ and from now on work with it. If α (equivalently $\hat{\alpha}$) is nondeterministic in s, there must exist a state variable p and successors $s', s'' \in \alpha(s)$ with $p \in s'$ and $p \notin s''$. If $p \in s$ then there must exist an assignment $-p$ which is executed and not overwritten by a $+p$ during at least one execution, and another execution where either no $-p$ is executed or it is overwritten by a $+p$. Therefore we have to check for each $p \in s$: is there a $-p$ leaf in α such that there is a possible execution without any $+p$-leaf being executed in parallel? We do this by executing the following "algorithm":

1. Remove all $+p$-nodes and reduce the remaining graph, obtaining $\hat{\alpha}_{-p}$. If there is still a leaf $-p$ with a path to the root, we go on with Item 2. If there is no such $-p$ leaf, we go on with a next variable which is true in s, or go to the step with the variables which are false in s.

2. There is a $-p$ leaf in $\hat{\alpha}_{-p}$, and thus there is at least one successor with p being false. So we come back to the original instantiated applicable graph $\hat{\alpha}$ and check whether there is a $+p$ leaf.

   - If there is a $+p$ leaf, it can be executed since in $\hat{\alpha}$ every leaf node can be executed in at least one possible execution, and therefore at least one successor where p is true, and thus α is nondeterministic.
   - If there is no $+p$ leaf in $\hat{\alpha}$, we remove all $-p$ leaves, reduce the graph and check whether the root is still there. If the answer is yes, there is at least one execution where p has not been assigned, and since it was true before, it means there is at least one successor with p being true, and thus α is nondeterministic. If the answer is "no", we go on with the next variable which is true in s, or go to the step with the variables which are false in s.

Now for each $p \notin s$ we have to do the dual thing. We first check whether there is a $+p$-leaf in $\hat{\alpha}$. If there is one, there is at least one execution which sets p to $\top$. Then we need to check whether there is an execution which keeps p false. If after removing all the $+p$-leaves and reducing the graph the root is still there, then p must have remained or been reassigned to $\bot$ and there is at least one successor with p being false and thus α is nondeterministic.

Now it remains to verify that the above possibilities were the only possible reasons for α to be nondeterministic. Indeed, the only reason for a variable p to change its value is an execution of some elementary assignment $\pm p$, and since we checked for each p if there is an option to keep and an option to change its value, and the answer has been negative for each variable, it means that α must have been deterministic. □

Proposition 6.21. *The complement of* Is-Applic *is polynomial-time reducible to* Is-Det *for both representations of* **NNFAT** *and* **E-PDDL**.

PROOF. Let α be an **NNFAT** action with scope P and s be a state. Let $q \notin P$ be a fresh variable. Then α is non-applicable in s if and only if the action $\hat{\alpha} \coloneqq \alpha \wedge (q' \vee \neg q')$ is deterministic in s, because due to the $(q' \vee \neg q')$-part for each α-successor s' of s there are at least two $\hat{\alpha}$-successors of s, namely s' and $s' \cup \{q\}$. Thus $\hat{\alpha}$ is nondeterministic if and only if α has no successors in s.

Analogously, an **E-PDDL** action β is not applicable in s if the action $\beta \sqcap (+q \cup -q)$ is deterministic in s with an auxiliary variable q.

The proof is the same for the tree and the circuit representations. □

Proposition 6.22. Is-Mon *can be answered in polynomial time for* **O-PDDL$^{\mathbf{T}}$** *and* **O-PDDL$^{\mathbf{C}}$**.

PROOF. It is enough show the claim for the circuit representation of **O-PDDL**. Let α be an **O-PDDL$^{\mathbf{C}}$** action and s a state.

The proof is analogous to the proof of Proposition 6.20.

The only option for α to be not monotone is to contain an assignment $-p$ which will be executed while $p \in s$.

Therefore, after instantiating the circuit by s, removing all $\bot$-nodes and reducing the graph (because a non-applicable action is automatically monotone), for each $p \in s$ we remove all $+p$, reduce the graph and check whether the connected remainder contains a $-p$. If it does, there exists an execution which changes a positive value to a negative, and thus α is not monotone in s.

If we checked all $p \in s$ and didn't find a variable which could be assigned $\bot$, then α is positively monotone.

Checking whether α is negatively monotone in s is even easier: an action is not negatively monotone in s if and only if for some $p \notin s$ there exists a $+p$-leaf in the reduced remainder of the instantiated graph. If there is one, then there is necessarily an execution which changes the value of p from $\bot$ to $\top$ and thus α is not negatively monotone. If no $+p$ is executable for all $p \notin s$, then α is negatively monotone. □

Proposition 6.23. *The complement of* Is-Applic *is polynomial-time reducible to* Is-Mon *for both representations of* **NNFAT** *and* **E-PDDL**.

PROOF. The proof resembles that of Proposition 6.21. Let α be an **NNFAT$^{\mathbf{C}}$** action with scope P and s be a state. Let $q \notin P$ be a fresh variable. Then α is non-applicable in s if and only if the action $\alpha \wedge \neg q'$ is positively monotone in $s \cup \{q\}$. Analogously, an **E-PDDL$^{\mathbf{C}}$** action β is non-applicable in state s if and only if the action $\beta \sqcap -q$ is positively monotone in $s \cup \{q\}$ for variable q. The proof for checking negative monotony works in the same way: an **NNFAT** action α is non-applicable in s if and only if the action $\alpha \wedge q'$ is negatively monotone in s, and an **E-PDDL$^{\mathbf{C}}$** action β is non-applicable in state s if and only if the action $\beta \sqcap +q$ is negatively monotone in s. □

Together with the results about applicability (Propositions 6.12 and 6.13) we conclude.

Corollary 6.24. Is-Det *is* coNP*-complete for both representations of* **NNFAT** *and* **E-PDDL**.

Corollary 6.25. Is-Mon *is* coNP*-complete for both representations of* **NNFAT** *and* **E-PDDL**.

We finally turn to the query ST.

Proposition 6.26. ST *is linear-time for both representations of* **E-PDDL** *and* **NNFAT**.

PROOF. We first show the claim for **E-PDDL**. Let α be an **E-PDDL**-action and s a state. The following cases are possible:

- $\alpha = +p$: then $s \in \alpha(s)$ if and only if $p \in s$
- $\alpha = -p$: then $s \in \alpha(s)$ if and only if $p \notin s$
- $\alpha = \beta \cup \gamma$: then $s \in \alpha(s)$ if and only if $s \in \beta(s)$ or $s \in \gamma(s)$
- $\alpha = \varphi \rhd \beta$: then $s \in \alpha(s)$ if and only if $s \not\models \varphi$ or $s \in \beta(s)$
- $\alpha = \beta \sqcap \gamma$: then $s \in \alpha(s)$ if and only if $s \in \beta(s)$ and $s \in \gamma(s)$. Indeed, if $s \in \beta(s)$ and $s \in \gamma(s)$ then $s \in \beta(s) \cap \gamma(s) \subseteq (\beta \sqcap \gamma)(s)$ with the latter inclusion shown in Item 2 of Lemma 6.1. Conversely, if $s \in (\beta \sqcap \gamma)(s)$ then there exist effects $\langle Q_\beta^+, Q_\beta^- \rangle$ of β in s and $\langle Q_\gamma^+, Q_\gamma^- \rangle$ of γ in s with $s = (s \cup (Q_\beta^+ \cup Q_\gamma^+)) \setminus (Q_\beta^- \cup Q_\gamma^-)$ and $Q_\beta^+ \cap Q_\gamma^- = \emptyset = Q_\gamma^+ \cap Q_\beta^-$. Hence $Q_\beta^+, Q_\gamma^+ \subseteq s$ and $s \cap Q_\beta^- = s \cap Q_\gamma^- = \emptyset$, and therefore $(s \cup Q_\beta^+) \setminus Q_\beta^- = (s \cup Q_\gamma^+) \setminus Q_\gamma^- = s$ and $s \in \beta(s)$ and $s \in \gamma(s)$.

Thus ST can be checked in a bottom-up manner from the leaves $\pm p$ to the root in linear time.

For **NNFAT** the algorithm works in exactly the same manner, but with $\vee$ instead of $\cup$, $\wedge$ instead of $\sqcap$, p' instead of $+p$, $-p'$ instead of $-p$. For atomic actions of the form $\alpha = p$ or $\alpha = \neg p$ we observe that $s \in (p)(s)$ if and only if $p \in s$, and $s \in (\neg p)(s)$ if and only if $p \notin s$. □

Proposition 6.27. ST *is* NP*-complete for both* **O-PDDL**$^{\mathbf{T}}$ *and* **O-PDDL**$^{\mathbf{C}}$.

PROOF. Since IS-SUCC is in *NP* for **O-PDDL**$^{\mathbf{C}}$ by Proposition 6.9, we only need to show hardness for **O-PDDL**$^{\mathbf{T}}$ (recall Remark 4.12).

Recall the encoding of 3-CNFs over $X_n = \{x_1, \ldots, x_n\}$ from Notation 6.4 over the variables $p_1, \ldots, p_{N_n}$ and the corresponding clauses $\gamma_1, \ldots, \gamma_{N_n}$. The proof will use an **O-PDDL**$^{\mathbf{T}}$ action description β_n which is very similar to $\alpha_n^{\mathrm{sat},\&}$ from Notation 6.6. We define β_n as

$$\beta_n := \left(\operatorname*{\&}_{x \in X_n} \left(\left(\operatorname*{\&}_{\gamma_i : x \in \gamma_i} (+p_i \cup \varepsilon) \right) \cup \left(\operatorname*{\&}_{\gamma_i : \neg x \in \gamma_i} (+p_i \cup \varepsilon) \right) \right) \right) \,\&\, \left(\operatorname*{\&}_{j=1}^{N_n} -p_j \right)$$

Intuitively, it constructs satisfiable 3-CNFs and works in the same way as $\alpha_n^{\mathrm{sat},\&}$ with the additional effect that it assigns *all* the p_i variables. This means, it chooses an assignment to the x-variables, and then chooses for each clause which is satisfied by this assignment whether to include it or not, and in parallel it sets all the variables which correspond to clauses which are not chosen to false. We claim that a 3-CNF φ is satisfiable if and only if $s(\varphi) \in \beta_n(s(\varphi))$. Since $+$ overrides $-$, if φ is satisfiable then there exists a satisfying assignment to the x-variables such that the induced outer choices in β_n allow us to choose exactly the p_i corresponding to the clauses of φ. Since those p_i are already true in $s(\varphi)$, they are just reassigned and $s(\varphi)$ is indeed a β_n-successor of $s(\varphi)$. Conversely, if $s(\varphi) \in \beta_n(s(\varphi))$, then there exists a choice for each $\cup$-node of β_n such that all the $p_i \in s(\varphi)$ have been assigned $\top$. The choice made at the outer choice node induces a satisfying assignment to φ. □

6.2 Succinctness

Recall from Section 3.3 that the assumptions $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ and $\mathsf{coNP} \not\subseteq \mathsf{NP/poly}$ which we will use in the following are standard ones.

Proposition 6.28. *If* $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ *then there is no polynomial-size translation from* **E-PDDL**$^{\mathbf{T}}$ *to* **NNFAT**$^{\mathbf{T}}$*, nor from* **E-PDDL**$^{\mathbf{C}}$ *to* **NNFAT**$^{\mathbf{C}}$.

PROOF. We consider the action $\alpha_n^{\mathrm{sat},\sqcap}$ from Notation 6.6 again. The size of $\alpha_n^{\mathrm{sat},\sqcap}$ is clearly polynomial in n in both the tree and circuit representations. Assume that there is a polynomial-size translation f from **E-PDDL**$^{\mathbf{T}}$ to **NNFAT**$^{\mathbf{T}}$, and for all $n \in \mathbb{N}$ let $\beta_n^{\mathrm{sat}} := f(\alpha_n^{\mathrm{sat},\sqcap})$.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 6.4;
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 6.3), the algorithm is correct (Lemma 6.7), 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 $\mathsf{NP} \subseteq \mathsf{P/poly}$ holds.

The proof is exactly the same for circuits. □

The proof of Proposition 6.28 illustrates the scheme that we will use for our separation results throughout the paper. All results of the kind "L_1 is not polynomial-size translatable into L_2" will be proven using the fact that there exists a query which is hard in L_1 and less hard in L_2, while some other hard problem is reducible to the query in L_1 in a very general sense. The answer to the well-known hard problem with input x will be positive if and only if the answer to the query to a specific action description α together with $s(x)$ will be positive in L_1. If there existed a polynomial-size translation from L_1 into L_2, we would obtain a non-uniform algorithm with the complexity as that of the query in L_2.

Proposition 6.29. *If* $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ *then there is no polynomial-size translation neither from* **NNFAT**$^{\mathbf{T}}$ *into* **O-PDDL**$^{\mathbf{T}}$ *nor from* **NNFAT**$^{\mathbf{C}}$ *into* **O-PDDL**$^{\mathbf{C}}$.

PROOF. Recall the encoding of 3-CNF's from Notation 6.4. Consider the **NNFAT** action description β_n over the variables $X_n \cup P_n = \{x_1, \ldots, x_n, p_1, \ldots, p_{N_n}\}$.

$$\beta_n := \bigwedge_{i=1}^{N_n} (p_i \rightarrow (\bigvee_{\ell \in \gamma_i} \ell'))$$

Here ℓ is a literal over X_n, and ℓ' is obtained from ℓ by replacing the variable with its primed copy (e.g. if $\ell = \neg p_2$, $\ell' = \neg p_2'$). Intuitively, when β_n is executed in $s(\varphi)$ (with a 3-CNF φ over X_n) it creates nondeterministically an assignment to the X_n that satisfies φ, if there is one. Thus β_n is applicable in $s(\varphi)$ if and only if φ is satisfiable. Now if there existed a polynomial-size translation from **NNFAT** into **O-PDDL** there would be a non-uniform polynomial algorithm for checking satisfiability of 3-CNF's and thus $\mathsf{NP} \subseteq \mathsf{P/poly}$. The proof works for both trees and circuits, because applicability is in P for both **O-PDDL**$^{\mathbf{T}}$ and **O-PDDL**$^{\mathbf{C}}$ (Proposition 6.11). □

From Proposition 6.2 we conclude that if there existed a polynomial-size translation from **E-PDDL** into **O-PDDL** there would also exist a polynomial-size translation from **NNFAT** into **O-PDDL**, contradicting Proposition 6.29:

Corollary 6.30. *If* $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ *then there exists no polynomial-size translation neither from* **E-PDDL**$^{\mathbf{T}}$ *into* **O-PDDL**$^{\mathbf{T}}$ *nor from* **E-PDDL**$^{\mathbf{C}}$ *into* **O-PDDL**$^{\mathbf{C}}$.

In other words, the reason for the non-existence of a translation from **NNFAT** and **E-PDDL** into **O-PDDL** is the easy IS-APPLIC of the **O-PDDL**. Now we will prove that there is no polynomial-size translation from **O-PDDL** into **E-PDDL**. Since IS-SUCC and ENTAILS are equally hard for **O-PDDL** and **E-PDDL**, we will use ST to prove this separation result. We argue with the same reasoning as in the proof of Proposition 6.28:

Corollary 6.31. *If* $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ *then there is no polynomial-size translation neither from* **O-PDDL**$^{\mathbf{T}}$ *into* **E-PDDL**$^{\mathbf{T}}$ *nor from* **O-PDDL**$^{\mathbf{C}}$ *into* **E-PDDL**$^{\mathbf{C}}$.

PROOF. If **O-PDDL** was polynomial-size translatable into **E-PDDL** we could translate β_n from the proof of Proposition 6.27 into an **E-PDDL**-action description $f(\beta_n)$ and then check in linear time (Proposition 6.26) the ST-query whether $s(\varphi) \in f(\beta_n)(s(\varphi))$ to decide whether a 3-CNF φ is satisfiable, thus obtaining a nonuniform polynomial-time algorithm for 3-SAT. □

Since **NNFAT** is polynomial-time translatable into **E-PDDL** (Proposition 6.2), we conclude the following.

Corollary 6.32. *If* $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ *then there is no polynomial-size translation neither from* **O-PDDL**$^{\mathbf{T}}$ *into* **NNFAT**$^{\mathbf{T}}$ *nor from* **O-PDDL**$^{\mathbf{C}}$ *into* **NNFAT**$^{\mathbf{C}}$.

6.3 Transformations

Of course, if a language allows for arbitrary nesting of an operator then it trivially supports a transformation. For example, **NNFAT** allows to express CHOICE of α_1 and α_2 via $\alpha_1 \vee \alpha_2$. Therefore the following is a straightforward observation:

Corollary 6.33. Choice *is linear-time for both representations of* **NNFAT**, **E-PDDL** *and* **O-PDDL**.

In other cases, determining the complexity of a transformation seems to be not much easier than determining the succinctness of the language enriched with the corresponding operator. We start with NEGATION.

Proposition 6.34. *If* $\mathsf{coNP} \not\subseteq \mathsf{NP/poly}$ *then* Negation *is not polynomial-size in any (tree or circuit) representation of* **O-PDDL** *and* **E-PDDL**.

PROOF. Recall from Lemma 6.7 that

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

is such that $s(\varphi) \in \alpha_n^{\mathrm{sat},\sqcap}(\emptyset)$ holds if and only if φ is satisfiable. Now suppose that NEGATION is polynomial-size in **E-PDDL**$^{\mathbf{T}}$ or **E-PDDL**$^{\mathbf{C}}$. Then there exists a polynomial-sized equivalent $f(\alpha_n^{\mathrm{sat},\sqcap})$ of the negation of $\alpha_n^{\mathrm{sat},\sqcap}$. Thus $s(\varphi) \in f(\alpha_n^{\mathrm{sat},\sqcap})$ holds if and only if φ is unsatisfiable, and so there is a nonuniform NP-algorithm for a coNP-complete problem, because IS-SUCC is in NP for **E-PDDL** by Proposition 6.9.
The proof for **O-PDDL**$^{\mathbf{T}}$ and **O-PDDL**$^{\mathbf{C}}$ is exactly the same with $\alpha_n^{\mathrm{sat},\&}$ instead of $\alpha_n^{\mathrm{sat},\sqcap}$. □

Proposition 6.35. Negation *is linear time for both* **NNFAT**$^{\mathbf{C}}$ *and* **NNFAT**$^{\mathbf{T}}$.

PROOF. The negation of an NNF circuit can itself be transformed into an equivalent NNF circuit in linear time, using De Morgan's laws and elimination of double negations. □

The next results about SEQUENCE require a little preparation:

Lemma 6.36. *Let* $k \in \mathbb{N}$ *be fixed, and assume* $|P| \leq k$ *(i.e. the size of the scope of the described actions is fixed). Then* Is-Succ *can be solved in polynomial time for* **O-PDDL**$^{\mathbf{C}}$ *or* **E-PDDL**$^{\mathbf{C}}$.

PROOF. We assume the formulas in L to be represented as circuits, since this is the hardest setting. Let α be an L-action description over the variables $P := \{p_1, \ldots, p_k\}$. Next to each leaf node (which is itself an L-action description) we can write the corresponding set of effects in a given state s. The amount of possible effects m is exponential in k which is fixed, but m does not depend on the size of α. Similarly, each node pointing at subactions has a bounded number of effects ($\leq m$), which only depend on the possible effects of the direct subactions. It is thus possible to identify all possible effects of the root node of α in a bottom-up manner, and therefore to identify all possible successors of s (the maximal amount of successors does not depend on the size of α neither). It remains to check whether s' is one of them. □

Note that the same argumentation also works for IS-APPLIC and ENTAILS. We will see later that the argumentation from the proof works for every superlanguage $\langle L, I\rangle$ of **O-PDDL$^{\mathbf{C}}$** or **E-PDDL$^{\mathbf{C}}$** from this thesis.

Before proceeding to show that SEQUENCE is not polynomial-size for the minimally complete languages, 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, similar to Boolean conditioning. Given two disjoint sets of variables P, Q, a $(P \cup Q)$-action a, and an assignment $t \subseteq Q$ to the variables in Q, we define the *t-conditioning* of a to be the P-action $a_{|t}$ satisfying $\forall s \subseteq P\colon\ a_{|t}(s) = \{s' \mid (s' \cup t) \in \alpha(s \cup t)\}$.

Example 6.37. *For $P \coloneqq \{p_1, p_2\}$, $Q \coloneqq \{q_1, q_2\}$, and the* **E-PDDL** *action a described by*

$$\alpha \coloneqq \begin{array}{cc} & \big((p_1 \vee p_2 \vee q_2) \quad \rhd + p_2 \cup -q_2\big) \\ \sqcap & \big(\neg(p_1 \vee p_2) \wedge q_1 \quad \rhd + q_2\big) \\ \sqcap & \big((q_2 \vee \neg q_1) \quad \rhd - p_1\big) \end{array}$$

if $t \coloneqq \{q_1\}$, i.e. $q_1 = \top$, $q_2 = \bot$, then $\alpha(\{q_1\}) = \{\{q_1, q_2\}\}$, $\alpha(\{p_1, q_1\}) = \{\{p_1, q_1\}, \{p_1, p_2, q_1\}\}$, $\alpha(\{p_2, q_1\}) = \{\{p_2, q_1\}\}$ and $\alpha(\{p_1, p_2, q_1\}) = \{\{p_1, p_2, q_1\}\}$, so $a_{|t}$ can be described by

$$\alpha_{|t} \coloneqq \begin{array}{c} (p_1 \vee p_2) \rhd + p_2 \cup \varepsilon \\ \sqcap \neg(p_1 \vee p_2) \rhd \bot \end{array}$$

Lemma 6.38. *Let P and Q be disjoint sets of variables, α be an* **E-PDDL$^{\mathbf{T}}$** *(resp.* **E-PDDL$^{\mathbf{C}}$***) expression for a $(P \cup Q)$-action, and $t \subseteq Q$ be an assignment to the variables in Q. Then we can compute an* **E-PDDL$^{\mathbf{T}}$** *(resp.* **E-PDDL$^{\mathbf{C}}$***) expression $f(\alpha)$ for the t-conditioning of α in time polynomial in $|\alpha|$.*

PROOF. We define $f(\alpha)$ as follows:

1. if $\alpha = \pm p$ with $p \in P\colon\ f(\alpha) \coloneqq \alpha$
2. if $\alpha = +q$ with $q \in t$ or $\alpha = -q$ with $q \notin t\colon\ f(\alpha) \coloneqq \varepsilon$
3. if $\alpha = -q$ with $q \in t$ or $\alpha = +q$ with $q \notin t\colon\ f(\alpha) \coloneqq \bot$
4. if $\alpha = \beta \sqcap \gamma\colon\ f(\alpha) \coloneqq f(\beta) \sqcap f(\gamma)$
5. if $\alpha = \beta \cup \gamma\colon\ f(\alpha) \coloneqq f(\beta) \cup f(\gamma)$
6. if $\alpha = \varphi \rhd \beta\colon\ f(\alpha) \coloneqq \varphi_{|t} \rhd f(\beta)$

where $\varphi_{|t}$ denotes propositional conditioning. Clearly, $f(\alpha)$ can be computed in polynomial time. As for correctness, we show it by induction on the formula structure in a sligthly stricter form because we need this to deal with parallel execution. We claim that for all s, the effects of $f(\alpha)$ in s are exactly the effects $\langle E^+ \setminus Q, E^- \setminus Q\rangle$, where $\langle E^+, E^-\rangle$ is an effect of α in $s \cup t$ witnessing a transition of the form $(s \cup t, s' \cup t)$; clearly, this entails that $f(\alpha)$ indeed describes $\alpha_{|t}$.

1. Obvious since the only effect of α is $\langle\{p\}, \emptyset\rangle$ or $\langle\emptyset, \{p\}\rangle$ with $p \notin Q$, which leaves the values of the variables in Q unchanged.
2. Assume by symmetry $\alpha \coloneqq +q$ with $q \in t$. Then for all s, the only effect of α in $s \cup t$ is $\langle\{q\}, \emptyset\rangle$, which witnesses a transition to $s \cup t$ itself, and the only effect of $f(\alpha)$ in s is $\langle\emptyset, \emptyset\rangle$, as claimed.
3. In this case α sets a variable in Q inconsistently with t, hence there is no effect of α witnessing a transition of the form $(s \cup t, s' \cup t)$, hence $f(\alpha) = \bot$, which has no effect, is suitable.
4. By definition of $\sqcap$, for all P-states s the effects of $\beta \sqcap \gamma$ in $s \cup t$ witnessing a transition to some $s' \cup t$ are exactly the effects of the form $\langle E^+_\beta \cup E^+_\gamma, E^-_\beta \cup E^-_\gamma\rangle$, where $\langle E^+_\beta, E^-_\beta\rangle$ is an effect of β in $s \cup t$, $\langle E^+_\gamma, E^-_\gamma\rangle$ is an effect of γ in $s \cup t$, and they agree; moreover, this is if and only if $\langle E^+_\beta, E^-_\beta\rangle$ witnesses a transition from $s \cup t$ to some $s_\beta \cup t$, and so does $\langle E^+_\gamma, E^-_\gamma\rangle$ to some $s_\gamma \cup t$ (because neither

could modify t without the combination being inconsistent or itself modifying t). By the induction hypothesis, this is equivalent to $f(\beta)$ having the effect $\langle E^+_\beta \setminus Q, E^-_\beta \setminus Q\rangle$ in s, and similarly for $f(\gamma)$, and since $\langle E^+_\beta \setminus Q, E^-_\beta \setminus Q\rangle$ and $\langle E^+_\gamma \setminus Q, E^-_\gamma \setminus Q\rangle$ obviously agree, this is in turn equivalent to $f(\beta \sqcap \gamma)$, as defined in the statement, having the effect $\langle (E^+_\beta \cup E^+_\gamma) \setminus Q, (E^-_\beta \cup E^-_\gamma) \setminus Q\rangle$, as claimed.

5. Follows from the definition of $\cup$.

6. For $(s \cup t) \not\models \varphi$, we get $s \not\models \varphi_{|t}$ and hence, the only effect of β and $f(\beta)$ is $\langle \emptyset, \emptyset\rangle$; now for $(s \cup t) \models \varphi$, we get $s \models \varphi_{|t}$ and hence, α behaves as β and $f(\alpha)$ as $f(\beta)$, so that the claim follows from the induction hypothesis.

The proof is the same for circuits and for trees. □

We remark that the proof would not work for **O-PDDL** because if computing $\alpha_{|t}$ was polynomial-time for **O-PDDL** P-actions α we could solve ST (which is NP-complete by Proposition 6.27) in polynomial time as follows: for a given s set $Q = \{p_2, \ldots, p_n\}$ and $t := Q \cap s$. Then compute $\alpha_{|t}$ and check ST for $\hat{s} := s \cap \{p_1\}$. By construction of $\alpha_{|t}$ it holds that $\hat{s} \in \alpha_{|t}(\hat{s})$ if and only if $s = (\hat{s} \cup t) \in \alpha(\hat{s} \cup t) = \alpha(s)$.

Proposition 6.39. *If* NP $\not\subseteq$ P/poly *then* Sequence *is not polynomial-size for both the tree and circuit representations of* **NNFAT**, **O-PDDL** *and* **E-PDDL**.

PROOF. We start with the tree representations. Let $n \in \mathbb{N}$, and let φ be a 3-CNF formula over a set of variables $X_n := \{x_1, \ldots, x_n\}$. Recall from Notation 6.4 that we can encode φ over a set $P_n \subseteq \mathbb{P}$. Finally, let $p_{\text{sat}} \in \mathbb{P}$ be a fresh variable. We define three **E-PDDL** action descriptions over $P = X_n \cup P_n \cup \{p_{\text{sat}}\}$:

$$\beta_1 := \prod_{i=1}^{n} (+x_i \cup -x_i)$$
$$\beta_2 := (\chi_n \rhd +p_{\text{sat}}) \sqcap (\neg\chi_n \rhd -p_{\text{sat}})$$
$$\beta_3 := \prod_{i=1}^{n} -x_i$$

where χ_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, β_1 guesses an assignment to $\mathrm{V}(\varphi)$, β_2 sets p_{sat} according to whether φ is satisfied by the assignment to the x_i in the current state, and β_3 resets all guessed variables to false. Suppose that SEQUENCE is polynomial-size in **E-PDDL**. Then there must exists a polynomial-size **E-PDDL** action description δ^{sat}_n with $s' \in \delta^{\text{sat}}_n(s) \Leftrightarrow \exists\hat{s}\exists\tilde{s} : \hat{s} \in \beta_1(s), \tilde{s} \in \beta_2(\hat{s})$ and $s' \in \beta_3(\tilde{s})$. Note that δ^{sat}_n depends on n but not on φ, and that the size of δ^{sat}_n is polynomial in n because the sizes of β_1, β_2 and β_3 are polynomial in n.

We observe that $s(\varphi) \cup \{p_{\text{sat}}\}$ is a δ^{sat}_n-successor of $s(\varphi)$ if and only if φ is satisfiable, because it is an action which nondeterministically guesses an assignment and checks whether it satisfies φ. Hence the following decision problem is NP-hard:

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

Let φ be a 3-CNF formula over n variables. Since δ^{sat}_n is in **E-PDDL**, we can apply Lemma 6.38 with $Q := P_n \cup \{x_1 \ldots, x_n\}$ 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^{sat}_n-sucessor of $s(\varphi)$, that is, if and only if φ is satisfiable. Now since this expression has only one variable, it follows from Lemma 6.36 that successorship can be decided in polynomial time, implying NP $\subseteq$ P/poly.

We can reuse the proof for **NNFAT**$^{\mathbf{T}}$ and **NNFAT**$^{\mathbf{C}}$: observe that

- $\bigsqcap_{i=1}^{n}(+x_i \cup -x_i)$ can be expressed in **NNFAT**$^{\mathbf{T}}$ (and hence in **NNFAT**$^{\mathbf{C}}$) as $\left(\bigwedge_{j=1}^{N_n}\left((p_j \wedge p'_j) \vee (\neg p_j \wedge \neg p'_j)\right)\right) \wedge \left((p_{\text{sat}} \wedge p'_{\text{sat}}) \vee (\neg p_{\text{sat}} \wedge \neg p'_{\text{sat}})\right)$ (that is, tacitly reassign each x_i to an arbitrary value, and leave the other variables unchanged)
- $(\chi_n \rhd +p_{\text{sat}}) \sqcap (\neg\chi_n \rhd -p_{\text{sat}})$ as $\left(\bigwedge_{i=1}^{n}\left((x_i \wedge x'_i) \vee (\neg x_i \wedge \neg x'_i)\right)\right) \wedge \left(\bigwedge_{j=1}^{N_n}\left((p_j \wedge p'_j) \vee (\neg p_j \wedge \neg p'_j)\right)\right) \wedge \left((\neg\chi_n \vee p'_{\text{sat}}) \wedge (\chi_n \vee \neg p'_{\text{sat}})\right)$
- $\bigsqcap_{i=1}^{n} -x_i$ as $\left(\bigwedge_{i=1}^{n} \neg x'_i\right) \wedge \left(\bigwedge_{j=1}^{N_n}\left((p_j \wedge p'_j) \vee (\neg p_j \wedge \neg p'_j)\right)\right) \wedge \left((p_{\text{sat}} \wedge p'_{\text{sat}}) \vee (\neg p_{\text{sat}} \wedge \neg p'_{\text{sat}})\right)$

We recall that the "conditioning" as in Lemma 6.38 can be performed on both representations of **NNFAT** in polynomial-time since **NNFAT** is polynomial-time translatable into **E-PDDL** by Proposition 6.2.

As for **O-PDDL**, consider the following **O-PDDL** actions with the scope $P = X_n \cup P_n \cup \{p_{\text{sat}}\}$, as before: $\hat{\beta}_1 := \&_{i=1}^{n}(+x_i \cup -x_i)$ and $\hat{\beta}_2 := \neg\chi_n \rhd \bot$ with χ_n as in the proof for **E-PDDL**. Suppose that SEQUENCE is polynomial-size in **O-PDDL**, then there exists β_n^{sat} equivalent to the sequential execution of $\hat{\beta}_2$ after $\hat{\beta}_1$, and it holds for all 3-CNF's φ over X_n: φ is satisfiable if and only if β_n^{sat} is applicable in $s(\varphi)$ (intuitively, β_n^{sat} first guesses an assignment to X_n and then fails if this assignment does not satisfy φ). Since β_n^{sat} is polynomial-size, we obtain a non-uniform polynomial-time algorithm for 3-SAT, because deciding applicability is polynomial-time for both the tree and circuit representations of **O-PDDL** by Proposition 6.11.

The proof is the same for trees and circuits. □

Finally we turn to EXTRACT-PRECOND.

Proposition 6.40. *If* NP $\not\subseteq$ P/poly *then* Extract-Precond *is not polynomial-size for* **NNFAT** *nor for* **E-PDDL** *under neither the tree nor the circuit representation.*

PROOF. Recall that existential forgetting for a propositional language L is the problem of computing, given a formula φ in L over variables $x_1,\ldots,x_k,y_1,\ldots,y_\ell$, a formula in L equivalent to $\exists x_1,\ldots,x_k : \varphi$ (Darwiche and Marquis, 2002). It is easy to see for **NNFAT** that SEQUENCE is a special case of existential forgetting; indeed, a possible description for $\alpha;\beta$ (recall from Chapter 4 that this is how we denote the sequential execution of β after α) over the state variables P would be $\exists p''_1,\ldots,p''_n : \alpha(p_1,\ldots,p_n,p''_1,\ldots,p''_n) \wedge \beta(p''_1,\ldots,p''_n,p'_1,\ldots,p'_n)$. We show that polynomial-size EXTRACT-PRECOND implies polynomial-size existential forgetting.

Let $x_1,\ldots,x_k,y_1,\ldots,y_\ell$ and φ be as above. Let $n := \max(k,\ell)$ and $x_{k+1},\ldots,x_n,y_{\ell+1},\ldots,y_n$ be dummy variables. Clearly, φ can be seen as a formula over variables $x_1,\ldots,x_n,y_1,\ldots,y_n$, and $\exists x_1,\ldots,x_k : \varphi$ is logically equivalent to $\exists x_1,\ldots,x_n\varphi$. We define P to be $\{p_1,\ldots,p_n\}$, and define α to be the action description obtained from φ by replacing x_i by p'_i and y_i by p_i, for all i.

Now it is easy to see that if we can compute a polynomial-size NNF formula ψ with $s \models \psi \iff \alpha(s) \neq \emptyset$ we can compute from it an NNF formula equivalent to $\exists x_1,\ldots,x_n : \varphi$ in polynomial time by replacing each p_i $(i = 1,\ldots,\ell)$ by y_i, and setting each p_i $(i = \ell+1,\ldots,n)$ to $\bot$ (arbitrarily, since this is a dummy variable).

To conclude, if EXTRACT-PRECOND was polynomial-size, so would be existential forgetting, and thus SEQUENCE. But SEQUENCE is not polynomial-size if NP $\not\subseteq$ P/poly (Proposition 6.39).

Now, if EXTRACT-PRECOND was polynomial-size for **E-PDDL**, then it would be polynomial-size for **NNFAT**, too, since the latter is polynomial-time translatable into the former by Proposition 6.2. □

A consequence of this proof is that existential forgetting is not polynomial-size for NNF formulas if NP $\not\subseteq$ P/poly. This is a stronger version of the statement than the one in (Lang et al., 2003, Proposition 23), where the complexity-theoretic assumption was NP $\cap$ coNP $\not\subseteq$ P/poly.

Proposition 6.41. Extract-Precond *is linear-time in* **O-PDDL**$^{\mathbf{T}}$ *and* **O-PDDL**$^{\mathbf{C}}$.

PROOF. We do the proof indirectly, i.e. for an **O-PDDL** action α we first define an NNF formula $\varphi(\alpha)$ such that α is not applicable in s if and only if $s \models \varphi(\alpha)$.

$\varphi(\alpha)$ is recursively obtained from α by replacing each & by $\vee$, each $\cup$ by $\wedge$, each $\psi \rhd \beta$ by $\psi \wedge \varphi(\beta)$ and each $\pm q$ by $\bot$. This is because for $\beta \,\&\, \gamma$ it is enough for one branch to fail to cause the failure of the whole subaction, and for $\beta \cup \gamma$ we need both branches to fail. For the non-applicability of a conditional execution $\psi \rhd \beta$ we need the condition ψ to be true and β to be not applicable. $\pm q$ never fails, so $\varphi(\pm q)$ must be $\bot$. Thus α is applicable in s if and only if $s \models \neg\varphi$. Since the construction of φ is linear-time for both **O-PDDL**$^{\mathbf{T}}$ and **O-PDDL**$^{\mathbf{C}}$, we conclude the claim. □

6.4 Conclusion

The results of Chapter 6 are summarized in Table 6.1. We recall that **NNFAT** is polynomial-time translatable into **E-PDDL**, but the converse does not hold even for polynomial-size translations, and that **O-PDDL** and **E-PDDL** are succinctness-incomparable (all succinctness results hold under the assumption that $\mathsf{NP} \not\subseteq \mathsf{P/poly}$).

One conclusion is that converting an **E-PDDL** specification into an **NNFAT** representation necessarily yields an explosion in some cases. Hence it is worth developing planners which tackle directly the PDDL specification, or, dually, to read specifications in **NNFAT**, so as to avoid the conversion. The latter may indeed make sense in some settings, since **NNFAT** is a declarative language (where it is easy to specify that the action sets p_1 to the same value as p_2, for instance), quite complementary to the imperative **PDDL**.

Since **E-PDDL** and **O-PDDL** are succinctness-incomparable, the choice of a particular semantics for parallel execution could for example depend on the notion of plan one wants to find. Avoiding cycles could be more difficult when working with **O-PDDL** due to NP-hardness of ST, but this is compensated by the polynomial-time applicability checking and extracting of explicit NNF preconditions.

The advantage of **NNFAT** is that it is the only language with tractable IS-SUCC, but this query is not a central one in planning, contrary to IS-APPLIC and ENTAILS.

A general conclusion that we can make from the fact that the complexities do not differ for the tree and circuit representations is that whenever possible one should implement the planners to be able to work with the more compact circuit representations.

As for the more theoretical conclusions, most of the results themselves are not very surprising, for example, the hardness of applicability in **NNFAT** and **E-PDDL** comes from the "hidden" possible inconsistencies of positive and negative assignments to the same variable. In **O-PDDL** there is no such problem due to a "hierarchy" where positive assignments always win, and thus the only source of inapplicability is the explicit $\bot$ symbol. What is surprising is the simplicity of the expression $\alpha_n^{\mathrm{sat},\sqcap}$ which involves only positive assignments, no conditions and is of bounded depth. The latter fact is a motivation to study do not of **E-PDDL** with bounded depth of action descriptions, suspecting that these restrictions will remain highly expressive and difficult to reason about (we will see in the next chapter that this is indeed the case). An interesting result is that ST is difficult for **O-PDDL**, but easy for **E-PDDL**. We will see in Chapter 8 that even some of the very expressive extensions of **E-PDDL** still allow for tractable ST.

The complexity and succinctness pictures in this chapter are complete, but we can imagine many further research directions based on what we learned here: for example, the study of the **PDDL**-like action language which allows for both & and $\sqcap$ connectives. Another interesting question naturally arising from our results is which restrictions made on **E-PDDL** would allow to translate it into **NNFAT** (at least with a polynomial-size translation).

Altogether this chapter presents two of the three proof ideas which we use repeatedly in this thesis. The first one is for succinctness separation results, which is to find a query which is more difficult in one language than in the other, and to define a family of instances of this problem which toghether constitue a non-uniform algorithm for some complete problem, and argue that a polynomial-size translation would imply the existence of a "simpler" non-uniform algorithm for the same problem which is highly unlikely. The second idea is the instanciation and reduction of the circuit for showing membership results.

Queries			
Query/Transformation	**O-PDDL**	**E-PDDL**	**NNFAT**
IS-SUCC	NP-complete	NP-complete	linear time
IS-APPLIC	linear time	NP-complete	NP-complete
ENTAILS	coNP-complete	coNP-complete	coNP-complete
ST	NP-complete	linear time	linear time
IS-DET	polynomial-time	coNP-complete	coNP-complete
IS-MON	polynomial-time	coNP-complete	coNP-complete
Transformations			
CHOICE	linear time	linear time	linear time
NEGATION	○	○	linear time
SEQUENCE	○	○	○
EXTRACT-PRECOND	linear time	○	○

Table 6.1: Complexity results for minimally complete languages. ○ means that under some complexity-theoretic assumption the transformation is not polynomial-size. Results hold for both the tree and the circuit representations.

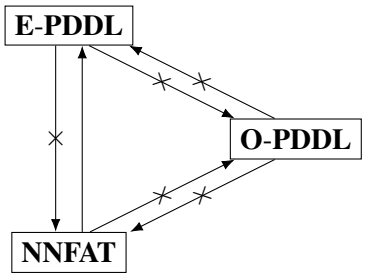


Figure 6.1: Succinctness results for minimally complete languages. An arc from L_1 to L_2 means that L_1 can be translated into L_2 in polynomial time (hence also in polynomial size). A crossed out arc from L_1 to L_2 means that under some complexity-theoretic assumption there exists not even a polynomial-size translation from L_1 into L_2. These relations hold whether the two languages are in the tree representation, or they are both in circuit representation.

Part III

Variants of Basic Languages

CHAPTER 7

Restrictions of Minimally Complete Languages

Contents

We have seen in the previous chapter that already very simple action descriptions with a bounded depth serve (such as $\alpha_n^{\text{sat},\&}$) as witnesses for nontractability of queries or impossibility of polynomial-size transformations. In this chapter we will consider restrictions of **O-PDDL** and **E-PDDL** whose grammar is so restrictive that $\alpha_n^{\text{sat},\&}$ (resp. $\alpha_n^{\text{sat},\sqcap}$) are not part of the language anymore.

We concentrate on restrictions of imperative minimally complete languages **O-PDDL** and **E-PDDL** in this chapter. The reason is that the only declarative language we study in our work is **NNFAT**, and its restrictions like CNF, DNF, BDD, DNNF and others have been well-studied already for a long time (Darwiche and Marquis, 2002).

7.1 Nondeterministic Conditional STRIPS

As we have already proven, we can't simply drop any connective from the grammar of **O-PDDL** and **E-PDDL** without losing completeness. But there are other ways to restrict the syntax. One can syntactically restrict the depth of the expression, enforce a specific order of connectives (like in CNF and DNF), assume the variables to always appear in a fixed order throughout the action description (like in OBDD's), or assume that certain subformulas don't share variables (like in DNNF).

The original relational **PDDL** can be considered a generalization of the Stanford Research Institute Problem Solver language **STRIPS** (Fikes and Nilsson, 1971). **STRIPS** can be seen as a syntactic restriction of grounded **PDDL** (i.e. the action descriptions must have a concrete structure "if condition then effect and if other condition then other effect…"). Thus we can define in an analogous way restrictions for our abstract variants of **PDDL** and obtain respectively two variants of nondeterministic conditional STRIPS. We call them **O-NCSTRIPS** and **E-NCSTRIPS**, with "C" for "conditional" (contrary to unconditional STRIPS by Fikes and Nilsson (1971) we allow for effects to be conditional, i.e. of the form $\varphi \rhd \ldots$) and "N" for "nondeterministic".

We recall that **O-PDDL** and **E-PDDL** relate to the general definitions of effects by Rintanen (2003), and it is not surprising that **O-NCSTRIPS** and **E-NCSTRIPS** resemble the 1NF-normal form (1NF

meaning "unary nondeterminism") of effects defined there. An effect (in Rintanen's definition) is in 1NF-normal form if it is either deterministic (without $\cup$) or a nondeterministic choice of deterministic effects. However our definitions here are a bit more general, since an **O-NCSTRIPS** or **E-NCSTRIPS** action can be seen as a parallel execution of 1ND-effects.

Definition 7.1 (**O-NCSTRIPS**)**.** An **O-NCSTRIPS** *action description* is an **O-PDDL** expression of the form

$$\mathop{\&}_{i=0}^{n}\Big(\varphi_i \rhd \big((\ell_i^{1,1} \,\&\, \ldots \,\&\, \ell_i^{1,j_1}) \cup \ldots \cup (\ell_i^{k_i,1} \,\&\, \ldots \,\&\, \ell_i^{k_i,j_{k_i}})\big)\Big)$$

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

In words, an **O-NCSTRIPS** 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.

Analogously restricting **E-PDDL** we obtain the language **E-NCSTRIPS**.

Definition 7.2 (**E-NCSTRIPS**)**.** An **E-NCSTRIPS** *action description* is an **E-PDDL** expression of the form

$$\prod_{i=0}^{n}\Big(\varphi_i \rhd \big((\ell_i^{1,1} \sqcap \ldots \sqcap \ell_i^{1,j_1}) \cup \ldots \cup (\ell_i^{k_i,1} \sqcap \ldots \sqcap \ell_i^{k_i,j_{k_i}})\big)\Big)$$

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

Note that for both languages now we need ε to be a possible atomic action, otherwise we can't express ε as a possible effect via $(p \wedge \neg p) \rhd +p$ because conditions can occur only directly below the root. Also note that since the root operator is always a simultaneous execution ($\&$ or $\sqcap$), we can put an **NNF** precondition φ for the action α via $(\neg\varphi \rhd \bot)\,\&\,\alpha$ and $(\neg\varphi \rhd \bot) \sqcap \alpha$.

The following notation will be useful to illustrate our example and to write some expressions in proofs more compactly.

Notation 7.3. For a state s let ψ_s denote the formula $(\bigwedge_{p\in s} p) \wedge (\bigwedge_{p\notin s} \neg p)$. For an NNF formula φ we denote by φ' the formula obtained by replacing all variables p in φ by their primed copies p'. The expression φ' describes the action which maps every state s to an arbitrary model s' of φ. In particular, $\psi'_{s'}$ describes the deterministic action mapping every state to s'.

Example 7.4. *The languages* **O-NCSTRIPS** *and* **E-NCSTRIPS** *are complete, since any action a can at least be represented by one condition for each state s (satisfied only by s), associated to either (1) a choice ($\cup$) between some "conjunctions" (via $\&$ or $\sqcap$) 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. For example, if $P := \{p_1,p_2,p_3\}$ and the P-states $s' := \{p_1,p_2\}, s'' := \{p_3\}, s''' := \{p_1\}$ are successors of $s := \{p_1\}$ via the action a, then a possible* **E-NCSTRIPS** *description α of a could be*

$$\alpha = \ldots \sqcap \Big((p_1 \wedge \neg p_2 \wedge \neg p_3) \rhd \big((+p_2) \cup (+p_3 \sqcap -p_1) \cup (\varepsilon)\big)\Big) \sqcap \cdots$$

Formally, when given an action a we can compute an **E-NCSTRIPS***-representation α of a (with a given scope P) as follows (with ψ_s as in Notation 7.3):*

$$\alpha := \prod_{s\subseteq P} \psi_s \rhd \begin{cases} \bigcup_{s'\in a(s)} \Big((\bigsqcap_{p\in s'} +p) \sqcap (\bigsqcap_{p\notin s'} -p)\Big) & \text{if } a(s_i) \neq \emptyset \\ \bot & \text{if } a(s) = \emptyset \end{cases}$$

The **O-NCSTRIPS** *representation of a is obtained analogously with $\&$ instead of $\sqcap$.*

Results for Complete Restrictions

Since both **O-NCSTRIPS** and **E-NCSTRIPS** are complete, it is not surprising that they have a lot in common with their "parent" superlanguages **O-PDDL** and **E-PDDL**. In this subsection we give complexity results for queries and partial results for transformations and succinctness.

Since **O-NCSTRIPS** and **E-NCSTRIPS** are 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 complexity results in this work, we only write **O-NCSTRIPS** and **E-NCSTRIPS** without specifying the representation, and we say that **E-NCSTRIPS** is a sublanguage of **E-PDDL**$^{\mathbf{T}}$ or **E-PDDL**$^{\mathbf{C}}$ (and **O-NCSTRIPS** is a sublanguage of **O-PDDL**$^{\mathbf{T}}$ and **O-PDDL**$^{\mathbf{C}}$), meaning each time the corresponding version of **E-NCSTRIPS/O-NCSTRIPS**.

7.1.1 Queries

The membership results follow directly from **O-NCSTRIPS** being a sublanguage of **O-PDDL** and **E-NCSTRIPS** being a sublanguage of **E-PDDL**:

Corollary 7.5. *For* **O-NCSTRIPS**, Is-Applic *can be answered in linear time and* Is-Det *and* Is-Mon *can be answered in polynomial time,* Is-Succ *and* ST *are in* NP *and* Entails *is in* coNP.

For **E-NCSTRIPS**, ST *can be answered in linear time,* Is-Succ *and* Is-Applic *are in* NP *and* Entails, Is-Det *and* Is-Mon *are in* coNP.

For the hardness results we will reuse the proof ideas from the previous chapter, but adjusted to the quite restrictive syntax of **O-NCSTRIPS** and **E-NCSTRIPS**.

Notation 7.6. Recall from Notation 6.4 (page 39) that we can encode 3-CNFs over X_n with a polynomial number of propositional variables P_n. For a set of 3-clauses Γ we write $\Gamma \subseteq_\ell^5 \Gamma_n$ if $\Gamma \subseteq \Gamma_n$, $|\Gamma| \leq 5$ and each clause in Γ contains ℓ exactly once. In other words, this means that the literal ℓ occurs in the clauses in Γ at most 5 times. For each n, $|\{\Gamma \mid \Gamma \subseteq_\ell^5 \Gamma_n\}|$ (the amount of such Γ's) is polynomial in n.

Proposition 7.7. ST *is* NP*-complete for* **O-NCSTRIPS**.

PROOF. We recall Notation 7.6 and define the expression $\text{choose}_n(\ell) := \bigcup_{\Gamma \subseteq_\ell^5 \Gamma_n} (\mathbin{\&}_{\gamma_j \in \Gamma} + p_j)$. The size of $\text{choose}_n(\ell)$ is polynomial in n. Let β_n be the following **O-NCSTRIPS**-action:

$$\beta_n := \left(\mathop{\&}_{i=1}^{n} \top \rhd (\text{choose}_n(x_i) \cup \text{choose}_n(\neg x_i)) \right) \mathbin{\&} \left(\mathop{\&}_{j=1}^{N_n} \top \rhd -p_j \right)$$

Our claim now is similar to the hardness claim for **O-PDDL**: a 3-CNF formula φ with at most 5 occurences of each literal is satisfiable if and only if $s(\varphi) \in \beta_n(s(\varphi))$. The claim is easy to see because $\text{choose}_n(x)$ chooses nondeterministically a set of clauses of which each contains x as a literal and the total number of occurences of this literal is at most 5 (especially note that the empty set of clauses is a possible choice, too). Thus for each satisfiable 3-CNF φ with at most five occurences of each literal there exists an execution of β_n which sets exactly the variables of $s(\varphi)$ true and all other variables false. Conversely, if $s(\varphi) \in \beta(s(\varphi))$ then there exists an execution setting exactly the variables in $s(\varphi)$ to true and the others to false. This execution induces an assignment to the x variables which is a witness for the satisfiability of φ.

Now recall that deciding satisfiability of a 3-CNF with at most 5 occurences of a literal is still an NP-complete problem (Lemma 3.9, page 17, with $k = 5$), for which we found a reduction to ST in **O-NCSTRIPS**. NP-completeness follows from Corollary 7.5.

We remark that β_n can also generate 3-CNF's with more than 5 occurences of each literal, but this does no harm for the proof. □

Proposition 7.8. Is-Succ *is* NP*-complete for both* **O-NCSTRIPS** *and* **E-NCSTRIPS**.

PROOF. Since ST is a special case of IS-SUCC and it is already NP-hard for **O-NCSTRIPS**, IS-SUCC is NP-hard for **O-NCSTRIPS**, too.

For **E-NCSTRIPS**, consider the **E-NCSTRIPS** expression, $\gamma_n = \bigsqcap_{i=1}^{n} \top \rhd (\text{choose}_n(x_i) \cup \text{choose}_n(\neg x_i))$ (with choose_n as in the proof of Proposition 7.7). With argumentation analogous to that for β_n from the proof of Proposition 7.7 we see that a 3-CNF φ with at most 5 occurences of each literal is satisfiable if and only if $s(\varphi) \in \gamma_n(\emptyset)$ and thus IS-SUCC is NP-hard for **E-NCSTRIPS**.

NP-membership follows from Corollary 7.5. □

Proposition 7.9. Is-Applic *is* NP*-complete for* **E-NCSTRIPS**.

PROOF. Membership was stated in Corollary 7.5.

To show hardness for **E-NCSTRIPS**, we define the **E-NCSTRIPS** action ξ_n over the variables $X_n \cup P_n$ with X_n, P_n as in Notation 6.4 (page 39):

$$\xi_n := \bigsqcap_{p_i \in P_n} \left(p_i \rhd \left(\left(\bigcup_{x \in \gamma_i} +x \right) \cup \left(\bigcup_{\neg x \in \gamma_i} -x \right) \right) \right)$$

Intuitively, in $s(\varphi)$ the action ξ_n chooses a satisfying assignment (to one of the x-variables) for each clause γ_i of φ; clearly, a successful execution of ξ_n in $s(\varphi)$ corresponds to one satisfying assignment for φ, and each satisfying assignment corresponds to at least one successful execution of ξ_n. Thus a 3-CNF formula φ is satisfiable if and only if ξ_n is applicable in $s(\varphi)$, which gives a reduction from 3-SAT to IS-APPLIC. □

Proposition 7.10. Entails *is* coNP*-complete for* **O-NCSTRIPS** *and* **E-NCSTRIPS**.

PROOF. The action $\alpha = \&_{x_i \in X_n}(+x_i \cup -x_i)$ from the proof of Proposition 6.16 (page 42) entails a DNF φ if and only if φ is valid. α is already an **O-NCSTRIPS** (we can insert a trivial precondition $\top$ obtaining $\&_{x_i \in X_n} \top \rhd (+x_i \cup -x_i)$) action and therefore ENTAILS is coNP hard for **O-NCSTRIPS**.

For **E-NCSTRIPS** we know that IS-APPLIC is NP-hard and since its complement is reducible to ENTAILS (Lemma 4.7, page 23) we conclude that ENTAILS is coNP-hard for **E-NCSTRIPS**. □

Proposition 7.11. Is-Det *and* Is-Mon *are* coNP*-complete for* **E-NCSTRIPS**.

PROOF. The proofs for reducing IS-APPLIC to IS-DET and IS-MON for **E-PDDL** (Propositions 6.21 and 6.23) can be reused to show the same reduction for **E-NCSTRIPS**: an **E-NCSTRIPS**-action α is non-applicable in s if $\alpha \sqcap (\top \rhd (+q \cup -q))$ with a fresh variable q is deterministic in s. The $\top \rhd \ldots$ serves to satisfy the formal requirements of the syntax of **E-NCSTRIPS**. The reduction to IS-MON is obtained in the same manner. □

7.1.2 Succinctness

We now show the succinctness results, which are not surprising given the known succinctness relations for **E-PDDL** and **O-PDDL**. We remark that since **O-NCSTRIPS** is a sublanguage of **O-PDDL** and **E-NCSTRIPS** is a sublanguage of **E-PDDL**, the identity function is an obvious linear-time translation in both cases.

Proposition 7.12. *If* NP $\not\subseteq$ P/poly *then* **O-NCSTRIPS** *is neither polynomial-size translatable into* **E-NCSTRIPS** *nor into* **E-PDDL**.

Under the same assumption **E-NCSTRIPS** *is neither polynomial-size translatable into* **O-NCSTRIPS** *nor into* **O-PDDL**.

PROOF. The argumentation is exactly the same as for incomparability of **O-PDDL** and **E-PDDL**, but now we use the more complicated action descriptions: the **O-NCSTRIPS** action β_n from the proof of Proposition 7.7 and the **E-NCSTRIPS** action ξ_n from the proof of Proposition 7.9. Suppose that there exists a polynomial-size translation from **O-NCSTRIPS** into **E-PDDL**. Then by translating the actions

β_n from the proof of Proposition 7.7 we obtain a non-uniform polynomial-time algorithm for deciding satisfiability of 3-CNF's with at most 5 occurences of each literal, since ST can be decided in linear time in **E-PDDL**. Thus, if $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ then **O-NCSTRIPS** is not polynomial-time translatable neither into **E-PDDL** nor into its sublanguage **E-NCSTRIPS**.

Conversely, if there existed a polynomial-size translation from **E-NCSTRIPS** into **O-PDDL** we would obtain a non-uniform polynomial-time algorithm for deciding satisfiability of 3-CNF's by translating ξ_n from the proof of Proposition 7.9 (which is applicable in $s(\varphi)$ if and only if φ is satisfiable) into **O-PDDL** where applicability can tested in polynomial time. Hence if $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ then **E-NCSTRIPS** is neither polynomial-size translatable into **O-PDDL** nor its sublanguage **O-NCSTRIPS**. □

Proposition 7.13. *If* $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ *then both* **NNFAT**$^{\mathbf{T}}$ *and* **NNFAT**$^{\mathbf{C}}$ *are succinctness-incomparable with* **O-NCSTRIPS**.

PROOF. From the proof of Proposition 7.7 we see with our usual argumentation about succinctness (like in the proof of Proposition 6.28, page 45) that if there was a polynomial-size translation from **O-NCSTRIPS** into **NNFAT** then we would have a nonuniform polynomial-time algorithm for checking the satisfiability of 3-CNFs with at most 5 occurences of each literal, because IS-SUCC is linear-time in **NNFAT**. Conversely, since **NNFAT** is not polynomial-size translatable into **O-PDDL** (Proposition 6.29, page 46), it is not translatable into **O-NCSTRIPS** either. □

Proposition 7.14. *If* $\mathsf{NP} \not\subseteq \mathsf{P/poly}$*, there exists no polynomial-size translation from* **E-NCSTRIPS** *into* **NNFAT**$^{\mathbf{T}}$ *nor into* **NNFAT**$^{\mathbf{C}}$.

PROOF. Analogously to the proof of Proposition 6.28 (page 45) we argue with γ_n from the proof of Proposition 7.8 instead of $\alpha_n^{\mathrm{sat},\sqcap}$: if there was a polynomial-size translation from **E-NCSTRIPS** into either the tree or the circuit representation of **NNFAT** then there would be a polynomial-time non-uniform algorithm for checking the satisfiability of 3-CNFs with at most 5 occurences of each literal, implying $\mathsf{NP} \subseteq \mathsf{P/poly}$. □

The following separation result is unconditional (this is the only separation result in this thesis which does not rely on unproven assumptions about complexity classes).

Proposition 7.15. *There exists no polynomial-size translation of* **NNFAT**$^{\mathbf{C}}$ *into* **E-NCSTRIPS**.

PROOF. The majority function (which returns true if and only if at least half of its arguments are true) can be computed by a boolean circuit ψ of linear size and logarithmic depth (Muller and Preparata, 1975). Consider an action a over $P_n = \{p_1, \ldots, p_n\}$ which is applicable only in $s = \emptyset$ and in this state produces nondeterministically all s' with $|s'| \geq \frac{n}{2}$. Thus it is representable by a polynomial-size circuit NNF action theory ψ'_n. Now every **E-NCSTRIPS** representation of a can without loss of generality be assumed to be of the form $(\varphi_n \rhd \alpha_n) \sqcap (\neg\varphi_n \rhd \bot)$ with $\varphi_n = \neg p_1 \wedge \ldots \wedge \neg p_n$ and α_n being an unconditioned **E-NCSTRIPS** expression. By replacing $\sqcap$ by $\wedge$, $\cup$ by $\vee$, $+p$ by p and $-p$ by $\neg p$ we would then obtain a formula over P_n whose models are α_n-successors of $\emptyset$, thus obtaining a boolean circuit of bounded depth of size polynomial in n for the majority function, which contradicts a result from Hastad (1986).□

Since **NNFAT**$^{\mathbf{C}}$ is polynomial-time translatable into **E-PDDL**$^{\mathbf{C}}$ by Proposition 6.2 (page 38), we conclude:

Corollary 7.16. *There exists no polynomial-time translation from* **E-PDDL**$^{\mathbf{C}}$ *into* **E-NCSTRIPS**.

After all we leave three succinctness questions open in this work: whether **O-NCSTRIPS** is strictly less succinct than **O-PDDL** for any representation of **O-PDDL**, whether **E-NCSTRIPS** is strictly less succinct than **E-PDDL**$^{\mathbf{T}}$ and whether **NNFAT**$^{\mathbf{T}}$ is polynomial-size translatable into **E-NCSTRIPS**.

7.1.3 Transformations

As for the transormations, we can mostly reuse the argumentation from Chapter 6. The only big problem here is that for **O-NCSTRIPS** and **E-NCSTRIPS** the operator $\cup$ is not allowed to occur at the root and hence it is not obvious whether it is possible to perform polynomial-size CHOICE. For the other results:

Proposition 7.17. *If* $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ *and* $\mathsf{coNP} \not\subseteq \mathsf{NP/poly}$ *then* Extract-Precond, Negation *and* Sequence *are not polynomial-size for* **E-NCSTRIPS**.

PROOF. Consider the **E-NCSTRIPS** action over $X_n \cup P_n$

$$\xi_n := \bigsqcap_{p_i} \left(p_i \rhd \left(\left(\bigcup_{x \in \gamma_i} +x \right) \cup \left(\bigcup_{\neg x \in \gamma_i} -x \right) \right) \right)$$

and recall from the proof of Proposition 7.9 that ξ_n is applicable in $s(\varphi)$ if and only if φ is satisfiable. Hence if there was a polynomial-size NNF representation of the precondition ψ_n of ξ_n, we could check whether φ is satisfiable by checking $s(\varphi) \models \psi_n$, and thus we would obtain a non-uniform polynomial-time algorithm for 3-SAT. As for SEQUENCE and NEGATION, the proof is a modification of the proof of Propositions 6.34 and 6.39 in the same way as we did it for the Proposition 7.18 (i.e. for showing the impossibility of polynomial-size NEGATION we show that by expressing the negation of γ_n from the proof of Proposition 7.8 in **E-NCSTRIPS** we obtain a non-uniform NP-algorithm for deciding unsatisfiability of 3-CNF's with at most 5 occurences of each literal). □

Proposition 7.18. *If* $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ *and* $\mathsf{coNP} \not\subseteq \mathsf{NP/poly}$ *then* Negation *and* Sequence *are not polynomial-size for* **O-NCSTRIPS**. Extract-Precond *is linear-time for* **O-NCSTRIPS**.

PROOF. The proof for SEQUENCE is exactly the same as in Proposition 6.39 (page 49), because all the partial action descriptions there are already in **O-NCSTRIPS**. As for NEGATION, if it was possible with a polynomial increase in size then by computing the polynomial-size negation of β_n from the proof of Proposition 7.7 we would obtain a non-uniform NP-algorithm for a coNP-complete problem of unsatisfiability of 3-CNFs with at most five occurences of each literal. Finally, since EXTRACT-PRECOND is linear-time for **O-PDDL**, it is linear time for **O-NCSTRIPS**. □

7.2 Incomplete Restriction: Non-negative E-PDDL/O-PDDL

We will now turn to incomplete restrictions of **E-PDDL** and **O-PDDL**. As we have shown in Chapter 5, dropping any of the connectives from the grammar yields an incomplete restriction, but most of them are not very interesting because the loss in expressiveness is too big (e.g. after excluding the parallel composition we can only define actions that change at most one variable per execution).

A more interesting restriction of the grammar is to exclude a sort of elementary assignments. Since all basic queries are intractable for **E-PDDL**, it is interesting to consider its sublanguages which could allow to answer at least some queries in polynomial time. One such language is the language $\mathbf{NPDDL_{nf}}$ of "negation-free **E-PDDL**-expressions", generated by the grammar

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

Note that the grammar includes the $\bot$ symbol to allow for actions to fail. It is important to note that "negation-free" here refers only to the type of elementary assignments, i.e. assignments of the form $-p$ are not allowed. However, negative literals are still allowed in conditions φ. "Negation-freeness" does neither refer to the negation operators which we will define in the next chapter, nor to the possibility or impossibility to perform polynomial-size NEGATION.

The choice of the restriction to only positive effects is arbitrary, and the results of this section hold for the "addition-free" restriction of **E-PDDL** by duality.

$\mathbf{NPDDL_{nf}}$ is obviously not complete. It is easy to see that the actions which it can describe are exactly those α's satisfying: $\forall s, s'$: $s' \in \alpha(s) \Rightarrow s' \supseteq s$, i.e. monotone actions. Therefore the query

IS-MON for positive monotony is not sensible for **NPDDL$_{\mathbf{nf}}$**, and IS-MON for negative monotony boils down to non-applicability.

Another observation which we make is that all effects of **NPDDL$_{\mathbf{nf}}$** actions are of the form $\langle Q^+, \emptyset \rangle$, i.e. the negative part is always empty. Thus the condition for two effects to agree is always trivially satisfied, and there is no difference between the effects of $\alpha \sqcap \beta$ and $\alpha \,\&\, \beta$. Thus, although **NPDDL$_{\mathbf{nf}}$** was first motivated as a restriction of **E-PDDL**, it turns out to be a restriction of **O-PDDL** as well, so it could also be defined by the grammar

$$\alpha \ ::= \bot \mid +p \mid \varphi \rhd \alpha \mid (\alpha \cup \alpha) \mid \alpha \,\&\, \alpha$$

We have decided to use $\sqcap$ instead of & for convenience and to stick to the **DL-PPA**-like syntax.

We first give the complexity results for **NPDDL$_{\mathbf{nf}}$**.

Proposition 7.19. Is-Applic *can be answered in polynomial time for both* **NPDDL$^{\mathbf{T}}_{\mathbf{nf}}$** *and* **NPDDL$^{\mathbf{C}}_{\mathbf{nf}}$**.

PROOF. The result follows from polynomial-time solvability of IS-APPLIC for **O-PDDL** (Proposition 6.11). □

Proposition 7.20. Is-Succ *is* NP*-complete for* **NPDDL$^{\mathbf{T}}_{\mathbf{nf}}$** *and* **NPDDL$^{\mathbf{C}}_{\mathbf{nf}}$**.

PROOF. Membership is inherited from **E-PDDL** (Proposition 6.13). For hardness, we consider the action $\alpha_n^{\mathrm{sat},\sqcap} := \bigsqcap_{x \in X_n} \Big(\big(\bigsqcap_{\gamma_i \colon x \in \gamma_i} (+p_i \cup \varepsilon)\big) \cup \big(\bigsqcap_{\gamma_i \colon \neg x \in \gamma_i} (+p_i \cup \varepsilon)\big) \Big)$ from Notation 6.6. We recall that a 3-CNF is satisfiable if and only if $s(\varphi) \in \alpha_n^{\mathrm{sat},\sqcap}(\emptyset)$ (Lemma 6.7) and observe that $\alpha_n^{\mathrm{sat},\sqcap}$ is an **NPDDL$_{\mathbf{nf}}$** expression. Thus we have a polynomial-time reduction from 3-SAT to IS-SUCC in **NPDDL$^{\mathbf{T}}_{\mathbf{nf}}$**. □

Proposition 7.21. Entails *is* coNP*-complete for both representations of* **NPDDL$_{\mathbf{nf}}$**.

PROOF. Membership follows from the coNP-membership of ENTAILS for **E-PDDL** (Proposition 6.14). As for the hardness, we claim that an NNF formula φ over $X_n = \{x_1, \ldots, x_n\}$ is valid if and only if the action $\alpha := \bigsqcap_{x_i \in X_n} (+x_i \cup \varepsilon)$ entails φ in $s = \emptyset$. This is true because $\alpha(s) = \{s' \mid s' \subseteq X_n\}$ and therefore if α entails φ this means that every assignment to X_n is a model of φ. □

Proposition 7.22. Is-Det *can be answered in polynomial time for both representations of* **NPDDL$_{\mathbf{nf}}$**.

PROOF. We argue analogously to the proof of Proposition 7.19: **NPDDL$_{\mathbf{nf}}$** is a sublanguage of **O-PDDL** and hence the result follows from Proposition 6.20. □

The complexity of the last remaining query ST is inherited from the superlanguage **E-PDDL** (Proposition 6.26).

Corollary 7.23. ST *is linear time for both representations of* **NPDDL$_{\mathbf{nf}}$**.

The fact that IS-SUCC is still NP-hard for **NPDDL$_{\mathbf{nf}}$** despite its very restrictive manner raises the question whether unrestricted **E-PDDL** is more succinct than **NPDDL$_{\mathbf{nf}}$** for monotone actions.

Proposition 7.24. *If* NP $\not\subseteq$ P/poly *then there are families of monotone actions with a polynomial-size description in* **E-PDDL$^{\mathbf{T}}$**, *but not in* **NPDDL$^{\mathbf{T}}_{\mathbf{nf}}$** *nor even in* **NPDDL$^{\mathbf{C}}_{\mathbf{nf}}$**.

PROOF. Consider the encoding of 3-CNFs as in Notation 6.4. We define the action

$$\beta_n := \bigsqcap_{p_i} \Big(p_i \rhd \big((\bigcup_{x \in \gamma_i} +x) \cup (\bigcup_{\neg x \in \gamma_i} \neg x \rhd -x) \big) \Big)$$

We observe that this action is positively monotone because it can only assign a variable to $\bot$ if it was already false before. Then a 3-CNF φ over $X := \{x_1, \ldots, x_n\}$ is satisfiable if and only if β_n is applicable in $s(\varphi)$, because β_n simply assigns a variable to satisfy at least one literal in each clause in $s(\varphi)$. If we could find an equivalent description in **NPDDL$^{\mathbf{C}}_{\mathbf{nf}}$** then we could check applicability in polynomial time because of Proposition 7.19, so after translating the α_n into equivalent negation-free expressions of polynomial size we would obtain a non-uniform polynomial-time algorithm for checking the satisfiability of 3-CNFs. □

Remark 7.25. Every **O-PDDL** description of a monotone action can be transformed into an equivalent negation-free **O-PDDL**-expression in polynomial time. Suppose that the **O-PDDL** description α describes a monotone action. Then $\hat{\alpha}$ is negation-free and $\hat{\alpha} \equiv \alpha$, with $\hat{\alpha}$ obtained by replacing all nodes of the form $-p$ by ε, because we know that whenever a $-p$-node is executed, there is either a $\bot$ or a $+p$-node which is executed in parallel or p was already false, and therefore $-p$ acts like ε here.

Hence the statement of Proposition 7.24 cannot be transferred to **O-PDDL**.

As for the succinctness, it doesn't make much sense to compare $\mathbf{NPDDL_{nf}}$ to complete languages in our setting because $\mathbf{NPDDL_{nf}}$ is not complete itself. However, the result of Proposition 7.24 can be seen as a succinctness result in the framework of heterogeneous compilation (Fargier et al., 2013).

Now we consider the transformations. The first one is straightforward because $\cup$ is part of the grammar of $\mathbf{NPDDL_{nf}}$.

Corollary 7.26. Choice *is linear-time in* $\mathbf{NPDDL_{nf}}$.

Proposition 7.27. Extract-Precond *can be carried out in linear time in both representations of* $\mathbf{NPDDL_{nf}}$.

PROOF. EXTRACT-PRECOND is polynomial-time for **O-PDDL** (Proposition 6.41, page 50) and hence for $\mathbf{NPDDL_{nf}}$. □

Proposition 7.28. *If* coNP $\not\subseteq$ NP/poly *then* Negation *is not polynomial-size for* $\mathbf{NPDDL^{C}_{nf}}$ *neither for* $\mathbf{NPDDL^{T}_{nf}}$.

PROOF. Since $\alpha_n^{\text{sat},\sqcap}$ is an $\mathbf{NPDDL_{nf}}$ action description, we can reuse the proof of Proposition 6.34 (page 47). If polynomial-size NEGATION was possible for $\mathbf{NPDDL_{nf}}$ then by using the representation of the negation of $\alpha_n^{\text{sat},\sqcap}$ for checking successorship we would obtain a non-uniform NP-algorithm for the coNP-complete problem of unsatisfiability of 3-CNFs. □

Proposition 7.29. *If* NP $\not\subseteq$ P/poly *then* Sequence *is not polynomial-size for* $\mathbf{NPDDL_{nf}}$.

PROOF. Consider the set of state variables $P_n \cup X_n \cup \{p_{\text{sat}}\} = \{p_1, \ldots, p_{N_n}, x_1, \ldots, x_n, p_{\text{sat}}\}$ with P_n as in Notation 6.4 (page 39) and the actions $\alpha_1 := \bigsqcap_{x_i \in X_n}(+x_i \cup \varepsilon)$, $\alpha_2 := \chi_n \rhd +p_{\text{sat}}$ with $\chi_n := \bigwedge_{i=1}^{N_n}(\neg p_i \vee \bigvee_{\ell \in \gamma_i} \ell)$ like in the proof of Proposition 6.39 (page 49) and $\alpha_3 := \neg p_{\text{sat}} \rhd \bot$. When executed in $s(\varphi)$, the sequence of actions $\alpha := \alpha_1 ; \alpha_2 ; \alpha_3$ (recall that we use this notation for "first do α_1, then do α_2, then do α_3") nondeterministically creates any possible assignment to the X_n variables, then assigns $\top$ to p_{sat} if the 3-CNF φ encoded by the initial values of the P_n-variables is satisfied, and then fails if p_{sat} is false, i.e. if the assignment made to X_n does not satisfy φ. If there was an $\mathbf{NPDDL_{nf}}$-action description $f(\alpha)$ of polynomial size and with the same semantics we could check whether an arbitrary 3-CNF φ is satisfiable by checking whether $f(\alpha)$ is applicable in $s(\varphi)$. Since applicability in $\mathbf{NPDDL_{nf}}$ is in P even for circuits (Proposition 7.19), we would obtain a non-uniform P-algorithm for an NP-complete problem. □

7.3 Conclusion

We have studied two complete restrictions of minimally complete imperative languages, **O-NCSTRIPS** and **E-NCSTRIPS**. We have shown that despite their very restrictive syntax they don't differ from their "parent languages" in terms of complexity, and thus the only reason to prefer **O-NCSTRIPS** over **O-PDDL** is that some planner is designed to reason only with representations of a certain structure. A positive side effect of the high complexity of these two representations is that they are complete (which is not very surprising, though, because CNF and DNF are even more restrictive in their syntax and they are still complete subclasses of the class of all NNF formulas).

We also studied a syntactic restriction of **E-PDDL** to only positive assignments and have observed two interesting facts: first, in this case the semantics of $\sqcap$ and & act in the same way, so in order to show their "full power" these connectives need a more expressive grammar. The other observation is that using negative assignments contributes to consiseness of representations even for monotone actions. The price

Query/Transformation	**O-NCSTRIPS**	**E-NCSTRIPS**	**NPDDL$_{nf}$**
Queries			
IS-SUCC	NP-complete	NP-complete	NP-complete
IS-APPLIC	linear time	NP-complete	linear time
ENTAILS	coNP-complete	coNP-complete	coNP-complete
ST	NP-complete	linear time	linear time
IS-DET	polynomial time	coNP-complete	polynomial time
IS-MON	polynomial time	coNP-complete	trivial/polynomial time
Transformations			
CHOICE	?	?	linear time
NEGATION	○	○	○
SEQUENCE	○	○	○
EXTRACT-PRECOND	linear time	○	linear time

Table 7.1: Complexity results for **O-NCSTRIPS**, **E-NCSTRIPS** and **NPDDL$_{nf}$**. ○ means that under some complexity-theoretic assumption the transformation is not polynomial-size. "?" means that the answer is yet unknown (but we suspect that it will be ○ after all). Recall that the answer to IS-MON is trivially "yes" for testing positive monotony.

to pay is the complexity of IS-APPLIC, and although it is hard to imagine a setting where we don't need to check applicability, this result is interesting from the theoretical point of view: sometimes a seemingly useless feature of a language can be useful from an unexpected point of view.

A natural continuation of the research of this chapter would be excluding the $\bot$ symbol from **O-PDDL**, which would yield actions that are always applicable.

Complexity results of this chapter are summarized in Table 7.1. The question whether any of the **STRIPS**-like restrictions supports polynomial-size CHOICE is still open, but we suspect that none of them does. It is worth mentioning that we tried various families of actions to prove this claim, but did not succeed even with quite technical constructions, which indicates that for expressing real-life problems **O-NCSTRIPS** and **E-NCSTRIPS** could be expressive enough. We don't give a succinctness diagram at this point because the diagram would not be illustrative for **O-NCSTRIPS** and **E-NCSTRIPS** being succinctness-incomparable and **NPDDL$_{nf}$** being their incomplete sublanguage.

CHAPTER 8

Extensions of **E-PDDL** and **O-PDDL**

Contents

In this chapter we will extend the imperative languages **O-PDDL** and **E-PDDL** by three connectives: the sequence connective ;, action negation connective $\neg_{\min}$ and action conjunction connective $\wedge$. These connectives will be added to the grammar without any further restriction on the action descriptions (contrary to the previous chapter). Thus it is again relevant to consider a tree and a circuit representation for each language.

8.1 E-PDDL and O-PDDL with Sequential Execution

The language **E-PDDL**, as we said before, is motivated to resemble a restricted version of the *dynamic logic of parallel propositional assignments* **DL-PPA**, and it has been shown that in **DL-PPA**, planning tasks can be expressed very compactly (Herzig et al., 2019). The grammar of **DL-PPA** consists of *(state) formulas* φ and *(program) formulas* π. It can be given in BNF as follows:

$$\pi ::= p \leftarrow \varphi \mid \pi\,;\pi \mid \pi \cup \pi \mid \pi \sqcap \pi \mid \pi \sqcup \pi \mid \pi^* \mid \varphi?$$
$$\varphi ::= p \mid \top \mid \bot \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\pi\rangle\varphi$$

The expression $p \leftarrow \varphi$ means that p is assigned to the value given by φ in the current state. **DL-PPA** itself is an extension of **DL-PA** which has elementary assignments of the form $+p$ and $-p$ instead of $p \leftarrow \varphi$ in its original definition (Balbiani et al., 2013). In our setting, however, such details as the chosen kind of elementary assignments play a very important role, and since we are originally motivated by **PDDL** and **STRIPS** we do not consider assignments $p \leftarrow \varphi$ for now. For **DL-PA** and **DL-PPA** such details are not crucial, because they are obviously complete and for the moment the studies about them do not concentrate on succinctness issues in our strict definition. For example, translations between them are allowed to introduce auxiliary variables because states in **DL-PA/DL-PPA** are assignments to a countably infinite set of variables.

Hence the differences between **E-PDDL** and **DL-PPA** are the following:

- Elementary assignments: $\pm p$ in **E-PDDL** and $p \leftarrow \varphi$ in **DL-PPA**,
- State formulas: NNF in **E-PDDL**, with modal operators in **DL-PPA**,
- Conditions: conditional execution $\varphi \rhd \alpha$ in **E-PDDL** executes ε instead of α if φ is not satisfied. Test (Fischer and Ladner, 1979) $\varphi?$ fails if φ is not true in the current state,
- **E-PDDL** does not have the *sequence* connective ;, the *non-exclusive choice* connective $\sqcup$ nor the *Kleene star* $(.)^*$

The interpretation of $\sqcap$ and $\cup$ is the same in both languages. Although trying to stick as much as possible to **DL-PPA** we decided to keep the $\rhd$-connective instead of the test $\varphi?$, because it naturally expresses the notion of effect conditions, i.e. conditions which "launch" effects but are not necessary for applicability. This notion is also natural to deterministic planning, but expressing it with the test necessarily requires a choice connective: $\varphi \rhd \alpha$ is expressed as $(\varphi?;\alpha) \cup (\neg\varphi?;\varepsilon)$.

Although the semantics of the connectives in **DL-PPA** are given via the valuations of programs π (which correspond to action descriptions in this thesis) in the form of triples of valuations of the state variables $(V,U,W) \in \|\pi\|$, meaning "program π can lead from state V to state U via an assignment to the variables in W", we have chosen the view via the set $\mathrm{E}(\alpha,P,s)$ because it is more direct, natural to planning and involves the scope P, which is not the case for **DL-PPA**. These semantics are however equivalent. The link between our definition and the definitions of **DL-PPA** is the following:

$$(V,U,W) \in \|\alpha\| \iff \langle W \cap U, W \setminus U\rangle \in \mathrm{E}(\alpha,P,V)$$

Thus, despite all the differences listed above, **E-PDDL** can indeed be considered to be a sublanguage of **DL-PPA**. Due to the high expressive power of **DL-PPA** it is easy to show that it is more succinct than **E-PDDL** (Scheck et al., 2020).Since it is natural in knowledge compilation to study chains (by inclusion) of languages, we are interested in finding intermediate languages, which are more succinct than **E-PDDL** but less succinct than **DL-PPA**. The sequence connective ; from **DL-PPA** describing via $\alpha;\beta$ the execution of an action β after the action α is the first and probably the most natural construct to add to **E-PDDL**. The resulting language **E-PDDL**$_{\text{seq}}$ is a good candidate for such an intermediate language.

Definition 8.1 (**E-PDDL**$_{\text{seq}}$)**.** An **E-PDDL**$_{\text{seq}}$ *action description* is an expression α generated by the grammar

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

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

Let us emphasize that ";" is not limited to occur only at the root of the expression (in contrast with the actions we aim to compute with SEQUENCE). The interpretation function is the same as for **E-PDDL** action descriptions, augmented with

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

Since an **E-PDDL**$_{\text{seq}}$ subaction can be combined with another action via the $\sqcap$ connective, we also need to specify our semantics in terms of explicit effects (since the semantics of $\sqcap$ depends on effects rather than only on state transitions). We define them in such a way that, if a variable is assigned by both α and β, then it appears in the effects of $\alpha;\beta$ with the same polarity as in the effects of β:

$$\mathrm{E}(\alpha;\beta,P,s) \coloneqq \left\{ \langle Q_\beta^+ \cup (Q_\alpha^+ \setminus Q_\beta^-), Q_\beta^- \cup (Q_\alpha^- \setminus Q_\beta^+)\rangle \;\middle|\; \begin{array}{r} \langle Q_\alpha^+, Q_\alpha^-\rangle \in \mathrm{E}(\alpha,P,s), \\ t \coloneqq (s \cup Q_\alpha^+) \setminus Q_\alpha^-, \\ \langle Q_\beta^+, Q_\beta^-\rangle \in \mathrm{E}(\beta,P,t) \end{array} \right\}$$

For example, for $\alpha \coloneqq +p_1 \sqcap +p_2$ and $\beta \coloneqq -p_1$, the only effect of $\alpha;\beta$ in the state $s \coloneqq \emptyset$ is $\langle\{p_2\},\{p_1\}\rangle$, because the $-p_1$ in β "beats" the $+p_1$ in α.

Example 8.2. *The sequence connective is particularly useful for describing actions featuring an intermediate nondeterministic process whose outcome influences the final result. For example, the* **E-PDDL**$_{\text{seq}}$ *expression*

$$
\begin{aligned}
&+p_{\text{even}}\,;\\
&\big(+p_1 \sqcap (p_{\text{even}} \rhd -p_{\text{even}}) \sqcap (\neg p_{\text{even}} \rhd +p_{\text{even}})\big) \;\cup\; -p_1\,;\\
&\ldots\\
&\big(+p_n \sqcap (p_{\text{even}} \rhd -p_{\text{even}}) \sqcap (\neg p_{\text{even}} \rhd +p_{\text{even}})\big) \;\cup\; -p_n\,;\\
&\neg p_{\text{even}} \rhd \bot
\end{aligned}
$$

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. An action description without the sequence connective would need to enumerate all possible combinations directly, resulting in an exponential tree.

The semantics of the effects defined above for $\alpha\,;\beta$ can also be used to enrich **O-PDDL** with the sequence connective.

Definition 8.3 (**O-PDDL**$_{\text{seq}}$)**.** **O-PDDL**$_{\text{seq}}$ is the language of action descriptions generated by the grammar

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

We will see later that **E-PDDL**$_{\text{seq}}$ and **O-PDDL**$_{\text{seq}}$ are less different than the initial **E-PDDL** and **O-PDDL** in terms of complexity, because ; and $\cup$ together are a very mighty and expressive tool which makes reasoning difficult.

8.2 E-PDDL and O-PDDL with Negation

In the language of NNF action theories it is easy to define the set of nonsuccessors of an action. This can be in particular useful when specifying an action which is explicitly prohibited to have certain consequences. An example is when there are low-level, built-in mechanisms ensuring that apart from its intended effects, the action also has a number of *non*effects: obstacle avoidance for (high-level) navigation actions is a typical example. This motivates us to enrich **O-PDDL** and **E-PDDL** with a negation connective.

We want an interpretation of "not α" to be as in Broersen (2004): an action describing the transition to any state which is not reachable by α, i.e., we want a connective $\neg$ such that $s' \in \neg\alpha(s) \iff s' \notin \alpha(s)$. This interpretation depends only on the transition relation of α and not on its effects in concrete states. Still, because $\neg\alpha$ may occur as an operand of the $\sqcap$ connective, we need to define its explicit effects. However, contrary to the sequence connective, there is no obvious natural way to define the explicit effects of $\neg\alpha$ in s when being given the effects of α in s. The following definitions are all possible candidates satisfying the desired property $s' \in \neg\alpha(s) \iff s' \notin \alpha(s)$, but with different sets of explicit effects.

1. $\mathrm{E}(\neg\alpha, P, s) := \{\langle s' \setminus s, s \setminus s' \rangle \mid s' \notin \alpha(s)\}$; we denote this connective by $\neg_{\text{min}}$ because all effects defined in this manner are minimal in s.

2. $\mathrm{E}(\neg\alpha, P, s) := \{\langle s', P \setminus s' \rangle \mid s' \notin \alpha(s)\}$; we denote this connective by $\neg_{\text{max}}$ because all effects defined in this manner are P-maximal.

3. $\mathrm{E}(\neg\alpha, P, s) := \bigcup_{s' \notin \alpha(s)} \{\langle A, B \rangle \mid s' \setminus s \subseteq A \subseteq s',\ s \setminus s' \subseteq B \subseteq P \setminus s'\}$; we denote this connective by $\neg_{\text{all}}$ because it defines actions which have all possible effects witnessing the desired transitions.

Example 8.4. *The difference in the interaction of negated actions with the $\sqcap$-connective can be illustrated with the following example. Let $P := \{p_1, p_2\}$, $\alpha := +p_1 \sqcap +p_2$, $\beta := +p_2$, and $s := \emptyset$. Then we have $(\neg_{\min}\alpha)(s) = (\neg_{\max}\alpha)(s) = \{\emptyset, \{p_1\}, \{p_2\}\}$, with the effects*

$$\mathrm{E}(\neg_{\min}\alpha, P, s) = \{\langle\emptyset, \emptyset\rangle, \langle\{p_1\}, \emptyset\rangle, \langle\{p_2\}, \emptyset\rangle\}$$
$$\mathrm{E}(\neg_{\max}\alpha, P, s) = \{\langle\emptyset, \{p_1, p_2\}\rangle, \langle\{p_1\}, \{p_2\}\rangle, \langle\{p_2\}, \{p_1\}\rangle\}$$

Therefore

$$\mathrm{E}((\neg_{\min}\alpha) \sqcap \beta, P, s) = \{\langle\{p_2\}, \emptyset\rangle, \langle\{p_1, p_2\}, \emptyset\rangle\}$$
$$\mathrm{E}((\neg_{\max}\alpha) \sqcap \beta, P, s) = \{\langle\{p_2\}, \{p_1\}\rangle\}$$

and thus $((\neg_{\min}\alpha) \sqcap \beta)(s) = \{\{p_2\}, \{p_1, p_2\}\}$, but $((\neg_{\max}\alpha) \sqcap \beta)(s) = \{\{p_2\}\}$. It is also easy to see that $((\neg_{\mathrm{all}}\alpha) \sqcap \beta)(s) = ((\neg_{\min}\alpha) \sqcap \beta)(s)$.

Adding each of the suggested connectives to **E-PDDL** would create a new language with (possibly) different complexity and succinctness properties. In this article we focus on the language of **E-PDDL** enriched with the connective $\neg_{\min}$. The reason to choose $\neg_{\min}$ over $\neg_{\max}$ is that effects of $\neg_{\max}\alpha$ are P-maximal and thus all successors of a state s via $(\neg_{\max}\alpha) \sqcap \beta$ are also necessarily successors via $\neg_{\max}\alpha$, whereas we are interested in the ability of $\sqcap$ to produce new successors. As for $\neg_{\mathrm{all}}$, we suspect that when added to **E-PDDL** or **O-PDDL** it will lead to similar complexity results, but this still requires a separate study.

Definition 8.5 (E-PDDL$_{\mathrm{not}}$, O-PDDL$_{\mathrm{not}}$). An **E-PDDL$_{\mathrm{not}}$** action description is an expression α generated by the grammar

$$\alpha \ ::= \ +p \mid -p \mid \varphi \rhd \alpha \mid (\alpha \cup \alpha) \mid \alpha \sqcap \alpha \mid \neg_{\min}\alpha$$

In the same manner we define **O-PDDL$_{\mathrm{not}}$** by the grammar

$$\alpha \ ::= \ +p \mid -p \mid \varphi \rhd \alpha \mid (\alpha \cup \alpha) \mid \alpha \,\&\, \alpha \mid \neg_{\min}\alpha$$

$\bot$ is not a necessary part of the grammar of **O-PDDL$_{\mathrm{not}}$** because for the scope P it can be expressed by $\neg_{\min}(\&_{p \in P}(+p \cup -p))$, i.e. as the negation af the action with all possible state-transitions.

Example 8.6. *Let $P := \{p_1, p_2, p_3\}$, $s := \{p_1\}$ and $\alpha := +p_2 \cup (-p_1 \sqcap +p_3)$. Then $(\neg_{\min}\alpha)(s) = \{\emptyset, \{p_1\}, \{p_2\}, \{p_1, p_3\}, \{p_2, p_3\}, \{p_1, p_2, p_3\}\}$.*

Example 8.7. *Consider a medical robot who is programmed to give a vaccine to a patient based on the available information. It makes no sense to give a vaccine to someone who receives immunotherapy to treat a cancer at the same time, because with high probability the vaccine will not work. Immunotherapy can possibly cure cancer by increasing the number of lymphocites. The action "immunotherapy" can be described as $(-\mathrm{cancer} \cup \varepsilon) \sqcap (\mathrm{lymphocites\_low} \rhd -\mathrm{lymphocites\_low})$. Then the robot's action "vaccinate" should be inconsistent with whatever outcome is achievable by immunotherapy, thus we want the action "vaccinate" to be expressed as*

$$+\mathrm{vaccinated} \wedge_{\min} \neg_{\min}\Big((-\mathrm{cancer} \cup \varepsilon) \sqcap (\mathrm{lymphocites\_low} \rhd -\mathrm{lymphocites\_low})\Big)$$

*Here the $\wedge_{\min}$ means "keep all common successors", and we will see in the subsection with the results for **E-PDDL$_{\mathrm{not}}$** that the $\wedge_{\min}$ is naturally expressed using $\neg_{\min}$ and $\cup$: $\alpha \wedge_{\min} \beta := \neg_{\min}(\neg_{\min}\alpha \cup \neg_{\min}\beta)$ (this resembles De Morgan's law).*

$\sqcap$ and $\&$ in this example work in the same way because they combine subexpressions which do not share variables.

8.3 E-PDDL and O-PDDL with Conjunction

As we saw in Example 8.7, the intersection of two transition relations can be defined via $\cup$ and $\neg_{\min}$. But since $\vee$ and $\neg$ are enough to express any Boolean function, but $\vee$ and $\wedge$ are not enough, we might conjecture that adding an explicit conjunction connective to **E-PDDL** might produce an intermediate language which is more succinct than just **E-PDDL** but less succinct than **E-PDDL**$_{\text{not}}$ (we will see later that **E-PDDL**$_{\text{not}}$ is indeed more succinct than **E-PDDL**).

Again, due to the semantics of $\sqcap$ we need to define the semantics of $\wedge$ in terms of effects. The effects for the $\wedge$-connective are motivated by Item 2 of Lemma 6.1, namely, that if $s' \in \alpha(s) \cap \beta(s)$ then $s' \in (\alpha \sqcap \beta)(s)$. Therefore we define the effects of $\wedge$ based on those for $\sqcap$. We set

$$\mathrm{E}(\alpha \wedge \beta, P, s) := \left\{ \langle Q_\alpha^+ \cup Q_\beta^+, Q_\alpha^- \cup Q_\beta^- \rangle \;\middle|\; \begin{array}{c} \langle Q_\alpha^+, Q_\alpha^- \rangle \in \mathrm{E}(\alpha, P, s), \langle Q_\beta^+, Q_\beta^- \rangle \in \mathrm{E}(\beta, P, s), \\ (s \cup Q_\alpha^+) \setminus Q_\alpha^- = (s \cup Q_\beta^+) \setminus Q_\beta^- \end{array} \right\}$$

I.e. the effects of $\alpha \wedge \beta$ are combinations of their effects which lead to the same state transitions. Thus $(\alpha \wedge \beta)(s) = \alpha(s) \cap \beta(s)$ is ensured. The requirement for the effects to be mutually consistent (as defined in Section 5.3) is not necessary here because the consistency is ensured automatically by the condition $(s \cup Q_\alpha^+) \setminus Q_\alpha^- = (s \cup Q_\beta^+) \setminus Q_\beta^-$: it holds that $\emptyset = Q_\alpha^- \cap \Big((s \cup Q_\alpha^+) \setminus Q_\alpha^- \Big) = Q_\alpha^- \cap \Big((s \cup Q_\beta^+) \setminus Q_\beta^- \Big)$, and $Q_\beta^+ \subseteq (s \cup Q_\beta^+) \setminus Q_\beta^-$ because $Q_\beta^+ \cap Q_\beta^- = \emptyset$. Thus $Q_\alpha^- \cap Q_\beta^+ = \emptyset$. The other part of consistency, $Q_\beta^- \cap Q_\alpha^+ = \emptyset$, follows analogously.

For the same reason we can at the same time say that the effects for $\wedge$ are motivated by the semantics of &, because if two effects are mutually consistent then their $\sqcap$-combination and &-combination are the same.

Observe that this semantics is different from the one of $\wedge_{\min}$ from the previous section: $\alpha \wedge_{\min} \beta := \neg_{\min}(\neg_{\min}\alpha \cup \neg_{\min}\beta)$ in reality has a $\neg_{\min}$-negation as the outermost connective and therefore all effects of $\alpha \wedge \beta$ are minimal, which is not the case for $\wedge$. For example, $\mathrm{E}(+p \wedge \varepsilon, P, \{p\}) = \{\langle \{p\}, \emptyset \rangle\}$, but $\mathrm{E}(+p \wedge_{\min} \varepsilon, P, \{p\}) = \{\langle \emptyset, \emptyset \rangle\}$.

Definition 8.8. An **E-PDDL**$_{\text{and}}$ action description is generated by the grammar

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

An **O-PDDL**$_{\text{and}}$ action description is an expression α generated by the grammar

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

Note that $\bot$ is not a necessary part of the grammar of **O-PDDL**$_{\text{and}}$ because it can be expressed via $\bot := +p \wedge -p$ because $+p$ and $-p$ never have common successors.

Example 8.9. *The $\wedge$-connective also can be useful to enforce an action execution to satisfy some NNF property. For a given* **E-PDDL** *action α the action $\hat{\alpha} := \alpha \wedge f(\varphi')$ satisfies*

$$s' \in \hat{\alpha}(s) \Leftrightarrow s' \in \alpha(s) \text{ and } s' \models \varphi$$

where φ' is the NNF action theory obtained by replacing each p in φ by its primed copy p' as in Notation 7.3 (page 56) and f is the (polynomial-time) translation from **NNFAT** *into* **E-PDDL** *from the proof of Proposition 6.2 (page 38).*

In general this procedure does not work for **O-PDDL** *because there is no polynomial-size translation from* **NNFAT** *into* **O-PDDL**. *Still the language* **O-PDDL**$_{\text{and}}$ *is worth being mentioned in our work because we will see that many (but not all) results in this chapter are the same for the extensions of* **O-PDDL** *and* **E-PDDL** *because the proofs often do not rely on parallel composition.*

8.4 Complexity: Queries

Lemma 8.10. *The following statements hold:*

1. *In general,* $(\alpha;\alpha) \not\equiv \alpha$
2. *If* $\alpha \equiv \beta$ *then* $\mathrm{E}(\neg_{\min}\alpha, P, s) = \mathrm{E}(\neg_{\min}\beta, P, s)$ *for all* s
3. $\|\neg_{\min}(\neg_{\min}\alpha \cup \neg_{\min}\beta)\| = \|\alpha\| \cap \|\beta\|$
4. *In general,* $\gamma \sqcap (\neg_{\min}(\neg_{\min}\alpha \cup \neg_{\min}\beta)) \not\equiv \gamma \sqcap (\alpha \wedge \beta)$

PROOF.

1. Let $P := \{p\}$, $s := \{p\}$ and $\alpha := (p \rhd -p) \sqcap (\neg p \rhd \bot)$. Then $\emptyset \in \alpha(s)$ but $\alpha;\alpha$ is not even applicable in s.
2. This follows directly from the fact that the effects of $\neg_{\min}\alpha$ are defined only in terms of $\|\alpha\|$.
3. It holds by definition of $\neg_{\min}$: $s' \in (\neg_{\min}\alpha)(s) \Leftrightarrow s' \notin \alpha(s)$, therefore $s' \in (\neg_{\min}\alpha \cup \neg_{\min}\beta)(s) \Leftrightarrow s' \notin \alpha(s)$ or $s' \notin \beta(s)$, thus $s' \in (\neg_{\min}(\neg_{\min}\alpha \cup \neg_{\min}\beta))(s)$ is equivalent to $\neg(s' \notin \alpha(s))$ and $\neg(s' \notin \beta(s))$, i.e. to $s' \in \alpha(s) \cap \beta(s)$.
4. Let $P = \{p\}$, $\gamma = -p$, $\alpha = +p = \beta$. Then $\gamma \sqcap (\alpha \wedge \beta) \equiv (-p) \sqcap (+p) \equiv \bot$ but $\big(\gamma \sqcap (\neg_{\min}(\neg_{\min}\alpha \cup \neg_{\min}\beta))\big)(\{p\}) = \{\emptyset\}$.

□

Example 8.11. *We illustrate Item 2 of the lemma. Let* $P := \{p_1, p_2\}$. *The actions* $\alpha := \varepsilon$ *and* $\beta := (p_1 \rhd +p_1) \sqcap (\neg p_1 \rhd -p_1) \sqcap (p_2 \rhd +p_2) \sqcap (\neg p_2 \rhd -p_2)$ *describe the same action (*$\alpha \equiv \beta$*), but have different effects in all states (*$\mathrm{E}(\alpha, P, s) = \{\langle \emptyset, \emptyset \rangle\}$ *and* $\mathrm{E}(\beta, P, s) = \{\langle s, P \setminus s \rangle\}$*). Still we have* $\mathrm{E}(\neg_{\min}\alpha, P, s) = \mathrm{E}(\neg_{\min}\beta, P, s)$. *In particular this means that* $(\neg_{\min}\alpha) \sqcap \gamma$ *and* $(\neg_{\min}\beta) \sqcap \gamma$ *describe the same action (even effectwise) for all actions* γ.

The third item of Lemma 8.10 states that we can use $\neg_{\min}$ to express the action that maps states s to the successors s' of s which α and β have in common. On the level of state transitions it is simply one of De Morgan's laws: $\varphi \wedge \psi \equiv \neg(\neg\varphi \vee \neg\psi)$. The last item of the Lemma means that despite having De Morgan's law on the level on state transitions we do not have this for effects, and therefore it is not obvious whether **E-PDDL**$_{\text{not}}$ is at least as succinct as **E-PDDL**$_{\text{and}}$ (the same for **O-PDDL**$_{\text{not}}$ and **O-PDDL**$_{\text{and}}$), because with Item 3 can only express conjunctions not in the scope of $\sqcap$ nor of &.

Altogether we will see that the complexity of the extensions of **E-PDDL** and **O-PDDL** which we concentrate on in this section is the same and the proofs are either exactly the same or very similar. Therefore we will write the proofs in details for the extensions of **E-PDDL** and then refer to them when proving results for **O-PDDL**.

The first result is very general.

Proposition 8.12. Is-Succ *is in* PSPACE *for* **E-PDDL**$^{\mathbf{C}}_{\text{seq}}$, **E-PDDL**$^{\mathbf{C}}_{\text{not}}$ *and* **E-PDDL**$^{\mathbf{C}}_{\text{and}}$, *as well as for* **O-PDDL**$^{\mathbf{C}}_{\text{seq}}$, **O-PDDL**$^{\mathbf{C}}_{\text{not}}$ *and* **O-PDDL**$^{\mathbf{C}}_{\text{and}}$.

PROOF. For generality we prove the claim for the circuit representation of the superlanguage of **E-PDDL**$^{\mathbf{C}}_{\text{not}}$, **E-PDDL**$^{\mathbf{C}}_{\text{and}}$, **E-PDDL**$^{\mathbf{C}}_{\text{seq}}$, **O-PDDL**$^{\mathbf{C}}_{\text{seq}}$, **O-PDDL**$^{\mathbf{C}}_{\text{not}}$ and **O-PDDL**$^{\mathbf{C}}_{\text{and}}$ obtained by enriching **E-PDDL** by &, $\neg_{\min}$, $\wedge$ and ;, i.e. defined by the grammar

$$\alpha ::= +p \mid -p \mid \varphi \rhd \alpha \mid (\alpha \cup \alpha) \mid \alpha \,\&\, \alpha \mid \alpha \sqcap \alpha \mid \alpha;\alpha \mid \neg_{\min}\alpha \mid \alpha \wedge \alpha$$

In order to deal with the semantics of $\sqcap$ the actual claim has to be "if $s' \in \alpha(s)$ then for every effect witnessing this transition we can prove that it is indeed a witness using a polynomial amount of memory space, and if $s' \notin \alpha(s)$ then we can prove this in polynomial space, too". We show the claim by induction

on the depth of α. Let q be the upper bound on the space needed to decide successorship. ence we have to prove that q is polynomial.

Obviously an atomic action has the form $\pm p$ and the only effect witnessing this transition can be found in polynomial space. The induction step is then that if we assume that for all action descriptions of depth d and size n any question about whether an effect tuple witnesses a transition can be answered using at most $q(n)$ memory space then any such question can be answered using at most $q(n)$ memory space for all action descriptions of depth $d+1$. Consider an action description α of depth $d+1$ and an effect $\langle Q_\alpha^+, Q_\alpha^- \rangle$. Then there are several options about the root node:

- if $\alpha = \neg_{\min}\beta$ then $\langle Q_\alpha^+, Q_\alpha^- \rangle$ witnesses $s' \in \alpha(s)$ if and only if $s' \notin \beta(s)$ and thus we can check for all possible effects whether they witness $s' \stackrel{?}{\in} \beta(s)$. If there exists at least one $\langle Q_\beta^+, Q_\beta^- \rangle$ of β in s witnessing this transition, then we output "no", otherwise – "yes",
- if $\alpha = \beta \cup \gamma$ then we need to check whether $\langle Q_\alpha^+, Q_\alpha^- \rangle$ is an effect of β witnessing $s' \in \beta(s)$, and if the answer is "no", then do the same for γ.
- if $\alpha = \beta \,\&\, \gamma$ then we need to check for all possible choices of effects $\langle Q_\beta^+, Q_\beta^- \rangle$ and $\langle Q_\gamma^+, Q_\gamma^- \rangle$ of β and γ in s respectively whether their &-combination is $\langle Q_\alpha^+, Q_\alpha^- \rangle$, and again, each of these steps is executed for an action description of depth at most d and thus we require each time $2n$ space units to store the two effects and $q(n-1)$ space units to compute whether e.g. $\langle Q_\beta^+, Q_\beta^- \rangle$ is indeed an effect of β in s,
- if $\alpha = \beta \sqcap \gamma$ then we need to check for all possible choices of compatible effects $\langle Q_\beta^+, Q_\beta^- \rangle$ and $\langle Q_\gamma^+, Q_\gamma^- \rangle$ of β and γ in s respectively whether their $\sqcap$-combination is $\langle Q_\alpha^+, Q_\alpha^- \rangle$, and again, each of these steps is executed for an action description of depth at most d and thus we require each time $2n$ space units to store the two effects and $q(n-1)$ space units to compute whether e.g. $\langle Q_\beta^+, Q_\beta^- \rangle$ is indeed an effect of β in s,
- If $\alpha = \beta \wedge \gamma$ we proceed exactly as for $\sqcap$ and in addition verify that $\langle Q_\beta^+, Q_\beta^- \rangle$ and $\langle Q_\gamma^+, Q_\gamma^- \rangle$ lead to the same state transition in s,
- if $\alpha = \beta\,;\gamma$ then we can repeatedly check for all states s'' and effects $\langle Q_\beta^+, Q_\beta^- \rangle$ and $\langle Q_\gamma^+, Q_\gamma^- \rangle$ whether $\langle Q_\beta^+, Q_\beta^- \rangle$ is a witness for $s'' \in \beta(s)$ and $\langle Q_\gamma^+, Q_\gamma^- \rangle$ is a witness for $s' \in \gamma(s'')$. The amount of space needed for each step is at most $q(n-1)$ by assumption, and we need at most $3n$ space to store s'', $\langle Q_\beta^+, Q_\beta^- \rangle$ and $\langle Q_\gamma^+, Q_\gamma^- \rangle$,
- if $\alpha = \varphi \rhd \beta$ then we need to check whether $s \models \varphi$ first. If this is not the case, then we just check whether $\langle Q_\alpha^+, Q_\alpha^- \rangle$ is $\langle \emptyset, \emptyset \rangle$. If $s \models \varphi$, then we check whether $\langle Q_\alpha^+, Q_\alpha^- \rangle$ is a witness for $s' \in \beta(s)$, which is again possible in $q(n-1)$ space by assumption.

We observe that a polynomial q satisfying the above requirements can be found. □

Corollary 8.13. Is-Applic, Entails, ST, Is-Det *and* Is-Mon *are in* PSPACE *for* **E-PDDL**$^{\mathbf{C}}_{\text{seq}}$, **E-PDDL**$^{\mathbf{C}}_{\text{not}}$, **E-PDDL**$^{\mathbf{C}}_{\text{and}}$, **O-PDDL**$^{\mathbf{C}}_{\text{seq}}$, **O-PDDL**$^{\mathbf{C}}_{\text{not}}$ *and* **O-PDDL**$^{\mathbf{C}}_{\text{and}}$.

PROOF. We reduce all queries to IS-SUCC. We start with IS-APPLIC. We can check that α is applicable in s by guessing a successor s' and verifying successorship. IS-SUCC is in PSPACE for all languages in the claim by Proposition 8.12 and thus IS-APPLIC is in NPSPACE = PSPACE.

A similar trick works for ENTAILS: to show that $\alpha_1\,;\ldots;\alpha_k$ do not entail φ in s we need to guess a chain of successors $s_1,\ldots,s_k$ with $s_1 \in \alpha_1(s)$, $s_2 \in \alpha_2(s_1)$,…,$s_k \in \alpha_k(s_{k-1})$, and verify successorship and $s_k \not\models \varphi$. Every step can be done after another, and each requires polynomial space. Thus, the complement of ENTAILS is in NPSPACE = PSPACE, therefore ENTAILS is in coPSPACE = PSPACE, too.

ST is a special case of IS-SUCC and therefore its membership is clear.

For IS-MON and IS-DET we observe that a witness for the nonmonotony of α in s can be given by one sucessor $s' \in \alpha(s)$ with $s \not\subseteq s'$. Similarly, a witness for nondeterminism of α in s are two successors

$s', s'' \in \alpha(s)$ with $s' \neq s''$. Thus the complements of IS-MON and IS-DET are in NPSPACE = PSPACE, and thus IS-MON and IS-DET are in PSPACE themselves.

The proof is the same for all the languages from the statement of the Corollary because it doesn't make use of the specific language features. □

Thus we have proven the (intuitive) claim that all queries which we study in this article are not harder than PSPACE.

Successorship

Now we consider the tree representations of **E-PDDL**$_{\text{seq}}$, **O-PDDL**$_{\text{seq}}$, **E-PDDL**$_{\text{and}}$ and **O-PDDL**$_{\text{and}}$, which turn out to allow for the easiest queries among the extensions.

Proposition 8.14. Is-Succ *is* NP*-complete for* **E-PDDL**$^{\mathbf{T}}_{\text{seq}}$, **O-PDDL**$^{\mathbf{T}}_{\text{seq}}$ **E-PDDL**$^{\mathbf{T}}_{\text{and}}$ *and* **O-PDDL**$^{\mathbf{T}}_{\text{and}}$.

PROOF. Hardness is clear because IS-SUCC is NP-hard (Proposition 6.9) for **E-PDDL** which is a sublanguage of **E-PDDL**$_{\text{seq}}$ and **E-PDDL**$_{\text{and}}$ (and **O-PDDL** which is a sublanguage of **O-PDDL**$_{\text{seq}}$ and **O-PDDL**$_{\text{and}}$). To show membership, 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 **E-PDDL**$_{\text{seq}}$ (resp. **O-PDDL**$_{\text{seq}}$, **E-PDDL**$_{\text{and}}$ or **O-PDDL**$_{\text{and}}$) description, which is doable in polynomial time by simulating the (only) execution. □

When we turn to circuit representations, the problems become harder

Proposition 8.15. Is-Succ *is* PSPACE*-complete for* **E-PDDL**$^{\mathbf{C}}_{\text{seq}}$ *and* **O-PDDL**$^{\mathbf{C}}_{\text{seq}}$.

PROOF. Membership is proven in Proposition 8.12.

Now for hardness, we give a reduction from the problem of deciding the validity of a QBF of the form $\Phi := \exists p_1 \forall p_2 \exists p_3 \forall p_4 \ldots \exists p_{2n-1} \forall p_{2n} \colon \varphi$, 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 **E-PDDL**$_{\text{seq}}$ (and **O-PDDL**$_{\text{seq}}$ at the same time) expression $\neg\varphi \rhd -p_{\text{sat}}$, and for $i := n-1, \ldots, 0$, define

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

Clearly, α_0 has a circuit of size polynomial in φ (e.g., Fig. 8.1 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 is easy to show the following, by induction on $i := n, n-1, \ldots, 0$:

> for all literals $\ell_1, \ldots, \ell_{2i}$ over $p_1, \ldots, p_{2i}$, 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 $\Phi_i := \exists p_{2i+1} \forall p_{2i+2} \ldots \exists p_{2n-1} \forall p_{2n} \varphi_{|\ell_1,\ldots,\ell_{2i}}$ is valid,

where $\varphi_{|\ell_1,\ldots,\ell_{2i}}$ denotes propositional conditioning, that is, φ with the occurrences of $p_1, \ldots, p_{2i}$ 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 $\Phi_0 = \Phi$. □

For the next proof we introduce some additional notation.

Notation 8.16. Let X_n, $\gamma_1, \ldots, \gamma_{N_n}$ and P_n be as in Notation 6.4, page 39. By $\nu_{=k}$ we denote an assignment to x_k and by $\nu_{<k}$ an assignment to $x_1, \ldots, x_{k-1}$, and $\nu_{\leq k}$ for an assignment to $x_1, \ldots, x_k$. If $\nu_{\leq k-1}$ and $\nu_{=k}$ are clear from the context, then $\nu_{\leq k}$ refers to the induced joint assignment to $x_1, \ldots, x_k$.

For an assignment ν we denote by $t(\nu)$ the encodings (via Notation 6.4) of 3-clauses which are satisfied by ν, i.e. those clauses γ which contain at least one literal which evaluates to $\top$ under ν.

For a fully quantified formula Φ in prenex form with matrix φ let $\Phi_{|\nu_{\leq i}}$ be the partially quantified formula obtained from Φ by forgetting the first i quantifiers and conditioning φ by $\nu_{\leq i}$, i.e. $\Phi_{|\nu_{\leq i}} = \square x_{i+1} \ldots \square x_{2n} \colon \varphi_{|\nu_{\leq i}}$ where $\square$ is a wildcard for $\exists$- and $\forall$-quantifiers, which depends on the formula Φ.

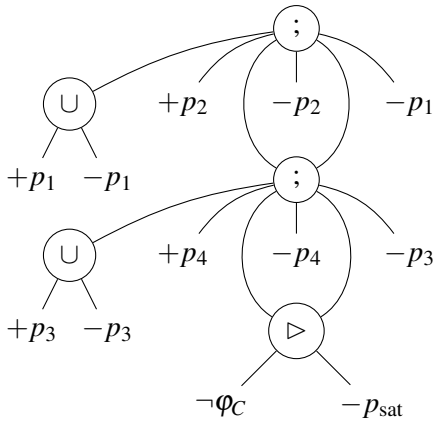


Figure 8.1: The (reduced) circuit representation of the **E-PDDL**$_{\text{seq}}$ action α_0 for $n = 2$ from the proof of Proposition 8.15. The children of all nodes are ordered from the left to the right; φ_C denotes a circuit for φ. The nodes $-p_1$ and $-p_3$ are duplicated only for readability.

Proposition 8.17. Is-Succ *is* PSPACE*-complete for* **E-PDDL**$^{\mathbf{C}}_{\text{and}}$ *and* **O-PDDL**$^{\mathbf{C}}_{\text{and}}$.

PROOF. PSPACE-membership follows from Proposition 8.12. We show PSPACE-hardness by reducing the validity of a QBF to IS-SUCC.

Let $\Phi := \exists x_1 \forall x_2 \ldots \forall x_{2n} : \varphi$ be a quantified Boolean formula with a 3-CNF $\varphi = \gamma_1 \wedge \ldots \wedge \gamma_k$ with clauses γ_j. Recall Notation 6.4 (page 39) with the set P_{2n} to encode all 3-clauses over $X_{2n} = \{x_1, \ldots, x_{2n}\}$. Our state variables will be $P_{2n} \cup X_{2n}$. Set

$$\alpha_{2n+1} := \bigsqcap_{x \in X_{2n}} \left[+x \sqcap \Big(\bigsqcap_{\gamma_i:\ x \in \gamma_i} (+p_i \cup \varepsilon) \Big) \right] \cup \Big(\bigsqcap_{\gamma_i:\ \neg x \in \gamma_i} (+p_i \cup \varepsilon) \Big)$$

This action is analogous to $\alpha_n^{\text{sat},\sqcap}$ from Notation 6.6 (page 40), i.e.. if φ is satisfiable then α_{2n+1} chooses nondeterministically a satisfying assignment (and all satisfying assignments are possible) and produces a successor encoding φ, but in contrast to $\alpha_n^{\text{sat},\sqcap}$ the assignment is part of the successor, too, i.e. $t \cup s(\varphi) \in \alpha_{2n+1}(\emptyset)$ if and only if $t \models \varphi$.

Let $k \in \{1, \ldots, 2n\}$. Set $A_k^+ := \bigsqcap_{j<k}(+x_j \cup \varepsilon) \sqcap \bigsqcap_{j \geq k} +x_j$ and $A_k^- := \bigsqcap_{j<k}(+x_j \cup \varepsilon) \sqcap \bigsqcap_{j>k} +x_j$, so the only difference is the indexation of the second $\bigsqcap$ in the definitions. Let

$$\alpha_{2k} := ((\alpha_{2k+1} \wedge A_{2k}^+) \sqcap +x_{2k}) \wedge ((\alpha_{2k+1} \wedge A_{2k}^-) \sqcap +x_{2k})$$

and

$$\alpha_{2k-1} := ((\alpha_{2k} \wedge A_{2k-1}^+) \sqcap +x_{2k-1}) \cup ((\alpha_{2k} \wedge A_{2k-1}^-) \sqcap +x_{2k-1})$$

Observe that the definitions of α_{2k} and α_{2k-1} differ only about the main (central) connective. There is a double occurence of α_{2k+1} (resp. α_{2k}) in the definitions, but this is not a problem since we use the circuit representation.

Now we prove by induction the following claim (with $s(\varphi)$ as in Notation 6.4 and $v_{\leq i}$, $\Phi_{|v_{\leq i}}$ as in Notation 8.16):

$$\text{for all } \Phi,\ \forall v_{\leq i} \subseteq \{x_1, \ldots, x_i\} : \Phi_{|v_{\leq i}} \text{ is valid } \Leftrightarrow (\{x_j \mid j > i\} \cup v_{\leq i} \cup s(\varphi)) \in \alpha_{i+1}(\emptyset)$$

We will be done when we have proven the claim for $i = 0$, i.e. the reduction in the end looks as follows:

$$\Phi \text{ is valid } \Leftrightarrow (X_{2n} \cup s(\varphi)) \in \alpha_1(\emptyset)$$

Base case of the induction: if $i = 2n$ then $v_{\leq i}$ is a complete assignment (each x_i is assigned) and $\Phi_{|v_{\leq i}}$ evaluates to $\top$ (i.e. $v_{\leq i}$ is a model of φ) if and only if $(v_{\leq i} \cup s(\varphi)) \in \alpha_{2n+1}(\emptyset)$, which follows from the definition of α_{2n+1}.

Induction step: suppose that we know that the induction hypothesis holds for i. We assume first that $i = 2m$ was odd, so we have to show it for $2m-1$ which is even, and thus $\alpha_{2m} = ((\alpha_{2m+1} \wedge A^+_{2m}) \sqcap +x_{2m}) \wedge ((\alpha_{2m+1} \wedge A^-_{2m}) \sqcap +x_{2m})$. We have to show that

$$\text{for all } \Phi,\ \forall \nu_{\leq 2m-1} \subseteq \{x_1,\ldots,x_{2m-1}\} : \Phi_{|\nu_{\leq 2m-1}} \text{ is valid } \Leftrightarrow (\{x_j \mid j > 2m-1\} \cup \nu_{\leq 2m-1} \cup s(\varphi)) \in \alpha_{2m}(\emptyset)$$

.

$\Phi_{|\nu_{\leq 2m-1}}$ being valid is equivalent to $\hat{\Phi} := \Phi_{|\nu_{\leq 2m-1} \cup \{x_{2m}\}}$ and $\tilde{\Phi} := \Phi_{|\nu_{\leq 2m-1}}$ being valid (because the first quantifier of $\Phi_{|\nu_{\leq 2m-1}}$ is universal). By the induction hypothesis we know that $\hat{\Phi}$ is valid if and only if $\{x_j \mid j > 2m\} \cup (\nu_{\leq 2m-1} \cup \{x_{2m}\}) \cup s(\varphi)$ is an α_{2m+1}-successor of $\emptyset$ which is equivalent to $\{x_j \mid j > 2m\} \cup (\nu_{\leq 2m-1} \cup \{x_{2m}\}) \cup s(\varphi)$ being an $\alpha_{2m+1} \wedge A^+_{2m}$-successor of $\emptyset$ because of the definition of A^+_{2m}.

Also from the induction hypothesis we know that $\tilde{\Phi}$ is valid if and only if $\{x_j \mid j > 2m\} \cup \nu_{\leq 2m-1} \cup s(\varphi)$ is an α_{2m+1}-successor of $\emptyset$ which is the case if and only if $\{x_j \mid j > 2m\} \cup \nu_{\leq 2m-1} \cup s(\varphi)$ is an $\alpha_{2m+1} \wedge A^-_{2m}$-successor of $\emptyset$ (again, because of the definition of A^-_{2m}).

Altogether we get that $\Phi_{|\nu_{\leq 2m-1}}$ is true if and only if $\{x_j \mid j > 2m\} \cup (\nu_{\leq 2m-1} \cup \{x_{2m}\}) \cup s(\varphi)$ is an $\alpha_{2m+1} \wedge A^+_{2m}$-successor of $\emptyset$, $\{x_j \mid j > 2m\} \cup \nu_{\leq 2m-1} \cup s(\varphi)$ is an $\alpha_{2m+1} \wedge A^-_{2m}$-successor of $\emptyset$ and thus $\{x_j \mid j > 2m-1\} \cup \nu_{\leq 2m-1} \cup s(\varphi)$ is an α_{2m}-successor of $\emptyset$, because before the central $\wedge$ in the definition of α_{2m} is executed, x_{2m} is set to $\top$ by the $+x_{2m}$-assignment on both sides.

The argumentation is analogous for i being even, but with an "or" instead of "and" for the validity of $\hat{\Phi}$ and $\tilde{\Phi}$.

The proof is exactly the same with & because there are only positive assignments in all action descriptions and hence $\sqcap$ and & behave in the same way. □

Before we can prove the theorem about PSPACE-hardness of IS-SUCC in **E-PDDL**$_{\text{not}}$, we need to prove a technical lemma.

Lemma 8.18. *Let φ, ψ be* 3-CNF *formulas, let $s(\varphi)$ and $s(\psi)$ be as in Notation 6.4 (page 39) with $\mathrm{V}(\varphi), \mathrm{V}(\psi) \subseteq \{x_1,\ldots,x_n\}$, and define the QBF $\Psi^n_\psi := \exists x_1 \colon \neg(\exists x_2 \colon \neg(\ldots \neg(\exists x_n \colon \psi)\ldots))$. Then the following hold:*

1. *If n is odd: if Ψ^n_ψ is true and $s(\varphi) \subseteq s(\psi)$ then Ψ^n_φ is true*
   dually: if Ψ^n_ψ is false and $s(\varphi) \supseteq s(\psi)$ then Ψ^n_φ is false

2. *If n is even: if Ψ^n_ψ is true and $s(\varphi) \supseteq s(\psi)$ then Ψ^n_φ is true*
   dually: if Ψ^n_ψ is false and $s(\varphi) \subseteq s(\psi)$ then Ψ^n_φ is false

PROOF. We prove the claim by induction on n. We first remark that for each item the statement is equivalent to its dual statement, so we need to show only one of them. For $n = 1$ deciding the truth of Ψ^n_ψ is just deciding the satisfiability of ψ. Obviously, if a set of clauses is satisfiable, so is every subset (and dually, if a set of clauses is unsatisfiable, so is every superset). Now suppose that the claim holds for n. We have to distinguish two cases (which are dual to each other).

First assume that n is odd and thus $n+1$ is even. Suppose that Ψ^{n+1}_ψ is true, then there is an assignment ν to x_1 such that $\Phi^\nu_\psi := \exists x_2 \colon \neg(\exists x_3 \colon \ldots \neg(\exists x_{n+1} \colon \psi|_{x_1=\nu(x_1)})\ldots)$ is false. Now let φ be a 3-CNF with $s(\varphi) \supseteq s(\psi)$, then $s(\varphi|_{x_1=\nu(x_1)}) \supseteq s(\psi|_{x_1=\nu(x_1)})$. Since both Φ^ν_ψ and Φ^ν_φ are formulas over n variables we conclude by the inductive assumption (dual case of the first item) that Φ^ν_φ is false. Therefore for Ψ^{n+1}_φ we can use the same assignment ν to show that altogether Ψ^{n+1}_φ is true. Since φ was chosen arbitrarily with the condition of $s(\varphi) \supseteq s(\psi)$, we conclude the claim (we have proven the first statement in the second item of the lemma).

The case when n is even follows with the dual reasoning. □

Proposition 8.19. Is-Succ *is* PSPACE*-complete for both the tree and the circuit representations of* **E-PDDL**$_{\text{not}}$ *and* **O-PDDL**$_{\text{not}}$.

PROOF. For membership we recall from Proposition 8.12 that all queries in this article are not harder than PSPACE.

Now let the sets P_n, X_n and clauses γ_i be as in Notation 6.4. Recall the notation $\wedge_{\min}$ from Example 8.7: we write $\beta \wedge_{\min} \delta$ for the action description $\neg_{\min}(\neg_{\min}\beta \cup \neg_{\min}\delta)$, like in Item 3 of Lemma 8.10. We recall that $(\beta \wedge_{\min} \delta)(s) = \beta(s) \cap \delta(s)$ for all states s.

Let $\Phi := \exists x_1 \colon \neg(\exists x_2 \colon \neg(\ldots(\exists x_n \colon \varphi)))$, with φ a 3-CNF over X_n. We recall from Lemma 3.10 (page 3.10) that deciding the truth of such formulas is a PSPACE-complete problem. We will prove hardness for the tree representation by showing that Φ is true if and only if $s(\varphi) \in \alpha_1^n(\emptyset)$ with

1. $\alpha_n^n := \chi_n^n$
2. $\alpha_k^n := \chi_k^n \sqcap \left(\neg_{\min}\neg_{\min}(\rho_k^n \wedge_{\min} (\neg_{\min}\alpha_{k+1}^n))\right)$ with
   - $\chi_k^n := \left(\bigsqcap_{\gamma_i \colon x_k \in \gamma_i}(+p_i \cup \varepsilon)\right) \cup \left(\bigsqcap_{\gamma_i \colon \neg x_k \in \gamma_i}(+p_i \cup \varepsilon)\right)$
   - $\rho_k^n := \left((\bigsqcap_{\gamma_i \colon x_k \notin \gamma_i}(+p_i \cup \varepsilon)) \cup (\bigsqcap_{\gamma_i \colon \neg x_k \notin \gamma_i}(+p_i \cup \varepsilon))\right)$

We first observe that α_1^n can be constructed in polynomial time. In the following we give an intuition for what these (sub)actions do in $s := \emptyset$. ρ_k^n chooses an assignment to x_k and reassigns all clauses which are not satisfied by this assignment, and the interpretation of this is that a conjunction with ρ_k^n explicitly prohibits to add any of the clauses that contain a literal which is true under this assignment. The subaction χ_k^n chooses an assignment to x_k and then produces nondeterministically any possible set of clauses which are satisfied by this assignment. Recall that for all actions δ we have $\delta \equiv \neg_{\min}\neg_{\min}\delta$; the double negation here serves to make sure that transitions are witnessed by minimal effects.

Recall from Notation 8.16 that $\nu_{=k}$ refers to an assignment to x_k and $\nu_{\leq k}$ to an assignment to $x_1, \ldots, x_k$, and that for an assignment ν we denote by $t(\nu)$ the encodings (via Notation 6.4, page 39) of 3-clauses which are satisfied by ν. Also recall that we denote by $\Phi_{\nu_{\leq k}}$ the fully quantified formula $\exists x_{k+1} \colon \neg(\exists x_{k+2} \colon \neg(\ldots \neg(\exists x_n \colon \varphi_{|\nu_{\leq k}})))$.

Now we prove the claim that for every $0 \leq k \leq n-1$ we have:

$$\text{for all } \Phi, \text{ for all } \nu_{\leq k} \colon [\Phi_{\nu_{\leq k}} \text{ true } \iff s(\varphi) \setminus t(\nu_{\leq k}) \in \alpha_{k+1}^n(\emptyset)]$$

After having shown the claim for $k = 0$ we are done with showing the main claim because $\Phi_{\nu \leq 0} = \Phi$ and $t(\nu_{\leq 0}) = \emptyset$ (since $\nu_{\leq 0}$ is the empty assignement).

We start with $k := n-1$. In this case, $\Phi_{\nu_{\leq k}}$ is just an existentially quantified 3-CNF which is true (being already conditioned by an assignment $\nu_{\leq k}$ to $x_1, \ldots, x_{n-1}$) if and only if the clauses which have not yet been satisfied by $\nu_{\leq k}$ (i.e. $s(\varphi) \setminus t(\nu_{\leq k})$) can be satisfied by an assignment to x_n, which is equivalent to $s(\varphi) \setminus t(\nu_{\leq k})$ being an α_n^n-successor of $\emptyset$ (the statement is analogous to the statement of Lemma 6.7, page 40).

Now we assume that the claim has already been shown for k, and we want to show it for $k-1$: we consider an assignment $\nu_{\leq k-1}$ to $x_1, \ldots, x_{k-1}$, and we prove that $\Phi_{\nu_{\leq k-1}}$ is true if and only if $s(\varphi) \setminus t(\nu_{\leq k-1}) \in \alpha_k^n(\emptyset) = \left(\chi_k^n \sqcap \left(\neg_{\min}\neg_{\min}(\rho_k^n \wedge_{\min} (\neg_{\min}\alpha_{k+1}^n))\right)\right)(\emptyset)$.

"$\Longrightarrow$:" Suppose that $\Phi_{\nu_{\leq k-1}}$ is true. Then there exists an assignment $\nu_{=k}$ to x_k such that $\Phi_{\nu_{\leq k}}$ is false. By the induction hypothesis this means that $T := s(\varphi) \setminus t(\nu_{\leq k}) \notin \alpha_{k+1}^n(\emptyset)$ and thus $T \in \neg_{\min}\alpha_{k+1}^n(\emptyset)$. Since $t(\nu_{=k}) \subseteq t(\nu_{\leq k})$, we have $t(\nu_{=k}) \cap T = \emptyset$, therefore there is an execution of ρ_k^n (the outer choice is defined by the $\nu_{=k}$ and the inner choices are defined by assigning variables in T to $\top$) which produces T. Hence $T \in (\rho_k^n \wedge_{\min} \neg_{\min}(\alpha_{k+1}^n),)(\emptyset)$, and adding the double negation we have $T \in \neg_{\min}\neg_{\min}(\rho_k^n \sqcap \neg_{\min}\alpha_{k+1}^n)(\emptyset)$ with the guarantee that the effect e_T witnessing this transition is minimal – and thus, since the transition starts from $\emptyset$, that e_T has no negative effects.

Now observe that, by construction of χ_k^n, any subset U of $t(\nu_{=k})$ is a successor of $\emptyset$ via χ_k^n and the transition is witnessed by a purely positive effect e_U. This applies to the subset $U := (s(\varphi) \setminus t(\nu_{\leq k-1})) \cap t(\nu_{=k})$. All in all, combining e_T and e_U (which necessarily agree since both contain only positive effects), we get that $U \cup T \in \chi_k^n \sqcap \left(\neg_{\min}\neg_{\min}(\rho_k^n \wedge_{\min} (\neg_{\min}\alpha_{k+1}^n))\right)(\emptyset) = \alpha_k^n(\emptyset)$. Given that $t(\nu_{\leq k})) = t(\nu_{\leq k-1}) \cup t(\nu_{=k})$, it is not hard to see that $U \cup T = s(\varphi) \setminus t(\nu_{\leq k-1})$, which proves the claim.

" $\Longleftarrow$:" Now assume that $s(\varphi) \setminus t(\nu_{\leq k-1}) \in \alpha_k(\emptyset) = \chi_k^n \sqcap \left(\neg_{\min}\neg_{\min}(\rho_k^n \wedge_{\min} (\neg_{\min}\alpha_{k+1}^n))\right)(\emptyset)$.

We first consider the case when $n-k$ is odd. From the assumption we get that there exist U, T such that $s(\varphi) \setminus t(\nu_{\leq k-1}) = U \cup T$ with $U \in \chi_k^n(\emptyset)$ and $T \in \neg_{\min}\neg_{\min}(\rho_k^n \wedge_{\min} (\neg_{\min}\alpha_{k+1}^n))(\emptyset)$. So $T \subseteq s(\varphi) \setminus t(\nu_{\leq k-1})$ so there exists a V such that $(s(\varphi) \setminus t(\nu_{\leq k-1})) \setminus V \in \neg_{\min}\neg_{\min}(\rho_k^n \wedge (\neg_{\min}\alpha_{k+1}^n))(\emptyset)$, and therefore $(s(\varphi) \setminus t(\nu_{\leq k-1})) \setminus V \in (\rho_k^n \wedge (\neg_{\min}\alpha_{k+1}^n))(\emptyset)$. By definition of $\wedge_{\min}$ this transition is possible both via ρ_k^n and $\neg_{\min}\alpha_{k+1}^n$.

Let $\langle Q^+, \emptyset\rangle$ be the effect of ρ_k^n (ρ_k^n has only positive effects by construction) witnessing this transition (i.e. $Q^+ = (s(\varphi) \setminus t(\nu_{\leq k-1})) \setminus V$), which especially means that $Q^+ \in \rho_k^n(\emptyset)$ and $Q^+ \in \neg_{\min}\alpha_{k+1}^n(\emptyset)$. By definition of ρ_k^n there exists an assignment $\nu_{=k}$ to x_k such that $t(\nu_{=k}) \cap Q^+ = \emptyset$. So it holds that $Q^+ \subseteq (s(\varphi) \setminus t(\nu_{\leq k-1})) \setminus t(\nu_{=k}) = s(\varphi) \setminus t(\nu_{\leq k})$. We recall that it also holds that $Q^+ \in \neg_{\min}\alpha_{k+1}^n(\emptyset)$. Thus Q^+ is a set of clauses which encodes a 3-CNF formula ψ with $s(\psi) \subseteq s(\varphi) \setminus t(\nu_{\leq k}) \subseteq s(\varphi)$ (and thus it follows $s(\psi) = s(\psi) \setminus t(\nu_{\leq k})$) and by the inductive assumption $\exists x_{k+1} \colon \neg(\exists x_{k+2} \colon \neg \ldots \neg(\exists x_n \colon \psi_{|\nu_{\leq k}}) \ldots)$ is false. With the first (dual) statement of Lemma 8.18 (recall that $n-k$ and hence the number of quantifiers was odd) it follows that $\exists x_{k+1} \colon \neg(\exists x_{k+2} \colon \neg \ldots \neg(\exists x_n \colon \varphi_{|\nu_{\leq k}}) \ldots)$ is false, too, so $\Phi_{\nu_{\leq k-1}}$ is true.

We now consider the case that $n-k$ is even. For the effect $\langle Q^+, \emptyset\rangle$ (recall that both χ_k^n and $\neg_{\min}\neg_{\min}(\rho_k^n \wedge_{\min} (\neg_{\min}\alpha_{k+1}^n))$ have only positive effects in $\emptyset$) witnessing the transition from $\emptyset$ to $s(\varphi) \setminus t(\nu_{\leq k-1})$ (i.e. $Q^+ = s(\varphi) \setminus t(\nu_{\leq k-1})$) there must be an effect $\langle U^+, \emptyset\rangle$ of χ_k^n with $U^+ \subseteq Q^+$ and by construction of χ_k^n there exists an assignment $\nu_{=k}$ with $U^+ \subseteq t(\nu_{=k})$. Therefore there must exist an effect $\langle T, \emptyset\rangle$ of $\neg_{\min}\neg_{\min}(\rho_k^n \wedge_{\min} (\neg_{\min}\alpha_{k+1}^n))$ in $\emptyset$ with $U^+ \cup T = Q^+$ and thus $s(\varphi) \setminus t(\nu_{\leq k-1}) = Q^+ \supseteq T \supseteq Q^+ \setminus U^+ \supseteq (s(\varphi) \setminus t(\nu_{\leq k-1})) \setminus t(\nu_{=k}) = s(\varphi) \setminus t(\nu_{\leq k})$. Since $T \in \neg_{\min}\neg_{\min}(\rho_k^n \wedge_{\min} (\neg_{\min}\alpha_{k+1}^n))(\emptyset)$ and thus $T \in (\rho_k^n \wedge_{\min} (\neg_{\min}\alpha_{k+1}^n))(\emptyset)$ it follows that $T \in \neg_{\min}\alpha_{k+1}^n(\emptyset)$ and $T \in \rho_k^n(\emptyset)$ by definition of $\wedge_{\min}$. Now we distinguish two cases.

Case 1: assume that $T \cap \nu_{=k} = \emptyset$. Since $T \cap t(\nu_{\leq k-1}) = \emptyset$, we can define a 3-CNF ψ by setting $s(\psi) := T \cup (s(\varphi) \cap t(\nu_{\leq k}))$ such that $s(\psi) \setminus t(\nu_{\leq k}) \supseteq s(\varphi) \setminus t(\nu_{\leq k})$ (we recall that $T \supseteq s(\varphi) \setminus t(\nu_{\leq k})$). By the inductive assumption $\exists x_{k+1} \colon \neg(\exists x_{k+2} \colon \neg \ldots \neg(\exists x_n \colon \psi_{|\nu_{\leq k}}) \ldots)$ is false. Since $n-k$ is even we apply the second statement of Lemma 8.18 and conclude that $\exists x_{k+1} \colon \neg(\exists x_{k+2} \colon \neg \ldots \neg(\exists x_n \colon \varphi_{|\nu_{\leq k}}) \ldots)$ is false, too. Therefore $\Phi_{\nu_{\leq k-1}}$ is true.

Case 2: $T \cap t(\nu_{=k}) \neq \emptyset$. Since T is a ρ_k^n-successor, there must exist an execution corresponding to the (contrary) assignment $\hat{\nu}_{=k}$ (i.e. with $\hat{\nu}_{=k}(x_k) = \neg\nu_{=k}(x_k)$) such that $T \cap \hat{\nu}_{=k} = \emptyset$. Thus it holds that $T \supseteq (s(\varphi) \setminus t(\nu_{\leq k-1})) \setminus t(\hat{\nu}_{=k})$. So the execution of χ_k^n corresponding to the assignment $\hat{\nu}_{=k}$ and producing $s(\varphi) \cap t(\hat{\nu}_{=k}) =: \hat{U}^+$ gives us the desired $\hat{U}^+$ such that $\hat{U}^+ \cup T = Q^+$. Further reasoning is analogous to case 1: we set $\nu_{\leq k} := \nu_{\leq k-1} \cup \hat{\nu}_{=k}$, then define a 3-CNF formula ψ (via $s(\psi) := T \cup (s(\varphi) \cap t(\nu_{\leq k}))$) such that $s(\psi) \setminus t(\nu_{\leq k}) \supseteq s(\varphi) \setminus t(\nu_{\leq k})$. By the inductive assumption $\exists x_{k+1} \colon \neg(\exists x_{k+2} \colon \neg \ldots \neg(\exists x_n \colon \psi_{|\nu_{\leq k}}) \ldots)$ is false. Since $n-k$ is even we apply the second statement of Lemma 8.18 and conclude that $\exists x_{k+1} \colon \neg(\exists x_{k+2} \colon \neg \ldots \neg(\exists x_n \colon \varphi_{|\nu_{\leq k}}) \ldots)$ is false, too. Therefore $\Phi_{\nu_{\leq k-1}}$ is true.

We observe that the connective $\sqcap$ is only involved to combine positive effects (because we consider transitions from $\emptyset$) and thus it can be replaced with & everywhere in the proof without changing the argumentation. □

Applicability and Entailment

We now turn to applicability and entailment.

Lemma 8.20. Is-Succ *is polynomial-time reducible to* Is-Applic *for both representations of* **E-PDDL**$_{\text{seq}}$, **E-PDDL**$_{\text{not}}$, **O-PDDL**$_{\text{seq}}$, **O-PDDL**$_{\text{not}}$, **E-PDDL**$_{\text{and}}$ *and* **O-PDDL**$_{\text{and}}$.

PROOF. Let α be a (tree- or circuit) **E-PDDL**$_{\text{seq}}$ action and let $\psi_{s'}$ be as in Notation 7.3 (page 56), i.e. for all s, $\psi_{s'}(s) = \{s'\}$. For all states s, s' it is easy to see that we have $s' \in \alpha(s)$ if and only if the action $(\alpha\,; (\neg\psi_{s'} \rhd \bot))$ is applicable in s, hence the result.

Now let β be a (tree- or circuit) **E-PDDL**$_{\text{not}}$ action description. Let s, s' be P-states. We define $\rho_{s'} := \left((\bigsqcap_{p \in s'} +p) \sqcap (\bigsqcap_{p \notin s'} -p)\right)$, the deterministic action leading from all states to s'. Then we have $s' \in \beta(s)$ if and only if the action $\gamma := \neg_{\min}(\neg_{\min}\beta \cup \neg_{\min}\rho_{s'})$ is applicable in s, since it follows from Item 3 of Lemma 8.10 that γ satisfies $\gamma(s) = \beta(s) \cap \rho_{s'}(s) \subseteq \{s'\}$.

Let γ be an **E-PDDL**$_{\text{and}}$ action description. Then $s' \in \gamma(s)$ if and only if $\hat{\gamma}$ is applicable in s with $\hat{\gamma} := \gamma \wedge \rho_{s'}$ with $\rho_{s'}$ as in the proof of the claim for **E-PDDL**$_{\text{not}}$.

All proofs work in the same way with & instead of $\sqcap$. □

Corollary 8.21. Is-Applic *is* NP*-complete for* **E-PDDL**$^{\mathbf{T}}_{\text{seq}}$, **O-PDDL**$^{\mathbf{T}}_{\text{seq}}$, **E-PDDL**$^{\mathbf{T}}_{\text{and}}$ *and* **O-PDDL**$^{\mathbf{T}}_{\text{and}}$ *and* PSPACE*-complete for* **E-PDDL**$^{\mathbf{C}}_{\text{seq}}$,**O-PDDL**$^{\mathbf{C}}_{\text{seq}}$, **E-PDDL**$^{\mathbf{C}}_{\text{and}}$, **O-PDDL**$^{\mathbf{C}}_{\text{and}}$ *and for both representations of* **E-PDDL**$_{\text{not}}$ *and* **O-PDDL**$_{\text{not}}$.

PROOF. Hardness follows from Lemma 8.20 together with the previous hardness results about IS-SUCC (Propositions 8.15 to 8.19).

PSPACE-membership follows from Corollary 8.13.

For membership in NP, since IS-SUCC is in NP for **E-PDDL**$^{\mathbf{T}}_{\text{seq}}$, **O-PDDL**$^{\mathbf{T}}_{\text{seq}}$, **E-PDDL**$^{\mathbf{T}}_{\text{and}}$ and **O-PDDL**$^{\mathbf{T}}_{\text{and}}$ (Proposition 8.14), it suffices to define a witness for IS-APPLIC with input α, s, to be a successor s' together with a witness for $s' \in \alpha(s)$. □

Corollary 8.22. Entails *is* coNP*-complete for* **E-PDDL**$^{\mathbf{T}}_{\text{seq}}$, **O-PDDL**$^{\mathbf{T}}_{\text{seq}}$, **E-PDDL**$^{\mathbf{T}}_{\text{and}}$ *and* **O-PDDL**$^{\mathbf{T}}_{\text{and}}$ *and* PSPACE*-complete for* **E-PDDL**$^{\mathbf{C}}_{\text{seq}}$,**O-PDDL**$^{\mathbf{C}}_{\text{seq}}$, **E-PDDL**$^{\mathbf{C}}_{\text{and}}$, **O-PDDL**$^{\mathbf{C}}_{\text{and}}$ *and both representations of* **E-PDDL**$_{\text{not}}$ *and* **O-PDDL**$_{\text{not}}$.

PROOF. We first recall that PSPACE = coPSPACE. Then hardness follows for all the languages from the reduction of non-applicability to entailment together with the hardness results about IS-APPLIC (Corollary 8.21).

For coNP-membership we observe that non-entailment of φ for the sequence $\alpha_1, \ldots, \alpha_k$ and the state s can be verified by guessing sucessors $s_1 \in \alpha_1(s)$, $s_2 \in \alpha_2(s_1)$, $\ldots s_k \in \alpha_k(s_{k-1})$ with $s_k \not\models \varphi$ and the corresponding certificates which can be verified in polynomial time because of the results for IS-SUCC. PSPACE-membership is stated in Corollary 8.13. □

We now turn to IS-DET and IS-MON.

Proposition 8.23. Is-Det *and* Is-Mon *are* coNP*-complete for* **E-PDDL**$^{\mathbf{T}}_{\text{seq}}$, **O-PDDL**$^{\mathbf{T}}_{\text{seq}}$, **E-PDDL**$^{\mathbf{T}}_{\text{and}}$ *and* **O-PDDL**$^{\mathbf{T}}_{\text{and}}$, *and* PSPACE*-complete for* **E-PDDL**$^{\mathbf{C}}_{\text{seq}}$, **O-PDDL**$^{\mathbf{C}}_{\text{seq}}$, **E-PDDL**$^{\mathbf{C}}_{\text{and}}$, **O-PDDL**$^{\mathbf{C}}_{\text{and}}$ *and both representations of* **E-PDDL**$_{\text{not}}$ *and* **O-PDDL**$_{\text{not}}$.

PROOF. PSPACE-membership was shown in Corollary 8.13. coNP-membership follows from NP-membership of the corresponding IS-SUCC (Proposition 8.14), because a certificate for non-monotony for α in s can be given by an s' with $s \not\subseteq s'$ and a certificate for $s' \in \alpha(s)$, and a certificate for non-determinism are two distinct successors together with the corresponding successorship certificates. As for hardness for all languages, it follows from the reduction from non-applicability in the same way as in the proof of Propositions 6.21 and 6.23: α is non-applicable in s if and only if the action $\alpha \sqcap (+q \cup -q)$ (with a fresh variable q) is deterministic in s, and if and only if the action $\alpha \sqcap -q$ is positively monotone in $s \cup \{q\}$. Since q is a fresh variable, $\sqcap$ can be replaced by & to prove the claim for the extensions of **O-PDDL**. Hardness then follows from the hardness result for IS-APPLIC (Corollary 8.21) together with the fact that PSPACE = coPSPACE. □

It remains to determine the complexity for ST.

Proposition 8.24. ST *is linear time solvable for both representations of* **E-PDDL**$_{\text{not}}$ *and* **E-PDDL**$_{\text{and}}$.

PROOF. We recall that in the proof of Proposition 6.26 we gave a polynomial-time algorithm for deciding whether $s \in \alpha(s)$ for **E-PDDL** actions α. This algorithm can be extended to work for **E-PDDL**$_{\text{not}}$ and **E-PDDL**$_{\text{and}}$ because it holds:

- if $\alpha = \beta \wedge \gamma$ then $s \in \alpha(s)$ if and only if $s \in \beta(s)$ and $s \in \gamma(s)$
- if $\alpha = \neg_{\min}\beta$ then $s \in \alpha(s)$ if and only if $s \notin \beta(s)$

□

Proposition 8.25. ST *is* NP*-complete for* $\mathbf{E\text{-}PDDL}^{\mathbf{T}}_{\text{seq}}$ *and* $\mathbf{O\text{-}PDDL}^{\mathbf{T}}_{\text{seq}}$*, and* PSPACE*-complete for* $\mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\text{seq}}$ *and* $\mathbf{O\text{-}PDDL}^{\mathbf{C}}_{\text{seq}}$.

PROOF. We first show the claim for $\mathbf{E\text{-}PDDL}_{\text{seq}}$. Membership is clear because ST is a special case of IS-SUCC and because of the membership results for IS-SUCC (Propositions 8.12 and 8.14). Hardness follows from Corollary 8.21 and the reduction of IS-APPLIC to ST: $\alpha(s) \neq \emptyset$ if and only if $s \in \alpha^s(s)$ with $\alpha^s := \alpha\,;(\bigsqcap_{p\in s} +p)\,;(\bigsqcap_{p\notin s} -p)$. The proof is exactly the same for $\mathbf{O\text{-}PDDL}_{\text{seq}}$ with & instead of $\sqcap$. □

Proposition 8.26. ST *is* NP*-complete for* $\mathbf{O\text{-}PDDL}^{\mathbf{T}}_{\text{and}}$ *and* PSPACE*-complete for* $\mathbf{O\text{-}PDDL}^{\mathbf{C}}_{\text{and}}$.

PROOF. The result for $\mathbf{O\text{-}PDDL}^{\mathbf{T}}_{\text{and}}$ follows from the hardness result for its sublanguage $\mathbf{O\text{-}PDDL}^{\mathbf{T}}$ (Proposition 6.27) and the membership result for $\mathbf{O\text{-}PDDL}^{\mathbf{T}}_{\text{and}}$ (Proposition 8.14). Membership for $\mathbf{O\text{-}PDDL}^{\mathbf{C}}_{\text{and}}$ was proven in Corollary 8.13. To show hardness we refer to the proof of Proposition 8.17: we observe that if the proof was made only for $\mathbf{O\text{-}PDDL}^{\mathbf{C}}_{\text{and}}$ (and not for $\mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\text{and}}$ as well), we could modify the definition of α_{2n+1} to be

$$\alpha_{2n+1} := \underbrace{\left[\left(\mathop{\&}_{x\in X_{2n}}\left(+x\,\&\left(\mathop{\&}_{\gamma_i:\ x\in\gamma_i}(+p_i\cup\varepsilon)\right)\right)\right)\cup\left(-x\,\&\left(\mathop{\&}_{\gamma_i:\ \neg x\in\gamma_i}(+p_i\cup\varepsilon)\right)\right)\right]}_{\beta}\ \&\ \underbrace{\left[\mathop{\&}_{p_i\in P_{2n}} -p_i\right]}_{\delta}$$

and then prove the reduction from deciding the truth of a fully quantified QBF Φ with a 3-CNF matrix (quantifier-free part) φ to ST of the following form:

$$\Phi \text{ is valid } \Leftrightarrow (X_{2n}\cup s(\varphi)) \in \alpha_1(X_{2n}\cup s(\varphi))$$

The modified induction hypothesis (with the same interpretation of $\Phi_{|\nu_{\le i}}$, $\nu_{\le i}$, $\nu_{<i}$ and α_i as in the proof of Proposition 8.17) looks as follows:

$$\text{for all } \Phi, \forall \nu_{\le i} \subseteq \{x_1,\ldots,x_i\} : \Phi_{|\nu_{\le i}} \text{ is valid } \Leftrightarrow (\{x_j \mid j > i\}\cup\nu_{\le i}\cup s(\varphi)) \in \alpha_{i+1}(X_{2n}\cup s(\varphi))$$

We first justify the base case ($i = 2n$): if φ is satisfiable then β produces an effect containing the encoding $s(\varphi)$ together with a satisfying assignment (now the satisfying assignment is obtained by reassigning all variables in X_{2n} since this assignment will be overwritten later by positive assignments to all the x_i anyway). The δ-part of the action description ensures that the encoding $s(\varphi)$ for any unsatisfiable φ will be "destroyed" because in this case it cannot be "saved" by β. The rest of the proof then works with exactly the same argumentation as in the original version. We still give it for clarity.

Induction step: suppose that we know that the induction hypothesis holds for i. We assume first that $i = 2m$ was odd, so we have to show it for $2m-1$ which is even, and thus $\alpha_{2m} = ((\alpha_{2m+1}\wedge A^+_{2m})\sqcap +x_{2m})\wedge((\alpha_{2m+1}\wedge A^-_{2m})\sqcap +x_{2m})$. We have to show for all Φ that

$$\forall \nu_{\le 2m-1} \subseteq \{x_1,\ldots,x_{2m-1}\} : \Phi_{|\nu_{\le 2m-1}} \text{ is valid } \Leftrightarrow (\{x_j \mid j > 2m-1\}\cup\nu_{\le 2m-1}\cup s(\varphi)) \in \alpha_{2m}(X_{2n}\cup s(\varphi))$$

.

$\Phi_{|\nu_{\le 2m-1}}$ being valid is equivalent to $\hat{\Phi} := \Phi_{|\nu_{\le 2m-1}\cup\{x_{2m}\}}$ and $\tilde{\Phi} := \Phi_{|\nu_{\le 2m-1}}$ being valid (because the first quantifier of $\Phi_{|\nu_{\le 2m-1}}$ is universal). By the induction hypothesis we know that $\hat{\Phi}$ is valid if and only if $\{x_j \mid j > 2m\}\cup(\nu_{\le 2m-1}\cup\{x_{2m}\})\cup s(\varphi)$ is an α_{2m+1}-successor of $X_{2n}\cup s(\varphi)$ which is equivalent to $\{x_j \mid j > 2m\}\cup(\nu_{\le 2m-1}\cup\{x_{2m}\})\cup s(\varphi)$ being an $\alpha_{2m+1}\wedge A^+_{2m}$-successor of $X_{2n}\cup s(\varphi)$ because of the definition of A^+_{2m}.

Also from the induction hypothesis we know that $\tilde{\Phi}$ is valid if and only if $\{x_j \mid j > 2m\}\cup\nu_{\le 2m-1}\cup s(\varphi)$ is an α_{2m+1}-successor of $X_{2n}\cup s(\varphi)$ which is the case if and only if $\{x_j \mid j > 2m\}\cup\nu_{\le 2m-1}\cup s(\varphi)$ is an $\alpha_{2m+1}\wedge A^-_{2m}$-successor of $X_{2n}\cup s(\varphi)$ (again, because of the definition of A^-_{2m}).

Altogether we get that $\Phi_{|\nu_{\leq 2m-1}}$ is true if and only if $\{x_j \mid j > 2m\} \cup (\nu_{\leq 2m-1} \cup \{x_{2m}\}) \cup s(\varphi)$ is an $\alpha_{2m+1} \wedge A^+_{2m}$-successor of $X_{2n} \cup s(\varphi)$, $\{x_j \mid j > 2m\} \cup \nu_{\leq 2m-1} \cup s(\varphi)$ is an $\alpha_{2m+1} \wedge A^-_{2m}$-successor of $X_{2n} \cup s(\varphi)$ and thus $\{x_j \mid j > 2m-1\} \cup \nu_{\leq 2m-1} \cup s(\varphi)$ is an α_{2m}-successor of $X_{2n} \cup s(\varphi)$, because before the central $\wedge$ in the definition of α_{2m} is executed, x_{2m} is set to $\top$ by the $+x_{2m}$-assignment on both sides.

The argumentation is analogous for i being even, but with an "or" instead of "and" for the validity of $\hat{\Phi}$ and $\tilde{\Phi}$.

This gives a reduction from the validity of QBF's to ST in $\mathbf{O\text{-}PDDL}^{\mathbf{C}}_{\text{and}}$. □

8.5 Complexity: Succinctness

Due to the high expressive power of the languages in this chapter it is much harder to determine the succinctness relations, therefore the picture is far from being complete.

Proposition 8.27. *If* $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ *then there is no polynomial-size translation from*

- $\mathbf{E\text{-}PDDL}^{\mathbf{T}}_{\text{seq}}$ *or* $\mathbf{O\text{-}PDDL}^{\mathbf{T}}_{\text{seq}}$ *to* $\mathbf{E\text{-}PDDL}^{\mathbf{T}}_{\text{not}}$ *or* $\mathbf{E\text{-}PDDL}^{\mathbf{T}}_{\text{and}}$
- *nor from* $\mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\text{seq}}$ *or* $\mathbf{O\text{-}PDDL}^{\mathbf{C}}_{\text{seq}}$ *to* $\mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\text{not}}$, *or* $\mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\text{and}}$.

PROOF. The algorithm for computing $\alpha_{|t}$ in the proof of Lemma 6.38 (conditioning an action by a partial assignment t to the variables Q) can be enriched for α being an $\mathbf{E\text{-}PDDL}_{\text{not}}$ (respectively $\mathbf{E\text{-}PDDL}_{\text{and}}$) action description. We will show that if $\alpha = \neg_{\min}\beta$, then $f(\alpha) = \neg_{\min} f(\beta)$ indeed satisfies the claim from that proof: it was claimed that for all s, the effects of $f(\alpha)$ in s are exactly the effects $\langle E^+ \setminus Q, E^- \setminus Q\rangle$, where $\langle E^+, E^-\rangle$ is an effect of α in $s \cup t$ witnessing a transition of the form $(s \cup t, s' \cup t)$.

Let s be a P-state. By definition of $\neg_{\min}$, $s' \cup t$ is an α-successor of $s \cup t$ if and only if $s' \cup t$ is not a β-successor of $s \cup t$, if and only if s' is not a $\beta_{|t}$-successor of s, if and only if (by the induction hypothesis) s' is not an $f(\beta)$-successor of s, if and only if (by definition of $\neg_{\min}$ again) s' is an $\neg_{\min} f(\beta)$-sucessor of s. Now by definition of $\neg_{\min}$, the only effect witnessing the transition $(s \cup t, s' \cup t)$ for α is $\langle (s' \cup t) \setminus (s \cup t), (s \cup t) \setminus (s' \cup t)\rangle = \langle s' \setminus s, s \setminus s'\rangle$, and similarly, for $f(\alpha) = \neg_{\min} f(\beta)$, the only effect witnessing the transition from s to s' is $\langle s' \setminus s, s \setminus s'\rangle$, which is indeed $\langle (s' \setminus s) \setminus Q, (s \setminus s') \setminus Q\rangle$ since s, s' do not intersect Q.

As for $\mathbf{E\text{-}PDDL}_{\text{and}}$, the algorithm for computing $\alpha_{|t}$ in the proof of Lemma 6.38 can also be enriched by "if $\alpha = \beta \wedge \gamma$ then $f(\alpha) = f(\beta) \wedge f(\gamma)$". This works for the same reason as for $\sqcap$.

Thus we have shown that $\alpha_{|t}$ can be computed in polynomial time for $\mathbf{E\text{-}PDDL}^{\mathbf{T}}_{\text{not}}$, $\mathbf{E\text{-}PDDL}^{\mathbf{T}}_{\text{and}}$, $\mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\text{not}}$ and $\mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\text{and}}$. We recall the following definitions from the proof of Proposition 6.39 (page 49): For $n \in \mathbb{N}$ and X_n, P_n as in Notation 6.4 and a fresh variable p_{sat} we define the $\mathbf{E\text{-}PDDL}_{\text{seq}}$ (or $\mathbf{O\text{-}PDDL}_{\text{seq}}$, since $\sqcap$ can be replaced by &) actions $\beta_1, \beta_2, \beta_3$ with scope $P = X_n \cup P_n \cup \{p_{\text{sat}}\}$:

$$\beta_1 := \prod_{i=1}^{n} (+x_i \cup -x_i)$$

$$\beta_2 := (\chi_n \rhd +p_{\text{sat}}) \sqcap (\neg\chi_n \rhd -p_{\text{sat}})$$

$$\beta_3 := \prod_{i=1}^{n} -x_i$$

where χ_n is the NNF $\bigwedge_{i=1}^{N_n} (\neg p_i \vee \bigvee_{\ell \in \gamma_i} \ell)$. Recall that for a 3-CNF φ is holds $s(\varphi) \cup \{p_{\text{sat}}\} \in (\beta_1 \,;\, \beta_2 \,;\, \beta_3)(s(\varphi))$ if and only if φ is satisfiable. If any of the languages $L \in \{\mathbf{E\text{-}PDDL}^{\mathbf{T}}_{\text{not}}, \mathbf{E\text{-}PDDL}^{\mathbf{T}}_{\text{and}}, \mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\text{not}}, \mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\text{and}}\}$ could express $\beta_1 \,;\, \beta_2 \,;\, \beta_3$ by an equivalent L-action description δ^n then we could condition δ^n in polynomial time by φ and decide then whether $\{p_{\text{sat}}\} \in \delta^n_{|s(\varphi)}(\emptyset)$ in polynomial time by Lemma 6.36. Thus we would obtain a non-uniform polynomial-time algorithm for deciding 3-CNF. □

Proposition 8.28. *If* $\mathsf{PSPACE} \not\subseteq \mathsf{NP/poly}$ *then there is no polynomial-size translation:*

- *neither from* $\mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\text{not}}$, $\mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\text{and}}$, $\mathbf{O\text{-}PDDL}^{\mathbf{C}}_{\text{not}}$ *or* $\mathbf{O\text{-}PDDL}^{\mathbf{C}}_{\text{and}}$ *into* $\mathbf{E\text{-}PDDL}^{\mathbf{C}}$ *or* $\mathbf{O\text{-}PDDL}^{\mathbf{C}}$,
- *nor from* $\mathbf{E\text{-}PDDL}^{\mathbf{T}}_{\text{not}}$ *or* $\mathbf{O\text{-}PDDL}^{\mathbf{T}}_{\text{not}}$ *into* $\mathbf{E\text{-}PDDL}^{\mathbf{T}}_{\text{seq}}$, $\mathbf{E\text{-}PDDL}^{\mathbf{T}}_{\text{and}}$, $\mathbf{O\text{-}PDDL}^{\mathbf{T}}_{\text{seq}}$ *or* $\mathbf{O\text{-}PDDL}^{\mathbf{T}}_{\text{and}}$.

PROOF. We first illustrate the (already presented) separation proof principle based on exploiting different complexity of IS-SUCC in languages.

If $\mathbf{E\text{-}PDDL}_{\text{not}}$ was translatable into **E-PDDL** with only a polynomial increase in size, we could translate α_1^n from Proposition 8.19 into a polynomial-sized equivalent action $f(\alpha_1^n)$ in $\mathbf{E\text{-}PDDL}^{\mathbf{T}}_{\text{seq}}$ or $\mathbf{E\text{-}PDDL}^{\mathbf{C}}$. Thus we could decide the validity of a quantified Boolean formula of the form $\exists x_1 : \neg(\exists x_2 : \neg(\ldots\neg(\exists x_n : \varphi)))$ by checking $s(\varphi) \in f(\alpha_1^n)(\emptyset)$. Since IS-SUCC is in NP for both representations of **E-PDDL**, this would imply PSPACE $\subseteq$ NP/poly. Thus we have shown, that if PSPACE $\not\subseteq$ NP/poly then $\mathbf{E\text{-}PDDL}^{\mathbf{T}} \not\preceq \mathbf{E\text{-}PDDL}^{\mathbf{T}}_{\text{not}}$ and $\mathbf{E\text{-}PDDL}^{\mathbf{C}} \not\preceq \mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\text{not}}$. With an analogous argumentation (with the $\mathbf{O\text{-}PDDL}_{\text{not}}$ version of α_1^n) we conclude that if PSPACE $\not\subseteq$ NP/poly then $\mathbf{E\text{-}PDDL}^{\mathbf{T}} \not\preceq \mathbf{O\text{-}PDDL}^{\mathbf{T}}_{\text{not}}$ and $\mathbf{E\text{-}PDDL}^{\mathbf{C}} \not\preceq \mathbf{O\text{-}PDDL}^{\mathbf{C}}_{\text{not}}$, and the same results hold with **O-PDDL** instead of **E-PDDL** because IS-SUCC is in NP for both representations of **O-PDDL** and **E-PDDL** (Proposition 6.9).

All other results follow analogously. We recall from the proof of Proposition 8.17 that a fully quantified Boolean formula with 3-CNF matrix φ is true if and only if $(X_{2n} \cup s(\varphi)) \in \alpha_1(\emptyset)$ with the $\mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\text{and}}$ (resp. $\mathbf{O\text{-}PDDL}^{\mathbf{C}}_{\text{and}}$) action α_1. Thus $\mathbf{E\text{-}PDDL}^{\mathbf{T}}_{\text{not}}$, $\mathbf{O\text{-}PDDL}^{\mathbf{T}}_{\text{not}}$, $\mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\text{not}}$, $\mathbf{O\text{-}PDDL}^{\mathbf{C}}_{\text{not}}$, $\mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\text{and}}$ and $\mathbf{O\text{-}PDDL}^{\mathbf{C}}_{\text{and}}$ allow to define families of action descriptions which give a non-uniform algorithm for solving PSPACE-complete decision problems. If there was a polynomial-size translation from one of these languages into one where IS-SUCC is in NP this would imply PSPACE $\subseteq$ NP/poly. We conclude with the observation that IS-SUCC is in NP for $\mathbf{E\text{-}PDDL}^{\mathbf{T}}_{\text{seq}}$, $\mathbf{E\text{-}PDDL}^{\mathbf{T}}_{\text{and}}$, $\mathbf{O\text{-}PDDL}^{\mathbf{T}}_{\text{seq}}$ and $\mathbf{O\text{-}PDDL}^{\mathbf{T}}_{\text{and}}$ by Proposition 8.14 and for $\mathbf{E\text{-}PDDL}^{\mathbf{C}}$ and $\mathbf{O\text{-}PDDL}^{\mathbf{C}}$ by Proposition 6.9 (page 40).

□

8.6 Complexity: Transformations

Since EXTRACT-PRECOND is not polynomial-size in **E-PDDL** (Proposition 6.40), it follows

Corollary 8.29. *If* NP $\not\subseteq$ P/poly *then* Extract-Precond *is not polynomial-size for both representations of* $\mathbf{E\text{-}PDDL}_{\text{and}}$, $\mathbf{E\text{-}PDDL}_{\text{not}}$, $\mathbf{E\text{-}PDDL}_{\text{seq}}$, $\mathbf{O\text{-}PDDL}_{\text{and}}$, $\mathbf{O\text{-}PDDL}_{\text{not}}$ *and* $\mathbf{O\text{-}PDDL}_{\text{seq}}$.

PROOF. If EXTRACT-PRECOND was polynomial-size for any of $\mathbf{E\text{-}PDDL}_{\text{and}}$, $\mathbf{E\text{-}PDDL}_{\text{not}}$, $\mathbf{E\text{-}PDDL}_{\text{seq}}$, it would be polynomial-size for its sublanguage **E-PDDL**, and this is not possible if NP $\not\subseteq$ P/poly by Proposition 6.40 (page 50).

As for **O-PDDL**, recall the following **O-PDDL** actions $P = X_n \cup P_n \cup \{p_{\text{sat}}\}$ from the proof of Proposition 6.39 (page 49): $\hat{\beta}_1 := \&_{i=1}^{n}(+x_i \cup -x_i)$ and $\hat{\beta}_2 := \neg\chi_n \rhd \bot$ with $\chi_n := \bigwedge_{i=1}^{N_n}(\neg p_i \vee \bigvee_{\ell\in\gamma_i}\ell)$. Then $\hat{\beta} := \hat{\beta}_1 ; \hat{\beta}_2$ is applicable in $s(\varphi)$ if and only if the 3-CNF φ is satisfiable. If EXTRACT-PRECOND was polynomial-size for $\mathbf{O\text{-}PDDL}_{\text{seq}}$ then we could extract the polynomial-size NNF precondition of $\hat{\beta}_1 ; \hat{\beta}_2$ and obtain a family of NNF formulas which would constitute a non-uniform polynomial-time algorithm for 3-SAT.

As for $\mathbf{O\text{-}PDDL}_{\text{and}}$: for a literal ℓ we write $+\ell$ meaning $+x$ if $\ell = x$ and $-x$ if $\ell = \neg x$, i.e. we assign the variable of ℓ to the value which satisfies ℓ. Now recall the encoding of 3-CNFs from Notation 6.4 (page 39) and consider the action α_n over $X_n \cup P_n$ with

$$\alpha_n := \bigwedge_{p_i\in P_n}\Big((p_i \rhd \bigcup_{\ell\in\gamma_i} +\ell) \,\&\, \&_{x\in X_n}(-x \cup \varepsilon)\Big)$$

Intuitively, each conjunct is an action which, when applied in $s(\varphi) \cup X_n$, generates any assignment to X_n nondeterministically, and if p_i is true (i.e. the clause γ_i occurs in φ) every generated assignment in addition is a model of the clause γ_i. The conjunction over all $i = 1,\ldots,N_n$ of $(p_i \rhd \bigcup_{\ell\in\gamma_i} +\ell) \,\&\, \&_{x\in X_n}(-x \cup \varepsilon)$ thus produces (when applied in $s(\varphi) \cup X_n$) nondeterministically all models of φ, if φ is satisfiable, and fails otherwise. Now, if EXTRACT-PRECOND was polynomial-size for $\mathbf{O\text{-}PDDL}_{\text{and}}$ than by extracting

the preconditions of α_n we would obtain a family of NNF formulas which would constitute a non-uniform polynomial-time algorithm for 3-SAT.

The proof for **O-PDDL**$_{\text{and}}$ can be reused for **E-PDDL**$_{\text{not}}$: observe that an arbitrary conjunction $\bigwedge_j \beta_j$ of actions β_j can be expressed as $\neg_{\min}(\bigcup_j \neg_{\min}\beta_j)$ (Lemma 8.10), and thus α_n can be expressed in **O-PDDL**$_{\text{not}}$. □

Since the connective $\cup$, is part of the grammar of all languages in this chapter, $\neg_{\min}$ is part of the grammar of **E-PDDL**$_{\text{not}}$ and **O-PDDL**$_{\text{not}}$ and ; is part of the grammar of **E-PDDL**$_{\text{seq}}$ and **O-PDDL**$_{\text{seq}}$, we have the following.

Proposition 8.30. Choice *is linear-time for both representations of* **E-PDDL**$_{\text{and}}$, **E-PDDL**$_{\text{not}}$, **E-PDDL**$_{\text{seq}}$, **O-PDDL**$_{\text{and}}$, **O-PDDL**$_{\text{not}}$, **O-PDDL**$_{\text{seq}}$.

Sequence *is linear-time for both representations of* **E-PDDL**$_{\text{seq}}$ *and* **O-PDDL**$_{\text{seq}}$.

Negation *is linear-time for both representations of* **E-PDDL**$_{\text{not}}$ *and* **O-PDDL**$_{\text{not}}$.

Proposition 8.31. *If* NP $\not\subseteq$ P/poly *then neither the tree nort the circuit representations of* **E-PDDL**$_{\text{and}}$, **E-PDDL**$_{\text{not}}$ *support polynomial-size* Sequence.

PROOF. In the proof of Proposition 8.27 we have actually shown the claim: if NP $\not\subseteq$ P/poly then it is impossible to express $\beta_1 ; \beta_2 ; \beta_3$ in either of the languages **E-PDDL**$^{\mathbf{T}}_{\text{not}}$, **E-PDDL**$^{\mathbf{T}}_{\text{and}}$, **E-PDDL**$^{\mathbf{C}}_{\text{not}}$ and **E-PDDL**$^{\mathbf{C}}_{\text{and}}$, but if SEQUENCE was polynomial-size for any of those languages then expressing two sequences would be possible in polynomial size, too. □

We emphasize that proofs about impossibility of SEQUENCE in this chapter all rely on polynomial-time computability of $\alpha_{|t}$, and since this is not possible for **O-PDDL** and its extensions, we can state results about SEQUENCE only for extensions of **E-PDDL**.

We finally consider NEGATION.

Proposition 8.32. *If* coNP $\not\subseteq$ NP/poly *then* Negation *is not polynomial-size for* **E-PDDL**$^{\mathbf{T}}_{\text{seq}}$, **O-PDDL**$^{\mathbf{T}}_{\text{seq}}$, **E-PDDL**$^{\mathbf{T}}_{\text{and}}$ *and* **O-PDDL**$^{\mathbf{T}}_{\text{and}}$.

PROOF. Recall from Lemma 6.7 (page 40) that

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

is such that $s(\varphi) \in \alpha_n^{\text{sat},\sqcap}(\emptyset)$ holds if and only if φ is satisfiable. Now suppose that NEGATION is polynomial-size in **E-PDDL**$^{\mathbf{T}}_{\text{seq}}$, **O-PDDL**$^{\mathbf{T}}_{\text{seq}}$, **E-PDDL**$^{\mathbf{T}}_{\text{and}}$ or **O-PDDL**$^{\mathbf{T}}_{\text{and}}$. Then there exists a polynomial-sized equivalent $f(\alpha_n^{\text{sat},\sqcap})$ of the negation of $\alpha_n^{\text{sat},\sqcap}$. Thus $s(\varphi) \in f(\alpha_n^{\text{sat},\sqcap})$ holds if and only if φ is unsatisfiable, and so there is a nonuniform NP-algorithm for a coNP-complete problem (IS-SUCC in the above languages is in NP by Proposition 8.14). □

Thus we have several open questions which are related to each other: whether **E-PDDL**$^{\mathbf{C}}_{\text{seq}}$ and **O-PDDL**$^{\mathbf{C}}_{\text{seq}}$ support polynomial-size NEGATION and whether they are at least as succinct as **E-PDDL**$^{\mathbf{C}}_{\text{not}}$ or **O-PDDL**$^{\mathbf{C}}_{\text{not}}$.

8.7 Conclusion

We motivated and introduced three connectives which we used to extend the grammars of **O-PDDL** and **E-PDDL**. We obtained a complete picture of complexity of queries and made the interesting observation that the queries which we study remain in NP for extensions with $\wedge$ and ; in the tree representation, but become PSPACE-complete for the circuit representations. 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.

We observe that the α_0 from the proof of Proposition 8.15 does not involve neither $\sqcap$ nor & and thus IS-SUCC is PSPACE-hard for the circuit representation of the language defined by the grammar

$$\alpha ::= +p \mid -p \mid \varphi \rhd \alpha \mid \alpha \cup \alpha \mid \alpha ; \alpha$$

It would be interesting to study this language in a continuation of this work.

We have only few results for the succinctness part (and many gaps to be filled), as well as for transformations, because the key to these results is identifying queries which have different complexities in different languages, but the circuit representations of our languages have almost only PSPACE-complete queries (except for ST thanks to which we can still draw some conclusions in our work). More sophisticated and artificially-looking queries (possibly queries for effects) are probably needed for proving further results about transformations and filling the gaps in succinctness diagrams.

It is interesting how asymmetry in the semantics of & in **O-PDDL** affects its complexity: all query results except for ST are proven almost identically for extensions of **E-PDDL** and **O-PDDL**, but when it comes to transformations there are still differences. For example, the proofs for impossibility of polynomial-size EXTRACT-PRECOND are quite different for $\mathbf{E\text{-}PDDL}_{\text{and}}$ and $\mathbf{O\text{-}PDDL}_{\text{and}}$. Furthermore, we do not know whether $\mathbf{O\text{-}PDDL}_{\text{not}}$ allows for polynomial-size SEQUENCE, which its egalitarian counterpart definitely does not.

As for concrete practical recommendations which can be derived from our results, a particular direct conclusion is that if a planner works with the tree representations it can be recommended to use the sequence connective when specifying actions, since this can possibly help to avoid exponentially big action descriptions without decreasing the efficiency of the planner.

The gaps in our results naturally define a direction for further research. Another interesting (and probably difficult) question is the succinctness of the "superlanguage" which combines all connectives which we defined in the proof of Proposition 8.12. We conjecture that all queries are PSPACE-complete for this superlanguage, and that it is strictly more succinct than every language from this chapter. It would be particularly interesting to compare this superlanguage to various versions of **DL-PA** and **DL-PPA** (e.g. various definitions of elementary assignments, $\pm p$ or $p \leftarrow \varphi$) from the point of view of succinctness.

Although we generally suspect that all languages defined in this chapter are succinctness-incomparable to each other, it could turn out that e.g. there exists a translation from $\mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\text{not}}$ into $\mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\text{seq}}$. Anyway, every answer to the questions about the existence of such translations would be interesting not only to the planning community, but also to e.g. logicians.

Our results are summarized in Figure 8.2 and in Tables 8.1, 8.2 and 8.3. We did not make a joint diagram for all extensions of **O-PDDL** and **E-PDDL** together because it would become bewildering. Instead we made two diagrams which allow to compare the relations between extensions and their "parent language". We observe that for extensions of **O-PDDL** we have more open questions, this is because the "action conditioning" $\alpha_{|t}$ cannot be computed in polynomial time in **O-PDDL**, contrary to **E-PDDL** (Lemma 6.38).

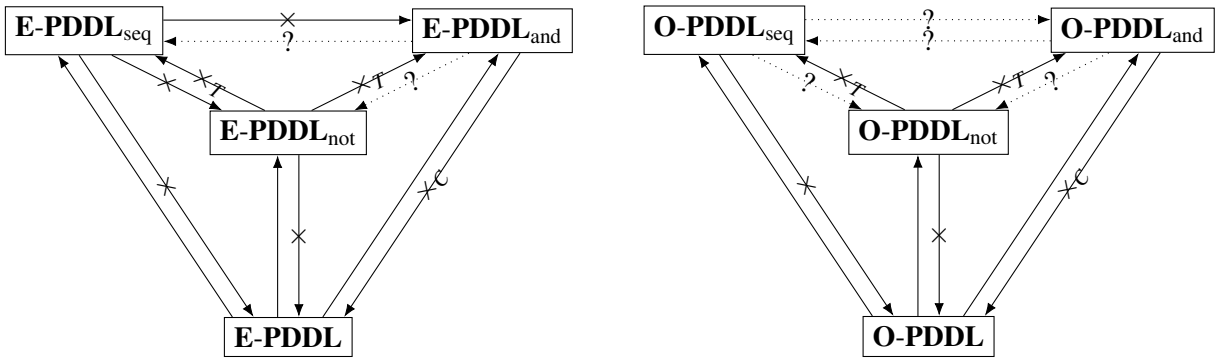


Figure 8.2: Succinctness results for extensions of **E-PDDL** and **O-PDDL**. An arc from L_1 to L_2 means that L_1 can be translated into L_2 in polynomial time (hence also in polynomial size). A crossed out arc from L_1 to L_2 means that there exists not even a polynomial-size translation from L_1 into L_2. These relations hold for both tree and circuit representations. An arc crossed out by $\times_T$ means that the non-existence of a polynomial-size translation is proven only for circuits.

Language	IS-SUCC	IS-APPLIC	ENTAILS
$\mathbf{O\text{-}PDDL}^{\mathbf{T}}_{\mathbf{seq}}$	NP-complete	NP-complete	coNP-complete
$\mathbf{O\text{-}PDDL}^{\mathbf{C}}_{\mathbf{seq}}$	PSPACE-complete	PSPACE-complete	PSPACE-complete
$\mathbf{O\text{-}PDDL}^{\mathbf{T}}_{\mathbf{and}}$	NP-complete	NP-complete	coNP-complete
$\mathbf{O\text{-}PDDL}^{\mathbf{C}}_{\mathbf{and}}$	PSPACE-complete	PSPACE-complete	PSPACE-complete
$\mathbf{O\text{-}PDDL}^{\mathbf{T}}_{\mathbf{not}}$, $\mathbf{O\text{-}PDDL}^{\mathbf{C}}_{\mathbf{not}}$	PSPACE-complete	PSPACE-complete	PSPACE-complete
$\mathbf{E\text{-}PDDL}^{\mathbf{T}}_{\mathbf{seq}}$	NP-complete	NP-complete	coNP-complete
$\mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\mathbf{seq}}$	PSPACE-complete	PSPACE-complete	PSPACE-complete
$\mathbf{E\text{-}PDDL}^{\mathbf{T}}_{\mathbf{and}}$	NP-complete	NP-complete	coNP-complete
$\mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\mathbf{and}}$	PSPACE-complete	PSPACE-complete	PSPACE-complete
$\mathbf{E\text{-}PDDL}^{\mathbf{T}}_{\mathbf{not}}$, $\mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\mathbf{not}}$	PSPACE-complete	PSPACE-complete	PSPACE-complete

Table 8.1: Complexity results for IS-SUCC, IS-APPLIC and ENTAILS.

Language	ST	IS-DET	IS-MON
$\mathbf{O\text{-}PDDL}^{\mathbf{T}}_{\mathbf{seq}}$	NP-complete	coNP-complete	coNP-complete
$\mathbf{O\text{-}PDDL}^{\mathbf{C}}_{\mathbf{seq}}$	PSPACE-complete	PSPACE-complete	PSPACE-complete
$\mathbf{O\text{-}PDDL}^{\mathbf{T}}_{\mathbf{and}}$	NP-complete	coNP-complete	coNP-complete
$\mathbf{O\text{-}PDDL}^{\mathbf{C}}_{\mathbf{and}}$	PSPACE-complete	PSPACE-complete	PSPACE-complete
$\mathbf{O\text{-}PDDL}^{\mathbf{T}}_{\mathbf{not}}$, $\mathbf{O\text{-}PDDL}^{\mathbf{C}}_{\mathbf{not}}$	PSPACE-complete	PSPACE-complete	PSPACE-complete
$\mathbf{E\text{-}PDDL}^{\mathbf{T}}_{\mathbf{seq}}$	NP-complete	coNP-complete	coNP-complete
$\mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\mathbf{seq}}$	PSPACE-complete	PSPACE-complete	PSPACE-complete
$\mathbf{E\text{-}PDDL}^{\mathbf{T}}_{\mathbf{and}}$	linear time	coNP-complete	coNP-complete
$\mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\mathbf{and}}$	linear time	PSPACE-complete	PSPACE-complete
$\mathbf{E\text{-}PDDL}^{\mathbf{T}}_{\mathbf{not}}$, $\mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\mathbf{not}}$	linear time	PSPACE-complete	PSPACE-complete

Table 8.2: Complexity results for ST, IS-DET and IS-MON.

Language	CHOICE	SEQUENCE	NEGATION	EXTRACT-PRECOND
$\mathbf{O\text{-}PDDL}^{\mathbf{T}}_{\mathbf{seq}}$	✓	✓	○	○
$\mathbf{O\text{-}PDDL}^{\mathbf{C}}_{\mathbf{seq}}$	✓	✓	?	○
$\mathbf{O\text{-}PDDL}^{\mathbf{T}}_{\mathbf{and}}$	✓	?	○	○
$\mathbf{O\text{-}PDDL}^{\mathbf{C}}_{\mathbf{and}}$	✓	?	?	○
$\mathbf{O\text{-}PDDL}^{\mathbf{T}}_{\mathbf{not}}$	✓	?	✓	○
$\mathbf{O\text{-}PDDL}^{\mathbf{C}}_{\mathbf{not}}$	✓	?	✓	○
$\mathbf{E\text{-}PDDL}^{\mathbf{T}}_{\mathbf{seq}}$	✓	✓	○	○
$\mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\mathbf{seq}}$	✓	✓	?	○
$\mathbf{E\text{-}PDDL}^{\mathbf{T}}_{\mathbf{and}}$	✓	○	○	○
$\mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\mathbf{and}}$	✓	○	?	○
$\mathbf{E\text{-}PDDL}^{\mathbf{T}}_{\mathbf{not}}$, $\mathbf{E\text{-}PDDL}^{\mathbf{C}}_{\mathbf{not}}$	✓	○	✓	○

Table 8.3: Results for transformations. "✓" means that the transformation can be done in time polynomial in the size of the input. "?" means that the question is open. ○ means that under some complexity-theoretic assumption (see formal statements) the size of the result of the transformation is in general not polynomial in the size of the input.

CHAPTER 9

Extensions of NNF Action Theories

Contents

We want to emphasize ahead that the results in this chapter are given only for the circuit representation. This is because the translation from Subsection 9.1.1 involves a construction which is only polynomial-time constructible when using circuits. We will comment on this at the technical part when giving the results.

A well-known question when representing action and change is how to succinctly specify the *noneffects* of actions, that is, to ensure that the specification precludes fluents to change value while this is not intended. This is known as the *frame problem*. While imperative languages naturally come with a solution to the frame problem, because operational semantics literally transform a situation into another one, the frame problem is crucial in declarative languages, and it has been thoroughly studied, in particular for the Situation Calculus (Reiter, 1991).

In this chapter we propose two different connectives for representing the persistency of fluents, and consider the language of NNF actions theories as enriched by one or the other. The originality of this contribution lies in the facts that (i) the language is *enriched* with an connective (analogously to Chapter 8), which, we argue, allows one to specify the persistency of variables more naturally and succinctly than expressions in the plain underlying logic (like successor-state axioms for the Situation Calculus), and (ii) we allow the connectives to occur anywhere in the formula, including nested occurrences, which again facilitates the description of actions.

Our connectives are one whose interpretation is dependent on the *syntax* of the action in its scope, and one whose interpretation depends only on its semantics (as a relation between the states before and after the action). The "semantic" one corresponds to interpreting the action in its scope under *circumscription* (McCarthy, 1980), here used as a semantics of minimal change through the action. The syntactic operator aims at unifying the semantics of **E-PDDL** and **NNFAT** (because we introduce the notion of effects for **NNFAT**) and simultaneously at understanding the reasons for higher succinctness and complexity of **E-PDDL** compared to **NNFAT**. We give a complete picture for the complexity of queries. Precisely, we show that the syntactic connective can be compiled away in polynomial time for the circuit representation; this shows that actions can be specified in the richer language without harming further calculations. Since the complexity of queries and the results for transformations do not depend on the representation for **NNFAT**, we can directly transfer the complexity results from Chapter 6. On the other

hand, we show that the semantic connective yields a more succinct language, but that the complexity of answering queries increases accordingly. This result again is given for the circuit representation only, but we conjecture that the same hardness results can be proven for the tree representation.

9.1 The Syntactic Frame Connective

We now define a *syntactic* connective, for representing the notion of persistency of languages like **O-PDDL** and **E-PDDL**, where a variable does not change value if there is no *explicit* reason for this (this is the reason why the effects we defined in previous chapters were called explicit effects). Concretely, we want a connective F such that $\{p\}$ is an $\mathsf{F}(p' \vee \neg p')$-successor of $s = \emptyset$, but not an $\mathsf{F}(q' \vee \neg q')$-successor for $q \neq p$, although $p' \vee \neg p'$ describes the same action as $q' \vee \neg q'$. Therefore we need to formalize the intuition that some change is "explicitly mentioned" in an action description. For this, we extend the notion of effects from Definition 5.3 (page 30) which apply in a state in the context of **NNFAT**, and this time effects will themselves be decomposed into *explicit* and *implicit* effects.

Definition 9.1. An *(extended) effect* (over $P \subseteq \mathbb{P}$) is a quadruple $\langle e^+, e^-, i^+, i^- \rangle$ such that e^+, e^-, i^+, i^- are pairwise disjoint subsets of P. The *explicit* (resp. *implicit*) part of such an effect is the pair $\langle e^+, e^- \rangle$ (resp. $\langle i^+, i^- \rangle$). An effect $\langle a^+, a^-, b^+, b^- \rangle$ is a *subeffect* of $\langle e^+, e^-, i^+, i^- \rangle$ if $a^+ \subseteq e^+, a^- \subseteq e^-, b^+ \subseteq i^+$ and $b^- \subseteq i^-$.

We say "extended effect" to distinguish it from the notion of explicit effects we used to define the semantics of **E-PDDL** and **O-PDDL**. But since it will be clear from the context which notion we use, we will simply say "effect" in the rest of the chapter, and the meaning will depend on which language we are talking about.

The positive ($e^+ \cup e^-$) and negative ($e^- \cup i^-$) parts are similar to add- and del-lists in **STRIPS** and to the notion we used before: $\varepsilon = \langle e^+, e^-, i^+, i^- \rangle$ provokes a transition from a state s to $\varepsilon(s) := (s' \cup e^+ \cup i^+) \setminus (e^- \cup i^-)$.

We now define effects for **NNFAT** action descriptions. For combinations of effects via $\wedge$, let $\varepsilon_1 = \langle e_1^+, e_1^-, i_1^+, i_1^- \rangle$ and $\varepsilon_2 = \langle e_2^+, e_2^-, i_2^+, i_2^- \rangle$. If $e_1^+ \cup i_1^+ = e_2^+ \cup i_2^+$ and $e_1^- \cup i_1^- = e_2^- \cup i_2^-$ holds then we call them *compatible* and write $\varepsilon_1 \approx \varepsilon_2$ and set $\varepsilon_1 + \varepsilon_2 := \langle e_1^+ \cup e_2^+, e_1^- \cup e_2^-, i_1^+ \cap i_2^+, i_1^- \cap i_2^- \rangle$. Intuitively, $\varepsilon_1 \approx \varepsilon_2$ means that they provoke exactly the same transitions, but possibly with different explicit/implicit changes, and $\varepsilon_1 + \varepsilon_2$ is the effect which makes explicit any change which is explicit in one of them. We call $\varepsilon_1 + \varepsilon_2$ the *sum* of ε_1 and ε_2.

Definition 9.2. Let α be an **NNFAT** action description and s be a state. Then the set of (extended) *effects* of α with scope P in s, written $\mathrm{E}^{\mathrm{x}}(\alpha, P, s)$, is defined inductively by

- $\mathrm{E}^{\mathrm{x}}(p, P, s) = \{\langle \emptyset, \emptyset, A, B \rangle \mid A, B \subseteq P\}$ for $s \models p$,
  and $\mathrm{E}^{\mathrm{x}}(p, P, s) = \emptyset$ for $s \not\models p$
- $\mathrm{E}^{\mathrm{x}}(\neg p, P, s) = \{\langle \emptyset, \emptyset, A, B \rangle \mid A, B \subseteq P\}$ for $s \not\models p$,
  and $\mathrm{E}^{\mathrm{x}}(\neg p, P, s) = \emptyset$ for $s \models p$
- $\mathrm{E}^{\mathrm{x}}(p', P, s) = \{\langle \emptyset, \emptyset, A, B \rangle \mid A, B \subseteq P \setminus \{p\}\} \cup \{\langle \{p\}, \emptyset, A, B \rangle \mid A, B \subseteq P \setminus \{p\}\}$ for $s \models p$,
  and $\mathrm{E}^{\mathrm{x}}(p', P, s) = \{\langle \{p\}, \emptyset, A, B \rangle \mid A, B \subseteq P \setminus \{p\}\}$ for $s \not\models p$
- $\mathrm{E}^{\mathrm{x}}(\neg p', P, s) = \{\langle \emptyset, \emptyset, A, B \rangle \mid A, B \subseteq P \setminus \{p\}\} \cup \{\langle \emptyset, \{p\}, A, B \rangle \mid A, B \subseteq P \setminus \{p\}\}$ for $s \not\models p$,
  and $\mathrm{E}^{\mathrm{x}}(\neg p', P, s) = \{\langle \emptyset, \{p\}, A, B \rangle \mid A, B \subseteq P \setminus \{p\}\}$ for $s \models p$
- $\mathrm{E}^{\mathrm{x}}(\alpha_1 \wedge \alpha_2, P, s) = \{\varepsilon_1 + \varepsilon_2 \mid \varepsilon_1 \in \mathrm{E}^{\mathrm{x}}(\alpha_1, s), \varepsilon_2 \in \mathrm{E}^{\mathrm{x}}(\alpha_2, s), \varepsilon_1 \approx \varepsilon_2\}$;
- $\mathrm{E}^{\mathrm{x}}(\alpha_1 \vee \alpha_2, P, s) = \mathrm{E}^{\mathrm{x}}(\alpha_1, s) \cup \mathrm{E}^{\mathrm{x}}(\alpha_2, s) \cup \mathrm{E}^{\mathrm{x}}(\alpha_1 \wedge \alpha_2, s)$.

where A, B are always disjoint from each other.

Intuitively, the first case says that the action p is not applicable if s does not satisfy p (second line), and otherwise imposes no constraint on the successor state, but moreover does not set any value *explicitly*: a transition may occur from, say, $\{p,q\}$ to $\{r\}$, but then the positive effects $A = \{r\}$ and negative effects $B = \{p,q\}$ are considered to be *implicit*.

For atomic actions of the form p', we distinguish two cases. When s does not already satisfy p, then all effects of p' *explicitly* set p. Constrastingly, when s already satisfies p, then the action p' leaves the value unchanged, and we include both effects with an explicit setting of p (to the same value) and effects with no setting of p at all (neither implicit nor explicit). Of course, for fixed A,B, those two effects provoke a transition from s to the same successor s'. Nevertheless, including the effects which do not set p at all turns out to be necessary for $\wedge$ to behave as expected in the extension of **NNFAT** with the F connective (to be defined soon).

Finally, it is worth noting that we include the effects of $\alpha_1 \wedge \alpha_2$ in those of $\alpha_1 \vee \alpha_2$. Of course, the *transitions* of $\alpha_1 \wedge \alpha_2$ are already included in those of α_1 and in those of α_2, but not necessarily with the same explicit parts.

Example 9.3. *Let $P = \{p,q\}$ $\alpha_1 \coloneqq p'$, $\alpha_2 \coloneqq q'$, $\alpha \coloneqq \alpha_1 \vee \alpha_2$ and $s = \{p\}$. Then α_1 and α_2 have the following effects:*

$$\mathrm{E}^{\mathrm{x}}(\alpha_1,P,s) = \{\langle\{p\},\emptyset,\emptyset,\emptyset\rangle, \langle\{p\},\emptyset,\{q\},\emptyset\rangle, \langle\{p\},\emptyset,\emptyset,\{q\}\rangle, \langle\emptyset,\emptyset,\emptyset,\emptyset\rangle, \langle\emptyset,\emptyset,\{q\},\emptyset\rangle, \langle\emptyset,\emptyset,\emptyset,\{q\}\rangle\}$$
$$\mathrm{E}^{\mathrm{x}}(\alpha_2,P,s) = \{\langle\{q\},\emptyset,\emptyset,\emptyset\rangle, \langle\{q\},\emptyset,\{p\},\emptyset\rangle, \langle\{q\},\emptyset,\emptyset,\{p\}\rangle\}$$

Then the effects of α are the union of those sets of effects together with the additional effect $\langle\{p,q\},\emptyset,\emptyset,\emptyset\rangle$ which is the sum of $\langle\{p\},\emptyset,\{q\},\emptyset\rangle$ and $\langle\{q\},\emptyset,\{p\},\emptyset\rangle$.

With this in hand, we can define the *framing* connective F_X. Intuitively, $\mathsf{F}_X(\alpha)$ retains only those effects of α which include no implicit effect on variables of X. As can be seen, this is equivalent to removing variables of X from the implicit part of all effects. So we define $\mathrm{E}^{\mathrm{x}}(\mathsf{F}_X(\alpha),P,s)$ to be

$$\{\langle e^+, e^-, i^+ \setminus X, i^- \setminus X\rangle \mid \langle e^+, e^-, i^+, i^-\rangle \in \mathrm{E}^{\mathrm{x}}(\alpha,P,s)\}$$

Analogously to Definition 5.4 (page 30) we say that an effect is *minimal* in s if $s \cap (e^+ \cup i^+) = \emptyset$ and $(e^- \cup i^-) \subseteq s$, i.e. all variables occuring in the effect really change the values of those variables in s.

Definition 9.4. The action language $\mathbf{NNFAT}_{\mathsf{F}}$ is the language whose expressions with scope P are defined by the grammar

$$\alpha ::= p \mid p' \mid \neg p \mid \neg p' \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \mathsf{F}_X(\alpha),$$

where p ranges over P and X over subsets of P, and the interpretation function I is defined for all P, $s \subseteq P$ and α with $\mathrm{V}(\alpha) \subseteq P \cup (P)'$ via

$$I(\alpha,P)(s) \coloneqq \{(s \cup e^+ \cup i^+) \setminus (e^- \cup i^-) \mid \langle e^+, e^-, i^+, i^-\rangle \in \mathrm{E}^{\mathrm{x}}(\alpha,P,s)\}$$

It can be seen by induction that F_X can only remove successors (i.e. $\mathsf{F}_X(\alpha)(s) \subseteq \alpha(s)$).

Example 9.5. *Consider the action of leaving one's bike in a garage for them to repair exactly one wheel, but not knowing which one in advance.*[1] *The action may have two effects: making the front wheel or the back wheel ok (in both cases not affecting the other). However, in the process of repairing the back wheel, it might occur that the gear is changed. Additionally, in no case would the brakes be affected. Such an action could be encoded by*

$$\mathsf{F}_{\mathsf{brakes}}\big(\mathsf{F}_{\mathsf{f\_wheel\_ok}}(\mathsf{b\_wheel\_ok}') \vee \mathsf{F}_{\mathsf{b\_wheel\_ok,gear}}(\mathsf{f\_wheel\_ok}')\big)$$

Importantly, in general, pushing all occurrences of F to the root of the expression changes its interpretation. For instance, in Example 9.5, this would yield the expression

$$\mathsf{F}_{\mathsf{brakes,f\_wheel\_ok,b\_wheel\_ok,gear}}\big(\mathsf{b\_wheel\_ok}' \vee \mathsf{f\_wheel\_ok}'\big)$$

according to which the gear can never change value.

[1] *E.g.*, knowing only that they are not aligned which each other.

9.1.1 Compiling the Syntactic Connective Away

While $\mathbf{NNFAT}_{\mathsf{F}}$ seems to be a "real" extension of **NNFAT** (i.e. a strictly more succinct language), we will show that the syntactic connective F_X can be compiled away, that is, eliminated from an $\mathbf{NNFAT}^{\mathbf{C}}_{\mathsf{F}}$ expression to yield an expression in $\mathbf{NNFAT}^{\mathbf{C}}$ which describes the same action and has the same interpretation and size (up to a polynomial). Moreover, the elimination can be done in polynomial time. This means that $\mathbf{NNFAT}_{\mathsf{F}}$ can be used as a convenient language for describing actions, still the algorithms which manipulate those descriptions (e.g. planners) need not be extended to cope with F, since all its occurrences can simply be eliminated in polynomial time and space.

Before we proceed, we need some technical lemmas and additional notation.

Notation 9.6. We set $\mathrm{E}^{\mathrm{x}}(\alpha,P,s,s') := \{\langle e^+,e^-,i^+,i^-\rangle \in \mathrm{E}^{\mathrm{x}}(\alpha,P,s) \mid (s\cup e^+\cup i^+)\setminus(e^-\cup i^-) = s'\}$, i.e. the set of effects of α in s which describe the transition (s,s'). If P is clear from the context we will write $\mathrm{E}^{\mathrm{x}}(\alpha,s,s')$ instead of $\mathrm{E}^{\mathrm{x}}(\alpha,P,s,s')$, and $\mathrm{E}^{\mathrm{x}}(\alpha,s)$ instead of $\mathrm{E}^{\mathrm{x}}(\alpha,P,s)$ to avoid unnecessary notation.

Thus, if we say that $\langle e^+,e^-,i^+,i^-\rangle \in \mathrm{E}^{\mathrm{x}}(\alpha,s,s')$ this in particular means that $s' \in \alpha(s)$ and $\langle e^+,e^-,i^+,i^-\rangle$ is a witness for this.

In the rest of the section we will stick to Notation 9.6 and not mention the scope P in the effect definitions, assuming for each statement that P is arbitrary, but fixed.

Lemma 9.7. *If $\varepsilon = \langle U,V,W,Z\rangle \in \mathrm{E}^{\mathrm{x}}(\alpha,s)$ and $\tilde{\varepsilon} = \langle A,B,C,D\rangle$ with $A\subseteq U, B\subseteq V, C\subseteq W, D\subseteq Z$ and $\varepsilon(s) = s' = \tilde{\varepsilon}(s)$ then $\tilde{\varepsilon} \in \mathrm{E}^{\mathrm{x}}(\alpha,s)$.*

PROOF. The statement holds for atomic actions $p,\neg p,p',\neg p'$ by definition.

For $\alpha = F_X(\beta)$: let $\varepsilon_x := \langle U,V,W,Z\rangle \in \mathrm{E}^{\mathrm{x}}(\beta,s)$ and $\varepsilon = \langle U,V,W\setminus X, Z\setminus X\rangle \in \mathrm{E}^{\mathrm{x}}(\alpha,s)$. We set $X_w := W\cap X$ and $X_z := Z\cap X$. Let $A\subseteq U, B\subseteq V, C\subseteq (W\setminus X), D\subseteq (Z\setminus X)$ with $\tilde{\varepsilon} := \langle A,B,C,D\rangle$ be such that $\tilde{\varepsilon}(s) = \varepsilon(s)$. We define $\tilde{\varepsilon}_x := \langle A,B,C\cup X_w, D\cup X_z\rangle$ which is by assumption in $\mathrm{E}^{\mathrm{x}}(\beta,s)$ because it has the same effect on s as ε_x. $\tilde{\varepsilon}$ is obtained from $\tilde{\varepsilon}_x$ by application of F_X so $\tilde{\varepsilon} \in \mathrm{E}^{\mathrm{x}}(\alpha,s)$ and we are done.

For $\alpha = \alpha_1\wedge\alpha_2$: let $\varepsilon = \langle U,V,W,Z\rangle \in \mathrm{E}^{\mathrm{x}}(\alpha,s)$ with $\varepsilon = \varepsilon_1+\varepsilon_2$, with $\varepsilon_i = \langle U_i,V_i,W_i,Z_i\rangle$. We want to show that if $\tilde{\varepsilon}(s) = \varepsilon(s)$ for $\tilde{\varepsilon} = \langle \tilde{U},\tilde{V},\tilde{W},\tilde{Z}\rangle$ with $\tilde{U}\subseteq U, \tilde{V}\subseteq V, \tilde{W}\subseteq W, \tilde{Z}\subseteq Z$, then $\tilde{\varepsilon} \in \mathrm{E}^{\mathrm{x}}(\alpha,s)$. We want to find effects $\tilde{\varepsilon}_1$ and $\tilde{\varepsilon}_2$ with the same effect on s as ε_1 and ε_2 but which are subeffects and whose sum gives $\tilde{\varepsilon}$. We set $\tilde{U}_1 := U_1\cap\tilde{U}$,$\tilde{W}_1 := W_1\cap(\tilde{U}\cup\tilde{W})$, $\tilde{U}_2 := U_2\cap\tilde{U}$,$\tilde{W}_2 := W_2\cap(\tilde{U}\cup\tilde{W})$, $\tilde{V}_1 := V_1\cap\tilde{V}$, $\tilde{Z}_1 := Z_1\cap(\tilde{V}\cup\tilde{Z})$, $\tilde{V}_2 := V_2\cap\tilde{V}$, $\tilde{Z}_2 := Z_2\cap(\tilde{V}\cup\tilde{Z})$. This definition is correct (i.e. $\tilde{U}_1\subseteq U_1$, $\tilde{U}_2\subseteq U_2$, $\tilde{W}_1\subseteq W_1$, $\tilde{W}_2\subseteq W_2$, $\tilde{V}_1\subseteq V_1$, $\tilde{V}_2\subseteq V_2$, $\tilde{Z}_1\subseteq Z_1$, $\tilde{Z}_2\subseteq Z_2$, $\tilde{\varepsilon}_1(s) = \varepsilon_1(s), \tilde{\varepsilon}_2(s) = \varepsilon_2(s)$ and $\tilde{\varepsilon}_1+\tilde{\varepsilon}_2 = \tilde{\varepsilon}$).

For $\alpha = \alpha_1\vee\alpha_2$ the claim follows from the induction hypothesis together with the definition of effects for $\vee$ and the result for $\wedge$. □

Especially it follows that there is always a minimal effect satisfying a given state-transition, i.e. there is at least one $\langle e^+,e^-,i^+,i^-\rangle \in \mathrm{E}^{\mathrm{x}}(\alpha,s,s')$ such that $e^+\cup i^+ = s'\setminus s$ and $e^-\cup i^- = s\setminus s'$.

Lemma 9.8. *Let α,β be two action expressions in $\mathbf{NNFAT}_{\mathsf{F}}$, and let s,s' be two states such that s' is both an α- and a β-successor of s. Let p be a variable with a different value in s and s', and such that there is an effect $\varepsilon_p = \langle e_p^+,e_p^-,i_p^+,i_p^-\rangle \in \mathrm{E}^{\mathrm{x}}(\alpha,s,s')\cup\mathrm{E}^{\mathrm{x}}(\beta,s,s')$ with $p\in e_p^+\cup e_p^-$ (that is, p is assigned explicitly). Then there is an effect $\varepsilon = \langle e^+,e^-,i^+,i^-\rangle \in \mathrm{E}^{\mathrm{x}}(\alpha\wedge\beta,s,s')$ such that $p\in e^+\cup e^-$ holds.*

PROOF. By symmetry, assume $\langle e_p^+,e_p^-,i_p^+,i_p^-\rangle \in \mathrm{E}^{\mathrm{x}}(\alpha,s,s')$ (so that $p\in e_p^+\cup e_p^-$ holds) and let $\varepsilon_\beta = \langle e_\beta^+,e_\beta^-,i_\beta^+,i_\beta^-\rangle \in \mathrm{E}^{\mathrm{x}}(\beta,s,s')$ be an effect of β (with not necessarily $p\in e_\beta^+\cup e_\beta^-$).

Write X for the set of all variables which have different values in s and s', write $\varepsilon_{p,X}$ for the effect $\langle e_p^+\cap X, e_p^-\cap X, i_p^+\cap X, i_p^-\cap X\rangle$, and similarly for $\varepsilon_{\beta,X}$. Since all variables which change value from s to s' are retained, we have $\varepsilon_{p,X}\in\mathrm{E}^{\mathrm{x}}(\alpha,s,s')$ and $\varepsilon_{\beta,X}\in\mathrm{E}^{\mathrm{x}}(\beta,s,s')$. Moreover, for all $q\in X$, since q has different values in s and s' and ε_p transform s into s', $q\in e_p^+\cup e_p^-\cup i_p^+\cup i_p^-$ must hold and hence, $q\in e_{p,X}^+\cup e_{p,X}^-\cup i_{p,X}^+\cup i_{p,X}^-$ holds. The same reasoning shows $q\in e_{\beta,X}^+\cup e_{\beta,X}^-\cup i_{\beta,X}^+\cup i_{\beta,X}^-$ and finally, $\varepsilon_{p,X}\approx\varepsilon_{\beta,X}$ are compatible together. Moreover, by construction $p\in e_{p,X}^+\cup e_{p,X}^-$ holds. It follows that the effect $\varepsilon := \varepsilon_{p,X}+\varepsilon_{\beta,X}$ is as desired. □

Lemma 9.9. *Let α be an* **NNFAT**$_{\mathsf{F}}$ *action description, let s be a state, and let $\varepsilon_1, \varepsilon_2 \in \mathrm{E}^{\mathrm{x}}(\alpha, s)$ with $\varepsilon_1 \approx \varepsilon_2$. Then the effect $\varepsilon_1 + \varepsilon_2$ is in $\mathrm{E}^{\mathrm{x}}(\alpha, s)$.*

PROOF. Since the effects are compatible with each other, their combination using $+$ is well-defined. We now reason by induction on the structure of α.

For atomic α, it is easy to see that ε_1 and ε_2 have to be equal or incompatible with each other. Hence the result clearly holds.

Now for $\alpha = \beta \wedge \gamma$, there exist $\varepsilon_i^\beta, \varepsilon_i^\gamma$, $i = 1,2$ with $\varepsilon_i^\beta \in \mathrm{E}^{\mathrm{x}}(\beta, s), \varepsilon_i^\gamma \in \mathrm{E}^{\mathrm{x}}(\gamma, s)$ and $\varepsilon_i = \varepsilon_i^\beta + \varepsilon_i^\gamma$. By definition of $\approx$ it follows that $\varepsilon_1^\beta \approx \varepsilon_2^\beta$ and $\varepsilon_1^\gamma \approx \varepsilon_2^\gamma$ and $\varepsilon_1^\beta + \varepsilon_2^\beta \approx \varepsilon_1^\gamma + \varepsilon_2^\gamma$, and by the inductive assumption it follows $\varepsilon_1^\beta + \varepsilon_2^\beta \in \mathrm{E}^{\mathrm{x}}(\beta, s)$ and $\varepsilon_1^\gamma + \varepsilon_2^\gamma \in \mathrm{E}^{\mathrm{x}}(\gamma, s)$, and by definition of the sum we conclude $\varepsilon = (\varepsilon_1^+ \approx \varepsilon_2^\beta) + (\varepsilon_1^\gamma + \varepsilon_2^\gamma)$

Similarly, for $\alpha = \beta \vee \gamma$, either both are effects of the same subexpression, in which case the result holds by the induction hypothesis, or they are effects of β and γ, respectively, in which case the result follows since the effects of $\beta \wedge \gamma$ are included in those of $\beta \vee \gamma$. Finally, for $\alpha = \mathsf{F}_X(\beta)$, we have that both effects are effects of β and their implicit part does not intersect X, so the same holds for their combination through $+$ and hence, the latter is also an effect of α. □

We are now in position to define the transformation for compiling the F_X connective away. This will be done by defining a polynomial-time transformation similar to introducing successor-state axioms, but taking into account the fact that F_X may occur at any level of nesting.

Definition 9.10. Let α be a **NNFAT**$^{\mathrm{C}}_{\mathsf{F}}$ action description with scope P. We recursively define the action $\mathrm{Expl}(\alpha, p)$ by

- α for $\alpha = p'$ or $\alpha = \neg p'$,
- $\bot$ for $\alpha = q'$ or $\alpha = \neg q'$ with $q \neq p$,
- $\bot$ for $\alpha = q$ or $\alpha = \neg q$, for all variables q,
- $(\mathrm{Expl}(\beta, p) \wedge \gamma) \vee (\beta \wedge \mathrm{Expl}(\gamma, p))$ for $\alpha = \beta \wedge \gamma$,
- $\mathrm{Expl}(\beta, p) \vee \mathrm{Expl}(\gamma, p)$ for $\alpha = \beta \vee \gamma$,
- $\bigwedge_{x \in X \cup \{p\}} \big((x \leftrightarrow x') \vee \mathrm{Expl}(\beta, x)\big)$ for $\alpha = \mathsf{F}_X(\beta)$.

We observe that the size of the reduced circuit of Expl is polynomial in the cardinality of P and the size of α.

The following is the core idea for the desired translation. It shows that if p changes its value via α explicitly, then the corresponding state transition must be possible via $\mathrm{Expl}(\alpha, p)$. For convenience we will write $(s, s') \models \alpha$ in the rest of the section, meaning $s' \in \alpha(s)$, because our translation will be defined in such a manner that $\mathrm{Expl}(\alpha, s)$ will be a pure **NNFAT** action description and thus a Boolean formula.

Lemma 9.11. *Let α be an expression in* **NNFAT**$_{\mathsf{F}}$*, s, s' be two states, and p be a variable with a different value in s and s'. Then there is an effect $\varepsilon = \langle e^+, e^-, i^+, i^- \rangle \in \mathrm{E}^{\mathrm{x}}(\alpha, s, s')$ with $p \in e^+ \cup e^-$ if and only if $(s, s') \models \alpha \wedge \mathrm{Expl}(\alpha, p)$.*

PROOF. Assume first that there is an effect $\varepsilon = \langle e^+, e^-, i^+, i^- \rangle \in \mathrm{E}^{\mathrm{x}}(\alpha, s, s')$ with $p \in e^+ \cup e^-$. By the definition of $\mathrm{E}^{\mathrm{x}}(\alpha, s, s')$ we have $(s, s') \models \alpha$. We show $(s, s') \models \mathrm{Expl}(\alpha, p)$ by induction on the structure of α.

For $\alpha = p'$ or $\alpha = \neg p'$, since there is an effect in $\mathrm{E}^{\mathrm{x}}(\alpha, s, s')$, we have in particular $(s, s') \models \alpha = \mathrm{Expl}(\alpha, p)$. For $\alpha = q'$ or $\alpha = \neg q'$ with $q \neq p$, and for $\alpha = q$ or $\alpha = \neg q$ with q being any variable, the result holds vacuously since there is no effect $\langle e^+, e^-, i^+, i^- \rangle$ of α at all with $p \in e^+ \cup e^-$.

Now for $\alpha = \beta \wedge \gamma$ with **NNFAT**$_{\mathsf{F}}$ actions β, γ, by definition of $\wedge$ there are effects $\varepsilon_\beta = \langle e_\beta^+, e_\beta^-, i_\beta^+, i_\beta^- \rangle \in \mathrm{E}^{\mathrm{x}}(\beta, s, s')$ and $\varepsilon_\gamma = \langle e_\gamma^+, e_\gamma^-, i_\gamma^+, i_\gamma^- \rangle \in \mathrm{E}^{\mathrm{x}}(\gamma, s, s')$ with $\varepsilon = \varepsilon_\beta + \varepsilon_\gamma$, and from $p \in e^+ \cup e^-$ and the definition of $+$ it follows $p \in e_\beta^+ \cup e_\beta^-$ or $p \in e_\gamma^+ \cup e_\gamma^-$. Assume by symmetry $p \in e_\beta^+ \cup e_\beta^-$. Then by the induction

hypothesis we have $(s,s') \models \mathrm{Expl}(\beta,p)$. Moreover, from $\varepsilon \in \mathrm{E}^{\mathrm{x}}(\alpha,s,s')$ it follows $(s,s') \models \alpha$ and hence $(s,s') \models \beta \wedge \gamma$ and $(s,s') \models \gamma$. In the end, we get $(s,s') \models \mathrm{Expl}(\beta,p) \wedge \gamma$, as desired.

For $\alpha = \beta \vee \gamma$, the result follows directly from the definition of $\vee$ and the induction hypothesis.

Finally, for $\alpha = \mathsf{F}_X(\beta)$, the result follows from the fact that the set of effects of α is a subset of the set of effects of β, together with the fact that by definition of F_X, variables from X which change value can only do so via an explicit effect of β and together with the induction hypothesis.

Conversely, assume $(s,s') \models \alpha$ and $(s,s') \models \mathrm{Expl}(\alpha,p)$.

For $\alpha = p'$, from $(s,s') \models \mathrm{Expl}(\alpha,p) = \alpha$ we get that s' is indeed an α-successor of s and hence, the effect $\langle\{p\},\emptyset,\emptyset,\emptyset\rangle$ is as desired. The reasoning is similar for $\alpha = \neg p'$. For $\alpha = q'$ or $\alpha = \neg q'$ with $q \neq p$, and for $\alpha = q$ or $\alpha = \neg q$ with q being any variable, the result holds vacuously since $\mathrm{Expl}(\alpha,p)$ is $\bot$ and hence, $(s,s') \models \mathrm{Expl}(\alpha,p)$ does not hold.

Now for $\alpha = \beta \wedge \gamma$ with $\mathbf{NNFAT}_{\mathsf{F}}$ actions β,γ, assume by symmetry $(s,s') \models \mathrm{Expl}(\beta,p) \wedge \gamma$. Then we have $(s,s') \models \mathrm{Expl}(\beta,p)$; on the other hand, by assumption $(s,s') \models \alpha$ and hence $(s,s') \models \beta$, and from the induction hypothesis it follows that there is an effect $\varepsilon_\beta = \langle e^+_\beta, e^-_\beta, i^+_\beta, i^-_\beta\rangle \in \mathrm{E}^{\mathrm{x}}(\beta,s,s')$ with $p \in e^+_\beta \cup e^-_\beta$. Moreover, from $(s,s') \models \gamma$ it follows that there is an effect $\varepsilon_\gamma \in \mathrm{E}^{\mathrm{x}}(\gamma,s,s')$. Then from Lemma 9.8 it follows that there is an effect $\langle e^+, e^-, i^+, i^-\rangle \in \mathrm{E}^{\mathrm{x}}(\alpha,s,s')$ with $p \in e^+ \cup e^-$, as desired.

For $\alpha = \beta \vee \gamma$, the result follows directly from the definition of $\vee$.

Finally, for $\alpha = \mathsf{F}_X(\beta)$, since $p \not\leftrightarrow p'$, from $(s,s') \models \mathrm{Expl}(\alpha,p)$ it follows that $(s,s') \models \mathrm{Expl}(\beta,p)$; moreover, since $\mathsf{F}_X(\beta)$-successors are always β-successors, it follows $(s,s') \models \beta$ and by the induction hypothesis and Lemma 9.7 we get that there is a minimal effect $\varepsilon_1 = \langle e^+_\beta, e^-_\beta, i^+_\beta, i^-_\beta\rangle \in \mathrm{E}^{\mathrm{x}}(\beta,s,s')$ with $p \in e^+_\beta \cup e^-_\beta$; on the other hand, by assumption $(s,s') \models \alpha$, so there exists a minimal effect $\varepsilon_2 = \langle a^+_\beta, a^-_\beta, b^+_\beta, b^-_\beta\rangle \in \mathrm{E}^{\mathrm{x}}(\beta,s,s')$ with $\varepsilon_2 \in \mathrm{E}^{\mathrm{x}}(\alpha,s,s')$, that is, the implicit part of ε_2 intersects X only in those variables which don't change their value from s to s'. By Lemma 9.9, $\varepsilon := \varepsilon_1 + \varepsilon_2 \in \mathrm{E}^{\mathrm{x}}(\beta,s,s')$ and since its implicit part is a subeffect of the implicit part of ε_2, $\varepsilon \in \mathrm{E}^{\mathrm{x}}(\alpha,s,s')$ and it has p as an explicit effect. □

We are now ready to prove the main result of the section. Our procedure essentially amounts to replacing all occurrences of F with subformulas similar to successor-state axioms.

Proposition 9.12. $\mathbf{NNFAT}^{\mathbf{C}}_{\mathsf{F}}$ *is translatable into* $\mathbf{NNFAT}^{\mathbf{C}}$ *in polynomial time.*

PROOF. Let α be a $\mathbf{NNFAT}^{\mathbf{C}}_{\mathsf{F}}$ action description with scope P. The translation $f(\alpha)$ is obtained by replacing each node $\mathsf{F}_X(\beta)$ (with $X \subseteq P$) of the circuit of α by $\beta \wedge \bigwedge_{p\in X}\big((p \leftrightarrow p') \vee \mathrm{Expl}(\beta,p)\big)$ in a bottom-up fashion and keeping the other nodes. The bottom-up manner guarantees that for each $\mathsf{F}_X(\beta)$ which we replace the argument node β has already been transformed into an **NNFAT**. Observe that $(p \leftrightarrow p') \equiv (p \wedge p') \vee (\neg p \wedge \neg p')$ and thus $f(\alpha)$ is indeed an NNF formula (we use $\leftrightarrow$ for better clarity).

We first show $f(\alpha) \equiv \alpha$, by induction on the structure of α. For all constructs but F_X the result is straightforward, since those constructs are semantic and the root of α is left unchanged; for instance, for $\alpha = \beta \wedge \gamma$ we have $f(\alpha) = f(\beta) \wedge f(\gamma)$, and by the induction hypothesis we have $f(\beta) \equiv \beta$ and $f(\gamma) \equiv \gamma$; since $\wedge$ is semantic, it follows $f(\alpha) \equiv \beta \wedge \gamma = \alpha$.

So let $\alpha = \mathsf{F}_X(\beta)$ be an expression in $\mathbf{NNFAT}_{\mathsf{F}}$, and let s,s' be two states.

Assume first $(s,s') \models f(\alpha)$. Then in particular we have $s' \in \beta(s)$. It remains to show that there is an effect $\varepsilon_\beta = \langle e^+_\beta, e^-_\beta, i^+_\beta, i^-_\beta\rangle \in \mathrm{E}^{\mathrm{x}}(\beta,s,s')$ whose explicit part $e^+_\beta \cup e^-_\beta$ contains all variables p from X which change value from s to s'; indeed, from this we will conclude that these effects are unaffected by F_X and hence, that this effect with X removed from its implicit part also transforms s to s' on behalf of α. Let p be a variable as above. From the definition of $f(\alpha)$ we conclude $(s,s') \models \beta \wedge \mathrm{Expl}(\beta,p)$, and by Lemma 9.11 it follows that there is an effect $\varepsilon_{\beta,p} = \langle e^+_{\beta,p}, e^-_{\beta,p}, i^+_{\beta,p}, i^-_{\beta,p}\rangle \in \mathrm{E}^{\mathrm{x}}(\beta,s,s')$ with $p \in e^+_{\beta,p} \cup e^-_{\beta,p}$. As in the proof of Lemma 9.8, it can be seen that by first restricting those $\varepsilon_{\beta,p}$'s (for each p) to the set of all variables which change value from s to s', we get effects which are all in $\mathrm{E}^{\mathrm{x}}(\beta,s,s')$ and which are compatible together. Hence their sum ε_β via $+$ is well-defined, and it transforms s into s'. Moreover, by construction ε_β has in its explicit part all those p's which change value from s to s'. Finally, Lemma 9.9 shows that ε_β is an effect of β, which concludes.

Conversely, assume $s' \in \alpha(s)$. Then by definition of F_X we first get $s' \in \beta(s)$. Now let p be a variable in X. If p has the same value in s and s', then $(s,s') \models (p \leftrightarrow p')$ holds. Otherwise, from $s' \in \beta(s)$ and the fact that p is in X but changes value from s to s', we get that there is an effect in $\mathrm{E}^{\mathrm{x}}(\beta,s,s')$ with p in its explicit part. Then from Lemma 9.11 it follows $(s,s') \models \mathrm{Expl}(\beta,p)$. Hence in the end, we get $(s,s') \models \beta \wedge \bigwedge_{p\in X}((p \wedge p') \vee (\neg p \wedge \neg p') \vee \mathrm{Expl}(\beta,p)) = f(\alpha)$, as desired.

Finally, $f(\alpha)$ can be computed in polynomial time from α since in the end, we have only introduced one new node per variable p and per node β of α (those nodes $\mathrm{Expl}(\beta,p)$), and none of these nodes creates any new node (they only use other $\mathrm{Expl}(\gamma,q)$ nodes), plus a linear number of nodes of the form $\beta \wedge \bigwedge_{p\in X}((p \wedge p') \vee (\neg p \wedge \neg p') \vee \mathrm{Expl}(\beta,p))$, which also only reuse other γ or $\mathrm{Expl}(\gamma,q)$ nodes. □

Since **NNFAT**$^{\mathbf{C}}$ and **NNFAT**$^{\mathbf{C}}_{\mathsf{F}}$ are polynomial-time translatable into each other, we conclude that all polynomial-time tractability results of **NNFAT**$^{\mathbf{C}}$ hold for **NNFAT**$_{\mathsf{F}}$, as well as all (polynomial-time possibility and polynomial-size impossibility) results about transformations and succinctness (see Chapter 6). The only results which need to be adjusted are those for IS-SUCC, ST and NEGATION (we recall that those queries were linear-time for **NNFAT**).

Corollary 9.13. Choice *can be performed in linear time for* **NNFAT**$^{\mathbf{C}}_{\mathsf{F}}$. Is-Succ *and* ST *can be answered and* Negation *can be performed in polynomial time for* **NNFAT**$^{\mathbf{C}}_{\mathsf{F}}$.

PROOF. For two **NNFAT**$^{\mathbf{C}}_{\mathsf{F}}$ action α and β CHOICE is achieved via the **NNFAT**$^{\mathbf{C}}_{\mathsf{F}}$ action $\alpha \vee \beta$. The claim for IS-SUCC, ST and NEGATION follows from the linear time results in Chapter 6 (Proposition 6.3, 6.26 and 6.35, pages 37-47) together with the existence of a polynomial-time translation and the sublanguage relation. □

Corollary 9.14. Is-Applic *is* NP*-complete and* Is-Det, Is-Mon *and* Entails *are* coNP*-complete for* **NNFAT**$^{\mathbf{C}}_{\mathsf{F}}$.

PROOF. The claim follows from results for **NNFAT** in Chapter 6 (Propositions 6.12 and Corollaries 6.17, 6.24 and 6.25, pages 42-44). □

Corollary 9.15. *If* NP $\not\subseteq$ P/poly *then* Sequence *and* Extract-Precond *are not polynomial-size for* **NNFAT**$^{\mathbf{C}}_{\mathsf{F}}$.

PROOF. Follows from the results for **NNFAT** (Propositions 6.39 and 6.40, pages 49 -50). □

Corollary 9.16. **NNFAT**$^{\mathbf{C}}_{\mathsf{F}}$ *is polynomial-time translatable into* **E-PDDL**$^{\mathbf{C}}$. *If* NP $\not\subseteq$ P/poly *then* **O-PDDL**$^{\mathbf{C}}$ *and* **NNFAT**$^{\mathbf{C}}_{\mathsf{F}}$ *are succinctness-incomparable to each other.*

PROOF. **NNFAT**$^{\mathbf{C}}_{\mathsf{F}}$ is polynomial-time translatable into **NNFAT**$^{\mathbf{C}}$ which is polynomial-time translatable into **E-PDDL**$^{\mathbf{C}}$ by Proposition 6.2 (page 38).

Assume NP $\not\subseteq$ P/poly. **NNFAT**$^{\mathbf{C}}_{\mathsf{F}}$ is not polynomial-size translatable into **O-PDDL**$^{\mathbf{C}}$ because **NNFAT**$^{\mathbf{C}}$ is not (Proposition 6.29, page 46). The converse result follows from the nonexistence of a polynomial-size translation from **O-PDDL**$^{\mathbf{C}}$ into **E-PDDL**$^{\mathbf{C}}$ (Proposition 6.31, page 46), but in the proof we must replace "then check in linear time" by "then check in polynomial time", which does not change the argumentation. □

9.2 The Semantic Frame Connective

The semantic frame connective which we study builds on *circumscription* (McCarthy, 1980), which is a nonmonotonic semantics for formulas enforcing a form of closed-world assumption, and especially on *propositional* circumscription (Eiter and Gottlob, 1993; Nordh, 2005). However, by introducing a *connective* for this interpretation, we allow circumscription to be enforced only on some parts of an expression.

Let α be an action description and s be a P-state. For all partitions $\{X,V,F\}$ of P, we introduce the connective $\mathsf{C}_{X,V,F}$ so that $\mathsf{C}_{X,V,F}(\alpha)(s)$ chooses those α-successors of s which change variables from X minimally among all states with the same values over F.

Precisely, we define a state s' to be *preferred* to a state s'' with respect to a state s and to X, V, F, which we write $s' \prec^s_{X,V,F} s''$, if $s' \cap F = s'' \cap F$ and $(s'\Delta s) \cap X \subsetneq (s''\Delta s) \cap X$ hold, where Δ denotes symmetric difference for sets.[2]

Definition 9.17. The action language $\mathbf{NNFAT}_{\mathsf{C}}$ is the language whose expressions with scope P are defined by the grammar

$$\alpha ::= p \mid p' \mid \neg p \mid \neg p' \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \mathsf{C}_{X,V,F}(\alpha),$$

where p ranges over P and $\langle X, V, F\rangle$ over partitions of P, and the interpretation function I is defined as for the original **NNFAT** (in Chapter 5), extended with

$$I(\mathsf{C}_{X,V,F}(\alpha), P)(s) = \{s' \in I(\alpha, P)(s) \mid \nexists s'' \in I(\alpha, P)(s) : s'' \prec^s_{X,V,F} s'\}$$

We observe that for every partition X, V, F of P, every state s and every $\mathbf{NNFAT}_{\mathsf{C}}$ action α the action $\mathsf{C}_{X,V,F}(\alpha)$ is applicable in s if and only if α is applicable in s, because a $\mathsf{C}_{X,V,F}(\alpha)$-successor is chosen among α-successors.

In other words, analogously to F_X from the previous section, $\mathsf{C}_{X,V,F}$ can only remove successors, i.e. $\mathsf{C}_{X,V,F}(\alpha)(s) \subseteq \alpha(s)$.

Another important observation is that F_X is *not* a special case of $\mathsf{C}_{X,V,F}$. It might seem that F_X is nothing more than $\mathsf{C}_{X,\emptyset,P\setminus X}$. However, taking $P = \{p\}$, $\alpha = p' \vee \neg p'$, and $s = \emptyset$, it can be seen that both $\emptyset$ and $\{p\}$ are $\mathsf{F}_X(\alpha)$-successors of s, while only $\emptyset$ is a $\mathsf{C}_{X,\emptyset,P\setminus X}(\alpha)$-successor. Indeed, $\mathsf{F}_X(\alpha)$ takes into account the fact that both alternatives are explicitly mentioned in the formula, while $\mathsf{C}_{X,\emptyset,P\setminus X}(\alpha)$ considers only the semantics of α and hence, is equivalent to $\mathsf{C}_{X,\emptyset,P\setminus X}(\top)$.

Example 9.18. *Let $P = \{p_1, \ldots, p_5\}$, $X = \{p_1, p_2\}$, $V = \{p_3\}$, and $F = \{p_4, p_5\}$. Let $\alpha = (p'_1 \vee p'_3) \wedge (p'_2 \vee p'_4) \wedge (p'_5)$ and $s = \emptyset$. Then $\{p_1, p_2, p_5\}$ is an α-successor, but not a $\mathsf{C}_{X,V,F}(\alpha)$-successor, of s, because $s'' = \{p_2, p_3, p_5\}$ is also an α-successor of s with $s'' \cap F = \{p_5\} = s' \cap F$ and $(s''\Delta s) \cap X = \{p_2\} \subset \{p_1, p_2\} = (s'\Delta s) \cap X$. On the other hand, s'' is a $\mathsf{C}_{X,V,F}(\alpha)$-successor of s even though $s''' = \{p_3, p_4, p_5\}$ changes fewer values over X, since s'' and s''' differ over F and hence are incomparable with each other.*

It can be seen that the $\mathsf{C}_{X,V,F}$ connective is convenient in particular for expressing actions which involve external causes (to be put in the set F) of changes of values for variables of interest (set X), in the presence of ramifications (set V).

Example 9.19. *Consider encoding the action of driving from work to home. A particularly succinct description is*

$$\mathsf{C}_{\{\mathsf{home}\},\{\mathsf{at\_work}\},\{\mathsf{flat\_tire},\mathsf{engine\_ok}\}} \Big((\mathsf{engine\_ok}' \wedge \neg \mathsf{flat\_tire}') \rightarrow \mathsf{home}') \wedge (\mathsf{home}' \leftrightarrow \neg \mathsf{at\_work}') \Big)$$

Consider $s = \{\mathsf{engine\_ok}, \mathsf{at\_work}\}$. Minimization of change over $\{\mathsf{home}\}$ entails that when the causes are not met (for instance, when $\mathsf{engine\_ok}'$ is false), among the possible successors of s only the ones with $\neg\mathsf{home}'$ are retained, ignoring the ramification $\mathsf{at\_work}'$; successors with home' are not retained, reflecting the fact that there is no "proof" provided by the action that home should change value. On the other hand, due to their presence in the set F, all combinations of causes will be retained. Precisely, the successors of s are $\{\mathsf{engine\_ok}, \mathsf{at\_work}, \mathsf{flat\_tire}\}$, $\{\mathsf{at\_work}\}$, $\{\mathsf{at\_work}, \mathsf{flat\_tire}\}$, and $\{\mathsf{engine\_ok}, \mathsf{home}\}$.

9.2.1 Complexity and Succinctness

We now study complexity and succinctness issues for $\mathbf{NNFAT}^{\mathbf{C}}_{\mathsf{C}}$.

Proposition 9.20. Is-Succ, Is-Applic, Entails, ST, Is-Mon *and* Is-Det *are in* PSPACE *for* $\mathbf{NNFAT}^{\mathbf{C}}_{\mathsf{C}}$.

[2] As mnemonics, variables in V may vary; those in F are fixed.

PROOF. We start with IS-SUCC. $s' \in \alpha(s)$ can be checked as follows:

- if α is atomic (i.e. $\alpha \in \{p, p', \neg p, \neg p'\}$) then $s' \in \alpha(s)$ can be checked in linear time,
- if $\alpha = \alpha_1 \vee \alpha_2$: $s' \in \alpha(s)$ if and only if $s' \in \alpha_1(s)$ or $s' \in \alpha_2(s)$,
- if $\alpha = \alpha_1 \wedge \alpha_2$: $s' \in \alpha(s)$ if and only if $s' \in \alpha_1(s)$ and $s' \in \alpha_2(s)$,
- if $\alpha = \mathsf{C}_{X,V,F}(\beta)$: $s' \in \alpha(s)$ if and only if $s' \in \beta(s)$ and for all s'': if $(s''\Delta s) \cap X \subsetneq (s'\Delta s) \cap X$ and $s'' \cap F = s' \cap F$ then $s'' \notin \beta(s)$.

Each of the above steps can be performed in polynomial space (the last step requires a loop, but can reuse the same space again and again).

The complement of IS-APPLIC is reducible to IS-SUCC: the $\mathbf{NNFAT}_{\mathsf{C}}$ action α over the variables P is non-applicable in s if and only if for a fresh variable q it holds that $\{q\} \in \mathsf{C}_{\{q\},P,\emptyset}(\alpha \vee q')(s)$. Indeed, the only case when setting q to $\top$ is a minimal change of its value is when there is no α-successor of s, and conversely, if q' can be set to $\bot$ then there must exist an α-successor of s.

As for ENTAILS: whether a given sequence $\alpha_1; \ldots; \alpha_k$ entails ψ in s can be checked by verifying for each sequence of states $s_1, \ldots, s_k$ whether $s_1 \in \alpha_1(s)$, $s_2 \in \alpha_2(s_1)$, $\ldots$, $s_k \in \alpha_k(s_{k-1})$ and for each sequence for which this is true: whether $s_k \models \psi$. This can be performed in polynomial space and thus ENTAILS is in PSPACE, too.

ST is a special case of IS-SUCC and thus in PSPACE.

As for IS-MON: to show that α is not positively monotone in s we give a successor $s' \in \alpha(s)$ with $s \not\subseteq s'$. This can be verified in polynomial space and thus the complement of IS-MON is in NPSPACE = PSPACE and thus IS-MON is in coPSPACE = PSPACE, and the case with negative monotony follows by symmetry. Analogous argumentation works for IS-DET: α is non-deterministic in s if there are two α-successors s', s'' of s with $s' \neq s''$, which can be verified in polynomial space. □

We now turn to the hardness results. We recall that for a formula ψ over the variables $\{q_j \mid j \in J\}$ we write ψ' for the formula obtained by replacing all variables q_j by q'_j.

Proposition 9.21. Is-Succ *is* PSPACE*-hard for* $\mathbf{NNFAT}^{\mathbf{C}}_{\mathsf{C}}$.

PROOF. We modify Notation 6.4 by introducing for every variable x_j two variables q_j, r_j and write $q_j \in \gamma_i$ if $x_j \in \gamma_i$, and $r_j \in \gamma_i$ if $\neg x_j \in \gamma_i$. We also use the notation χ_n from the proof of Proposition 6.39, with $\chi_n := \bigwedge_{i=1}^{N_n}(\neg p_i \vee \bigvee_{\ell \in \gamma_i} \ell)$. We set $Q_n := \{q_j, r_j \mid 1 \leq j \leq n\}$. $S_n := Q_n \cup \{p_1, \ldots, p_{N_n}\}$. We obtain β^n_{n+1} by replacing all x_j in χ_n by q'_j and all $\neg x_j$ by r'_j and then define recursively

$$\beta^n_i := \mathsf{C}_{\{q_i,r_i\},\emptyset,S_n\setminus\{q_i,r_i\}}\big((q'_i \wedge r'_i) \vee (q'_i \wedge \beta^n_{i+1}) \vee (r'_i \wedge \beta^n_{i+1})\big)$$

The double occurence of β^n_{i+1} does not lead to an explosion since it needs to be stored only once for the circuit, and thus β^n_1 can be constructed in polynomial time.

Let $\Phi := \forall x_1 : \exists x_2 : \ldots : \forall x_n : \varphi$, which is logically equivalent to $\forall x_1 : \neg(\forall x_2 : \neg(\ldots \neg(\forall x_n : \varphi)\ldots)$, be a quantified Boolean formula with a 3-CNF φ. Deciding the validity of such formulas is PSPACE-complete. We claim that Φ is true if and only if $s(\varphi) \cup Q_n \in \beta^n_1(s(\varphi))$.

Indeed, first observe that by minimal change over q_i, r_i for all i and all states $s \subseteq S_n \setminus \{q_i, r_i\}$, $t \subseteq \{p_1, \ldots, p_{N_n}\}$: $s \cup \{q_i, r_i\} \in \beta^n_i(t) \Leftrightarrow s \cup \{q_i\}, s \cup \{r_i\} \notin \beta^n_{i+1}(t)$. We set $V_i := \{q_j, r_j \mid i \leq j \leq n\}$ and $W_i := \{q_i, r_i\}$ and it follows with $t := s(\varphi)$

$$\begin{aligned}
& s(\varphi) \cup Q_n \in \beta^n_1(s(\varphi)) \\
\Leftrightarrow\ & s(\varphi) \cup V_2 \cup \{q_1\}, s(\varphi) \cup V_2 \cup \{r_1\} \notin \beta^n_2(s(\varphi)) \\
\Leftrightarrow\ & \forall z_1 \in W_1 : \neg(s(\varphi) \cup V_2 \cup \{z_1\} \in \beta^n_2(s(\varphi))) \\
& \quad \ldots \\
\Leftrightarrow\ & \forall z_1 \in W_1 : \neg\big(\forall z_2 \in W_2 : \neg(\forall z_3 \in W_3 : \neg \ldots (s(\varphi) \cup \{z_1, \ldots, z_n\} \in \beta^n_{n+1}(s(\varphi))))\big)
\end{aligned}$$

Now $s(\varphi) \cup \{z_1, \ldots, z_n\} \in \beta^n_{n+1}(s(\varphi))$ in the last line is equivalent to φ being true under the assignment defined by $x_i := (z_i = q_i)$. We have proven the claim and thus PSPACE-hardness of IS-SUCC. □

Proposition 9.22. Entails *and* Is-Applic *are* PSPACE*-complete for* $\mathbf{NNFAT}^{\mathbf{C}}_{\mathsf{C}}$.

PROOF. Membership follows from Proposition 9.20. For hardness of IS-APPLIC observe: $s' \in \alpha(s)$ if and only if $\alpha \wedge \psi_{s'}$ (with $\psi_{s'}$ as in Notation 7.3, page 56) is applicable in s.

Hardness of ENTAILS follows with the reduction of non-applicability to ENTAILS (Lemma 4.7, page 23). □

Proposition 9.23. ST *is linear-time solvable for* $\mathbf{NNFAT}^{\mathbf{C}}_{\mathsf{C}}$.

PROOF. The proof for **NNFAT** (Proposition 6.26, page 44) can be extended by the fact that $s \in \mathsf{C}_{X,V,F}(\alpha)(s)$ if and only if $s \in \alpha(s)$, because, on the one hand, $\mathsf{C}_{X,V,F}(\alpha)(s) \subseteq \alpha(s)$, and on the other hand, since no variable changes its value in the transition from s to s, the change to the variables X is minimal for every partition X, V, F. □

Proposition 9.24. Is-Det *and* Is-Mon *are* PSPACE*-complete for* $\mathbf{NNFAT}^{\mathbf{C}}_{\mathsf{C}}$.

PROOF. Membership follows from Proposition 9.20. The hardness proofs are typical: an $\mathbf{NNFAT}^{\mathbf{C}}_{\mathsf{C}}$ action α is non-applicable in s if and only if for a fresh variable q the action $\alpha \wedge (q' \vee \neg q')$ is deterministic in s, and it is also non-applicable in s if and only if $\alpha \wedge \neg q'$ is positively monotone in $s \cup \{q\}$. Together with PSPACE-hardness of IS-APPLIC (Proposition 9.22) we conclude PSPACE-hardness of IS-DET and IS-MON. □

We finally turn to transformations and succinctness. Due to a different semantics we cannot reuse the proofs from Chapter 8, and since it turned out that reasoning about succinctness and transformations is especially hard when languages get very rich, we do not have many results for $\mathbf{NNFAT}_{\mathsf{C}}$.

Remark 9.25. CHOICE for α and β in $\mathbf{NNFAT}_{\mathsf{C}}$ can be performed via $\alpha \vee \beta$.

Corollary 9.26. Extract-Precond *is not possible in polynomial size if* $\mathsf{NP} \not\subseteq \mathsf{P/poly}$.

PROOF. If $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ then EXTRACT-PRECOND is not polynomial-size already for **NNFAT** which is a sublanguage of $\mathbf{NNFAT}_{\mathsf{C}}$. □

The next proof is analogous to other succinctness proofs and uses the fact that a query (namely, IS-SUCC) for one language is harder than for the other.

Proposition 9.27. *If* $\mathsf{PSPACE} \not\subseteq \mathsf{P/poly}$ *then there is no polynomial-size translation from* $\mathbf{NNFAT}^{\mathbf{C}}$ *into* $\mathbf{NNFAT}^{\mathbf{C}}_{\mathsf{C}}$. *If* $\mathsf{PSPACE} \not\subseteq \mathsf{NP/poly}$ *then then there is no polynomial-size translation from* $\mathbf{NNFAT}^{\mathbf{C}}_{\mathsf{C}}$ *into* $\mathbf{O\text{-}PDDL}^{\mathbf{C}}$ *nor* $\mathbf{E\text{-}PDDL}^{\mathbf{C}}$.

PROOF. Recall from the proof of Proposition 9.21 that a QBF $\forall x_1 : \exists x_2 : \ldots \forall x_n : \varphi$ with a 3-CNF matrix φ is true if and only if $s(\varphi) \cup Q_n \in \beta_1^n(s(\varphi))$ with $Q_n = \{q_j, r_j \mid 1 \leq j \leq n\}$. If there existed a polynomial-size translation from $\mathbf{NNFAT}^{\mathbf{C}}_{\mathsf{C}}$ into $\mathbf{NNFAT}^{\mathbf{C}}$ we would obtain a non-uniform polynomial-time algorithm for a PSPACE-complete problem. The proof is analogous for $\mathbf{O\text{-}PDDL}^{\mathbf{C}}$ and $\mathbf{E\text{-}PDDL}^{\mathbf{C}}$, but the assumption must be $\mathsf{PSPACE} \not\subseteq \mathsf{NP/poly}$ because IS-SUCC is NP-complete for $\mathbf{O\text{-}PDDL}^{\mathbf{C}}$ and $\mathbf{E\text{-}PDDL}^{\mathbf{C}}$ (Proposition 6.9, page 40). □

Proposition 9.28. *If* $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ *then there exists no polynomial-size translation from* $\mathbf{O\text{-}PDDL}^{\mathbf{C}}$ *or any of its extensions into* $\mathbf{NNFAT}^{\mathbf{C}}_{\mathsf{C}}$.

PROOF. Recall from Proposition 6.27 (page 45)that a 3-CNF φ is satisfiable if and only if $s(\varphi) \in \beta_n(s(\varphi))$. If $\mathbf{O\text{-}PDDL}^{\mathbf{C}}$ was polynomial-size translatable into $\mathbf{NNFAT}^{\mathbf{C}}_{\mathsf{C}}$ there would exist a polynomial-time algorithm for 3-SAT because ST is linear-time for $\mathbf{NNFAT}^{\mathbf{C}}_{\mathsf{C}}$ by Proposition 9.23. □

9.3 Conclusion

We studied extensions of **NNF** action theories with connectives expressing two different types of persistency of variables in the circuit representation, with the goal of enriching the language. It turns out that using a frame connective to adapt the semantics of **PDDL** at any level of nesting does not change time nor space complexity; hence this connective can be used when specifying actions, then compiled away efficiently so as to use algorithms designed for (standard) NNF action theories. The language resulting for our second connective (related to the interpretation of formulas under circumscription) is more succinct but also has a greater complexity for basic queries.

The question whether $\mathbf{NNFAT}_{\mathsf{C}}^{\mathbf{C}}$ supports polynomial-size SEQUENCE or NEGATION remains open, as well as its succinctness relative to extensions of imperative languages.

Our complexity results are summarized in Table 9.1. We do not give a succinctness diagram because we have only partial succinctness results and thus the diagram would not be illustrative at this point.

Query/Transformation	$\mathbf{NNFAT}^{\mathbf{C}}_{\mathsf{F}}$	$\mathbf{NNFAT}^{\mathbf{C}}_{\mathsf{C}}$
Queries		
IS-SUCC	polynomial time	PSPACE-complete
IS-APPLIC	NP-complete	PSPACE-complete
ENTAILS	coNP-complete	PSPACE-complete
ST	polynomial time	linear time
IS-DET	coNP-complete	PSPACE-complete
IS-MON	coNP-complete	PSPACE-complete
Transformations		
CHOICE	linear time	linear time
NEGATION	polynomial time	?
SEQUENCE	○	?
EXTRACT-PRECOND	○	○

Table 9.1: Complexity results for circuit representations of extensions of **NNFAT**. ○ means that under some complexity-theoretic assumption the transformation is not polynomial-size. "?" denotes open questions.

Part IV

Conclusion

CHAPTER 10

Conclusion

10.1 Contributions

The contribution of this work is a formal framework for comparing nondeterministic action languages from the perspective of knowledge compilation together with some first results. We studied the succinctness and the complexity of answering queries and performing transformations for two representations (as trees and as circuits) of various (propositional) languages which are complete with respect to expressing nondeterministic actions.

The most important contribution is probably Chapter 6, which contains results for abstract versions of action languages used in "real life", namely, **NNFAT**, **O-PDDL** and **E-PDDL**. For these languages (which we called minimally complete) we obtained a complete picture regarding their mutual relative succinctness, the time complexity of answering queries and the possibility of performing polynomial-size transformations.

We introduced queries which we consider to be natural for planning and which fit well into the knowledge compilation framework: asking whether an action is applicable in a given state (IS-APPLIC) and whether a given action sequence always results in a state that satisfies some given property (ENTAILS). These two queries constitute what is called "belief tracking" in the literature (Bonet and Geffner, 2014). Other queries which we studied in this framework are less common but still important, especially the "model checking of planning", IS-SUCC. The results about the queries IS-DET and IS-MON are usually easily derived from results about IS-APPLIC, and results for ST are rather a technical tool for succinctness separation proofs.

As we added further connectives to the minimally complete languages, we asked ourselves whether the semantics of these connectives can be simulated at least once, i.e. we studied transformations motivated by aiming to express connectives with other connectives. This is e.g. motivated by results from the classical knowledge compilation map of Darwiche and Marquis (for example, the language MODS satisfies polynomial-time bounded $\wedge$-closure and does not satisfy polynomial-time general $\wedge$-closure, whereas there are other languages which do not satisfy even bounded $\wedge$-closure). The results we have proven agree with the intuition that if a grammar does not have a specific connective then it will be hard to mimick this connective, but many questions remain open, e.g. whether a single action negation can be expressed in $\mathbf{E\text{-}PDDL}_{\text{seq}}$ when using the circuit representation.

The most important and technically difficult contribution of this work are the succinctness separation results. Although it might seem that the complexity proofs for queries are more technical, it is actually the succinctness proofs which require the complexity proofs to be of a certain form, which is more general than necessary. For example, the **E-PDDL** action $\alpha_n^{\text{sat},\sqcap}$ (Notation 6.6, page 40) is intentionally defined to produce *all* satisfiable 3-CNFs, although a reduction from deciding satisfiability of a concrete family of 3-CNF's to IS-SUCC would be sufficient to show NP-hardness.

The most important part of the succinctness study of our work is the complete succinctness diagram from Chapter 6. We have shown that **O-PDDL** with its "asymmetric" parallel composition is succinctness-incomparable from **E-PDDL** and **NNFAT** and has nice computational properties (deciding applicability in polynomial-time), which indicates that there necessarily exist actions which allow for much higher computational efficiency when represented in **O-PDDL** as compared to **E-PDDL**.

As for interesting/surprising observations, we can point out at least three of them. The first is that the complexity of queries for "basic" minimally complete languages does not depend on the chosen representation (tree or circuit). This is in contrast with more expressive extensions of the basic languages, where the representation may indeed crucially affect the complexity.

The more specific second observation from Chapters 5 and 9 is related to the first one: it seems that the connectives of parallel composition $\sqcap$ (or &) together with nondeterminism are already sufficient to provoke nontractability of IS-SUCC, and the hardness results remain valid even for the very restricted syntax of **O-NCSTRIPS** and **E-NCSTRIPS**. $\alpha_n^{\mathrm{sat},\sqcap}$ (Notation 6.6, page 40) is very simply defined, does not involve neither conditional execution $\rhd$ nor negative assignments $-p$ (indirectly it involves $\rhd$ to define ε, though, but allowing the "skip" action to be part of the grammar is natural (Rintanen, 2003; Balbiani et al., 2013)). We can say that the main reason for **E-PDDL** being more complex and succinct than **NNFAT** is not the persistence of variables (because the syntactic persistence connective F_X is intended to mimick the semantics of **E-PDDL**), but that $\alpha \sqcap \beta$ can produce new successors in s which have not been successors of s neither via α nor via β. This contradicts the first intuition we had (and which motivated us to study $\mathbf{NNFAT}_\mathsf{F}$: we conjectured that it would be more succinct than **NNFAT** before having proven the converse for circuits).

The third observation is that in this thesis there is a very general succinctness scheme: if a language has a hard query then it is not polysize-translatable into a language where this query is easier. This is not obvious in general, but follows by construction of our hardness proofs. In contrast, the observation that any language supporting polynomial-size sequence is not translatable into a language not supporting polynomial-size sequence is straightforward because for SEQUENCE we are interested in the size of the output, as well as in succinctness questions.

10.2 Perspectives

We leave some problems open for transformations (see Table 10.3) and succinctness. For succinctness, we summarized the results for the minimally complete languages and the **STRIPS**-like restrictions in a succinctness diagram in Figure 10.1. For the extensions we reuse the diagrams from Chapter 8 in Figure 10.2, because all languages in one diagram would look confusing. In this diagram, a dotted arc from L_1 to L_2 on means that the existence of a polynomial-size translation from L_1 to L_2 is open; for all these arcs we conjecture that there is actually none. Furthermore, only little is known about the succinctness relations between the extensions. Filling all those gaps would be a direct continuation of this work, but these problems seem very hard to tackle, except for the gaps from Chapter 9: we conjecture that all results which were proven there for circuits also hold for tree representations (especially, we conjecture the existence of a polynomial-time translation from $\mathbf{NNFAT}_\mathsf{F}^\mathbf{T}$ into $\mathbf{NNFAT}^\mathbf{T}$ and PSPACE-completeness of IS-SUCC for $\mathbf{NNFAT}_\mathsf{C}^\mathbf{T}$).

In the following we will give some research questions which were beyound the scope of this thesis. In our setting it would be interesting to determine the complexity of other queries and transformations.

Further queries

Queries obviously useful to planning are to count and enumerate successors, generate a successor of a given state uniformly at random (as needed in Monte-Carlo approaches), test whether a given action sequence can cycle, and to determine whether all executions of a given action sequence are free of dead-ends (in contrast to ENTAILS, which when all executions result in dead ends returns true).

These queries could be formalized as follows (we assume, as before, that all action descriptions α, α_i have an interpretation over the scope P with $s \subseteq P$):

- RAND-UN-SUCC: for given α, s, output $s' \in \alpha(s)$ at random with a uniform probability distribution,
- ENUM-SUCC: for given α, s, output $\alpha(s)$,
- HAS-CYCLE: for given $\alpha_1, \ldots, \alpha_k$, s, decide if there exist $j, m \leq k$ with $m > j$ and s' such that $s' \in (\alpha_1 ; \ldots ; \alpha_j)(s)$ and $s' \in (\alpha_{j+1} ; \ldots ; \alpha_m)(s')$
- HAS-SUBEFFECT (for languages whose semantic is defined via explicit effects): for given α, s and effect $\langle Q^+, Q^- \rangle$ decide whether α has an effect $\langle A^+, A^- \rangle$ in s with $A^+ \subseteq Q^+, A^- \subseteq Q^-$,
- HAS-NON-SUCC: for given α, s decide whether there exists $s' \notin \alpha(s)$,
- NO-FAIL: for given $\alpha_1, \ldots, \alpha_k$, s, let $\alpha_0 := \varepsilon$. Decide for all $1 \leq j \leq k$ whether α_j is applicable in all s' such that $s' \in (\alpha_0 ; \ldots ; \alpha_{j-1})(s)$.

For some of these quries we could easily derive their complexity from the results in this work, for others (like RAND-UN-SUCC) it would be more difficult (there exist works on uniformly sampling NP-certificates, e.g. Bellare et al. (2000), which could be useful for generating successors for languages with NP-complete IS-SUCC).

In this thesis and in the definitions above the input of a query always consists of the action description and one particular state. However, we could relax this assumption and input *belief states* (i.e. sets of states) S, and naturally generalize the queries by universally or existentially quantifying over S: for example, two different versions of IS-APPLIC for belief states would be: "is there at least one $s \in S$ such that α is applicable in s?" and "is α applicable in s for all $s \in S$?".

Further transformations

As for transformations, we studied the difficulty of expressing a connective with other connectives. A natural transtormation in this setting would be combining two actions via logical conjunction $\wedge$ (without using $\wedge$, of course). We observed in Chapter 8 that **E-PDDL**$_{\text{and}}$ allows to express for an action α the action which keeps only those state transitions of α where the successor satisfies an NNF formula φ (in other words, it allows to dismiss successors violating φ). This motivates the question whether other languages can do this, too. Another interesting transformation, in particular for regression approaches to planning, is the computation of the "reverse" action of α (leading from s' to s whenever α leads from s to s').

We have seen that conditioning of an action by an assignment t (as defined in Chapter 6) can be performed in polynomial time for **E-PDDL**$_{\text{not}}$, **E-PDDL**$_{\text{and}}$ and **E-PDDL**, but not for **E-PDDL**$_{\text{seq}}$ or variants of **O-PDDL**. Thus we have implicitly shown results about another transformation which could be studied for all action languages.

These transformations can be formalized as follows (we remark that the output action description has to be in the same language as the inputs):

- CONJUNCTION: for given action descriptions α_1, α_2 compute an action description β such that for all s: $\beta(s) = \alpha_1(s) \cap \alpha_2(s)$,
- REVERSE: given α and P, compute an action description $\beta \in L$ s.t. for all s: $\beta(s) = \{s' \mid s \in \alpha(s')\}$,
- ENSURE: given action α and NNF formula φ, compute action β such that for all s: $\beta(s) = \{s' \in \alpha(s) \mid s' \models \varphi\}$,
- CONDITIONING: for given action α, set $Q \subseteq P$ and assignment t to Q compute $P \setminus Q$ action β such that $\beta(s) = \{s' \mid (s' \cup t) \in \alpha(s \cup t)\}$.

The complexity of CONJUNCTION does not seem to be easy to determine (except for variants of **NNFAT** and languages like **E-PDDL**$_{\text{and}}$ and **E-PDDL**$_{\text{not}}$), i.e. we only have shown (almost) trivial positive results,

but almost no impossibility results, and this is the reason why we did not include it into our framework in this thesis.

As for REVERSE, we observe that the reverse of an **NNFAT** expression can be obtained simply by swapping p and p' for all variables p, but there does not seem to be any obvious approach for the other languages, because of parallel execution $\sqcap$.[1] However, as it is shown in Herzig (2014), in **DL-PA** (which does not have parallel composition) the reverse of an action can be computed in linear time without auxiliary variables.

Further languages

Our main perspective is a more systematic study, for languages constructed using combinations of features like the sequence operator, modalities, Kleene star, etc. We could introduce a connective R_X dual to F_X with the semantics (in terms of explicit effects) $\mathrm{E}(\mathsf{R}_X(\alpha),P,s) \coloneqq \{\langle Q^+ \cup A^+, Q^- \cup A^- \rangle \mid A^+,A^- \subseteq X, A^+ \cap A^- = \emptyset\}$. We call this connective R_X because it is motivated by the *release propositions* from Kartha and Lifschitz (1994). Then, as an example, one could consider the language L with the grammar defined by

$$\alpha ::= +p \mid -p \mid \varphi \rhd \alpha \mid \alpha \cup \alpha \mid \alpha \sqcap \alpha \mid \alpha \wedge \alpha \mid \mathsf{R}_X(\alpha)$$

with NNF formulas φ and $X \subseteq P$ for the intended scope $P \subseteq \mathbb{P}$.

We could relax the assumption that formulas are always in NNF, and consider e.g. modal formulas $\langle\alpha\rangle\varphi$ meaning "there exists an execution of α such that φ is true afterwards", as in **DL-PA** (Balbiani et al., 2013), and in general, involving **DL-PA** into our study with the semantics restricted to the scope P would provide many natural questions.

An interesting question would be whether replacing $+p$ and $-p$ in some language by $p \leftarrow \varphi$, as in **DL-PPA**, would produce a more succinct language[2].

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.

Beyound the setting

There are questions which seem natural, but which do not fit into the framework we defined in this thesis. For example, we could allow translations f to introduce auxiliary variables Q with the restriction that they are false in each predecessor- and successor state, i.e. that with $t(q) = \bot$ for all $q \in Q$ the action descriptions α and $f(\alpha)_{|t}$ (defined as in Chapter 6) describe the same action.

Another perspective is to consider languages for stochastic actions, and for actions with observations (and hence queries on belief states, which we mentioned before).

Finally, as we said in chapter 2, our setting is related to that of preferences, and there exists research in knowledge compilation for languages for representing preferences (Fargier and Mengin, 2021). It could be interesting to continue this research by using imperative languages to represent preference relations.

[1] For **DL-PPA**, Herzig et al. (2019) give an algorithm for computing the reverse action (described for **DL-PA** by Herzig (2014)), but it uses auxiliary variables, which we do not allow.

[2] We obtained preliminary results about this question with Frédéric Maris

Language	IS-SUCC	IS-APPLIC	ENTAILS
Minimally complete languages			
NNFAT$^{\mathbf{T}}$, **NNFAT$^{\mathbf{C}}$**	linear time	NP-complete	coNP-complete
O-PDDL$^{\mathbf{T}}$, **O-PDDL$^{\mathbf{C}}$**	NP-complete	linear time	coNP-complete
E-PDDL$^{\mathbf{T}}$, **E-PDDL$^{\mathbf{C}}$**	NP-complete	NP-complete	coNP-complete
Restrictions of imperative languages			
O-NCSTRIPS	NP-complete	linear time	coNP-complete
E-NCSTRIPS	NP-complete	NP-complete	coNP-complete
NPDDL$_{\mathbf{nf}}$	NP-complete	linear time	coNP-complete
Extensions of imperative languages			
O-PDDL$^{\mathbf{T}}_{\mathbf{seq}}$	NP-complete	NP-complete	coNP-complete
O-PDDL$^{\mathbf{C}}_{\mathbf{seq}}$	PSPACE-complete	PSPACE-complete	PSPACE-complete
O-PDDL$^{\mathbf{T}}_{\mathbf{and}}$	NP-complete	NP-complete	coNP-complete
O-PDDL$^{\mathbf{C}}_{\mathbf{and}}$	PSPACE-complete	PSPACE-complete	PSPACE-complete
O-PDDL$^{\mathbf{T}}_{\mathbf{not}}$, **O-PDDL$^{\mathbf{C}}_{\mathbf{not}}$**	PSPACE-complete	PSPACE-complete	PSPACE-complete
E-PDDL$^{\mathbf{T}}_{\mathbf{seq}}$	NP-complete	NP-complete	coNP-complete
E-PDDL$^{\mathbf{C}}_{\mathbf{seq}}$	PSPACE-complete	PSPACE-complete	PSPACE-complete
E-PDDL$^{\mathbf{T}}_{\mathbf{and}}$	NP-complete	NP-complete	coNP-complete
E-PDDL$^{\mathbf{C}}_{\mathbf{and}}$	PSPACE-complete	PSPACE-complete	PSPACE-complete
E-PDDL$^{\mathbf{T}}_{\mathbf{not}}$, **E-PDDL$^{\mathbf{C}}_{\mathbf{not}}$**	PSPACE-complete	PSPACE-complete	PSPACE-complete
Extensions of **NNFAT**			
NNFAT$^{\mathbf{C}}_{\mathsf{F}}$	polynomial time	NP-complete	coNP-complete
NNFAT$^{\mathbf{C}}_{\mathsf{C}}$	PSPACE-complete	PSPACE-complete	PSPACE-complete

Table 10.1: Complexity results for IS-SUCC, IS-APPLIC and ENTAILS.

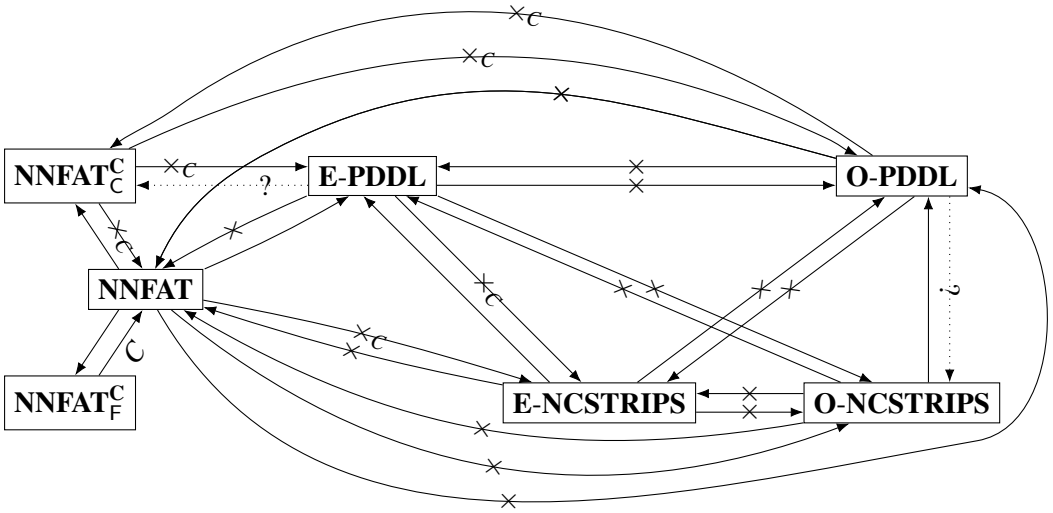
Language	ST	IS-DET	IS-MON
Minimally complete languages			
NNFAT$^{\mathbf{T}}$, **NNFAT$^{\mathbf{C}}$**	linear time	coNP-complete	coNP-complete
O-PDDL$^{\mathbf{T}}$, **O-PDDL$^{\mathbf{C}}$**	NP-complete	polynomial time	polynomial time
E-PDDL$^{\mathbf{T}}$, **E-PDDL$^{\mathbf{C}}$**	linear time	coNP-complete	coNP-complete
Restrictions of imperative languages			
O-NCSTRIPS	NP-complete	polynomial time	polynomial time
E-NCSTRIPS	linear time	NP-complete	coNP-complete
NPDDL$_{\mathbf{nf}}$	linear time	polynomial time	trivial/polynomial time
Extensions of imperative languages			
O-PDDL$^{\mathbf{T}}_{\mathbf{seq}}$	NP-complete	coNP-complete	coNP-complete
O-PDDL$^{\mathbf{C}}_{\mathbf{seq}}$	PSPACE-complete	PSPACE-complete	PSPACE-complete
O-PDDL$^{\mathbf{T}}_{\mathbf{and}}$	NP-complete	coNP-complete	coNP-complete
O-PDDL$^{\mathbf{C}}_{\mathbf{and}}$	PSPACE-complete	PSPACE-complete	PSPACE-complete
O-PDDL$^{\mathbf{T}}_{\mathbf{not}}$, **O-PDDL$^{\mathbf{C}}_{\mathbf{not}}$**	PSPACE-complete	PSPACE-complete	PSPACE-complete
E-PDDL$^{\mathbf{T}}_{\mathbf{seq}}$	NP-complete	coNP-complete	coNP-complete
E-PDDL$^{\mathbf{C}}_{\mathbf{seq}}$	PSPACE-complete	PSPACE-complete	PSPACE-complete
E-PDDL$^{\mathbf{T}}_{\mathbf{and}}$	linear time	coNP-complete	coNP-complete
E-PDDL$^{\mathbf{C}}_{\mathbf{and}}$	linear time	PSPACE-complete	PSPACE-complete
E-PDDL$^{\mathbf{T}}_{\mathbf{not}}$, **E-PDDL$^{\mathbf{C}}_{\mathbf{not}}$**	linear time	PSPACE-complete	PSPACE-complete
Extensions of **NNFAT**			
NNFAT$^{\mathbf{C}}_{\mathsf{F}}$	polynomial time	coNP-complete	coNP-complete
NNFAT$^{\mathbf{C}}_{\mathsf{C}}$	linear time	PSPACE-complete	PSPACE-complete

Table 10.2: Complexity results for ST, IS-DET and IS-MON.

Language	CHOICE	SEQUENCE	NEGATION	EXTRACT-PRECOND
Minimally complete languages				
NNFAT$^{\mathbf{T}}$, **NNFAT$^{\mathbf{C}}$**	✓	○	✓	○
O-PDDL$^{\mathbf{T}}$, **O-PDDL$^{\mathbf{C}}$**	✓	○	○	✓
E-PDDL$^{\mathbf{T}}$, **E-PDDL$^{\mathbf{C}}$**	✓	○	○	○
Restrictions of imperative languages				
O-NCSTRIPS	?	○	○	✓
E-NCSTRIPS	?	○	○	○
NPDDL$_{\mathbf{nf}}$	✓	○	○	✓
Extensions of imperative languages				
O-PDDL$^{\mathbf{T}}_{\text{seq}}$	✓	✓	○	○
O-PDDL$^{\mathbf{C}}_{\text{seq}}$	✓	✓	?	○
O-PDDL$^{\mathbf{T}}_{\text{and}}$	✓	?	○	○
O-PDDL$^{\mathbf{C}}_{\text{and}}$	✓	?	?	○
O-PDDL$^{\mathbf{T}}_{\text{not}}$, **O-PDDL$^{\mathbf{C}}_{\text{not}}$**	✓	?	✓	○
E-PDDL$^{\mathbf{T}}_{\text{seq}}$	✓	✓	○	○
E-PDDL$^{\mathbf{C}}_{\text{seq}}$	✓	✓	?	○
E-PDDL$^{\mathbf{T}}_{\text{and}}$	✓	○	○	○
E-PDDL$^{\mathbf{C}}_{\text{and}}$	✓	○	?	○
E-PDDL$^{\mathbf{T}}_{\text{not}}$, **E-PDDL$^{\mathbf{C}}_{\text{not}}$**	✓	○	✓	○
Extensions of **NNFAT**				
NNFAT$^{\mathbf{C}}_{\mathsf{F}}$	✓	○	✓	○
NNFAT$^{\mathbf{C}}_{\mathsf{C}}$	✓	?	?	○

Table 10.3: Difficulty of performing a transformation. "✓" means that the transformation can be done in time polynomial in the size of the input. "?" means that the question is open. ○ means that under some complexity-theoretic assumption (see formal statements) the size of the result of the transformation is in general not polynomial in the size of the input.

Figure 10.1: Succinctness results for minimally complete languages, complete restrictions and circuit representations of **NNFAT** (i.e. $\mathbf{NNFAT}^{\mathbf{C}}_{\mathbf{C}}$ and $\mathbf{NNFAT}^{\mathbf{C}}_{\mathbf{F}}$). An arc from L_1 to L_2 means that L_1 can be translated into L_2 in polynomial time (hence also in polynomial size). A crossed out arc from L_1 to L_2 means that there exists not even a polynomial-size translation from L_1 into L_2. These relations hold for both tree and circuit representations. An arc crossed out by $\times_C$ means that the non-existence of a polynomial-size translation is proven only for circuits. An arc labeled by C means that the translation works only into the circuit representation of the destination.



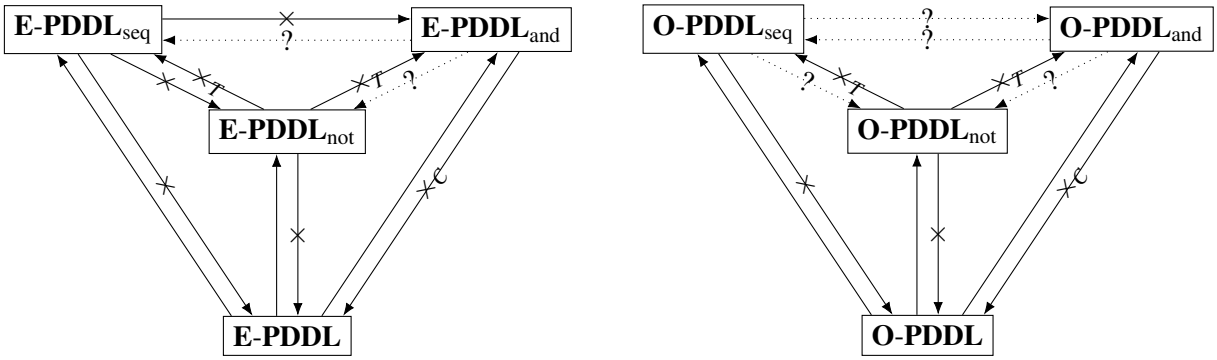


Figure 10.2: Succinctness results for extensions of **E-PDDL** and **O-PDDL**. An arc crossed out by $\times_T$ means that the non-existence of a polynomial-size translation is proven only for trees.

Bibliography

R. Alford, U. Kuter, and D. Nau. Translating HTNs to PDDL: A small amount of domain knowledge can go a long way. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 1629–1634, 01 2009.

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

C. Bäckström. Expressive Equivalence of Planning Formalisms. *Artificial Intelligence*, 76(1-2):17–34, 1995.

C. Bäckström and P. Jonsson. Algorithms and limits for compact plan representations. *Journal of Artificial Intelligence Research*, 44:141–177, 2012.

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

M. Bellare, O. Goldreich, and E. Petrank. Uniform Generation of NP-Witnesses using an NP-oracle. *Information and Computation*, 163(2):510–526, 2000.

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

B. Bonet and H. Geffner. Belief tracking for planning with sensing: Width, complexity and approximations. *Journal of Artificial Intelligence Research*, 50:923–970, 2014.

A. Botea, M. Enzenberger, M. Müller, and J. Schaeffer. Macro-FF: Improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research*, 24:581–621, 2005.

R. I. Brafman and G. Shani. Online belief tracking using regression for contingent planning. *Artificial Intelligence*, 241:131–152, 2016.

J. Broersen. Action negation and alternative reductions for dynamic deontic logics. *Journal of applied logic*, 2(1):153–168, 2004.

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

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

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

M. Cadoli and F. M. Donini. A Survey on Knowledge Compilation. *AI Communications*, 10(3-4): 137–150, 1997.

F. Capelli, J. Lagniez, and P. Marquis. Certifying Top-Down Decision-DNNF Compilers. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, pages 6244–6253. AAAI Press, 2021.

T. Charrier and F. Schwarzentruber. A Succinct Language for Dynamic Epistemic Logic. In *Proceedings of the Sixteenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017)*, pages 123–131, 2017.

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

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

T. Eiter and G. Gottlob. Propositional Circumscription and Extended Closed-World Reasoning are ΠP2-complete. *Theoretical Computer Science*, 114(2):231–245, 1993.

H. Fargier and P. Marquis. Extending the Knowledge Compilation Map: Krom, Horn, Affine and Beyond. In *AAAI*, pages 442–447, 2008.

H. Fargier and J. Mengin. A Knowledge Compilation Map for Conditional Preference Statements-based Languages. In *Proceedings of the Twentieth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021)*, pages 492–500, 2021.

H. Fargier, P. Marquis, and A. Niveau. Towards a Knowledge Compilation Map for Heterogeneous Representation Languages. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI 2013)*, pages 877–883, 2013.

R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of computer and system sciences*, 18(2):194–211, 1979.

M. Fox and D. 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.

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

M. R. Garey and D. S. Johnson. Computers and Intractability. *A Guide to the*, 1979.

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

T. Geffner and H. Geffner. Compact Policies for Fully Observable Non-Deterministic Planning as SAT. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS 2018)*, pages 88–96, 2018.

I. Georgievski and M. Aiello. HTN planning: Overview, comparison, and beyond. *Artificial Intelligence*, 222:124–156, 2015.

M. Ghallab, D. S. Nau, and P. Traverso. *Automated planning and acting*. Cambridge University Press, 2016.

D. Gnad, A. Torralba, M. Domínguez, C. Areces, and F. Bustos. Learning how to ground a plan–partial grounding in classical planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7602–7609, 2019.

J. Hastad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 6–20, 1986.

M. Helmert. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26: 191–246, 2006.

A. Herzig. Belief Change Operations: A Short History of Nearly Everything, Told in Dynamic Logic of Propositional Assignments. In *Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning*, 2014.

A. Herzig, F. Maris, and J. Vianey. Dynamic logic of parallel propositional assignments and its applications to planning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI 2019)*, pages 5576–5582, 2019.

J. Hoey, R. St Aubin, and C. Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of Uncertainty in Artificial Intelligence (UAI). Stockholm, Sweden. Page (s)*, 1999.

J. Hoffmann and R. Brafman. Conformant Planning via Heuristic Forward Search: A New Approach. *Artificial Intelligence*, 170(6–7):507–541, 2006.

G. N. Kartha and V. Lifschitz. Actions with Indirect Effects (Preliminary Report). In *Principles of Knowledge Representation and Reasoning*, pages 341–350. Elsevier, 1994.

J. Lang, P. Liberatore, and P. Marquis. Propositional independence: formula-variable independence and forgetting. *Journal of Artificial Intelligence Research*, 18:391–443, 2003.

B. Lesner and B. Zanuttini. Efficient Policy Construction for MDPs Represented in Probabilistic PDDL. In *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling (ICAPS 2011)*. AAAI Press, 2011.

V. Lifschitz. On the Semantics of STRIPS. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pages 1–9, 1987.

N. Lipovetzky and H. Geffner. Width and Serialization of Classical Planning Problems. In *ECAI*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, pages 540–545. IOS Press, 2012.

J. McCarthy. Circumscription—a form of non-monotonic reasoning. *Artificial intelligence*, 13(1-2): 27–39, 1980.

D. 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).

C. Meinel and T. Theobald. *Algorithms and Data Structures in VLSI Design: OBDD-foundations and applications*. Springer Science & Business Media, 1998.

C. J. Muise, S. A. McIlraith, and V. Belle. Non-Deterministic Planning With Conditional Effects. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*, pages 370—374. AAAI Press, 2014.

D. E. Muller and F. P. Preparata. Bounds to Complexities of Networks for Sorting and for Switching. *Journal of the ACM (JACM)*, 22(2):195–201, 1975.

D. S. Nau. Current trends in automated planning. *AI magazine*, 28(4):43–43, 2007.

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

G. Nordh. A Trichotomy in the Complexity of Propositional Circumscription. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pages 257–269. Springer, 2005.

U. Oztok and A. Darwiche. A Top-Down Compiler for Sentential Decision Diagrams. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)*, pages 3141–3148. AAAI Press, 2015.

H. Palacios and H. Geffner. Compiling Uncertainty Away in Conformant Planning Problems with Bounded Width. *Journal of Artificiall Intelligence Research*, 35:623–675, 2009.

R. Reiter. The Frame Problem in the Situation Calculus: A Simple Solution (Sometimes) and a Completeness Result for Goal Regression. In *Artificial and Mathematical Theory of Computation*, 1991.

J. Rintanen. Expressive Equivalence of Formalisms for Planning with Sensing. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003)*, pages 185–194. AAAI Press, 2003.

J. Rintanen. Complexity of planning with partial observability. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pages 345–354. AAAI Press, 2004.

S. Scheck, A. Niveau, and B. Zanuttini. Knowledge Compilation for Action Languages. In *Journées Francophones sur la Planification, la Décision et l'Apprentissage pour la conduite de systèmes (JFPDA 2020)*, 2020.

S. Scheck, A. Niveau, and B. Zanuttini. Explicit Representations of Persistency for Propositional Action Theories. In *Journées Francophones sur la Planification, la Décision et l'Apprentissage pour la conduite de systèmes (JFPDA 2021)*, 2021a.

S. Scheck, A. Niveau, and B. Zanuttini. Knowledge Compilation for Nondeterministic Action Languages. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, pages 308–316, 2021b.

S. Scheck, A. Niveau, and B. Zanuttini. A KC Map for Variants of Nondeterministic PDDL. In *16es Journées d'Intelligence Artificielle Fondamentale (JIAF 2022)*, 2022.

D. Speck, F. Geißer, and R. Mattmüller. Symbolic Planning with Edge-Valued Multi-Valued Decision Diagrams. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS 2018)*, pages 250–258. AAAI Press, 2018.

D. Speck, D. Borukhson, R. Mattmüller, and B. Nebel. On the Compilability and Expressive Power of State-Dependent Action Costs. In *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)*, pages 358–366. AAAI Press, 2021.

S. Thiébaux, J. Hoffmann, and B. Nebel. In defense of PDDL axioms. *Artificial Intelligence*, 168:38–69, 2005.

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

J. van Benthem, J. van Eijck, M. Gattinger, and K. Su. Symbolic Model Checking for Dynamic Epistemic Logic — S5 and Beyond. *Journal of Logic and Computation*, 28(2):367—402, 2018.

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

APPENDIX A

Table of Notation

Throughout the table we assume P to be a finite set of state variables $p \in P$, the sets X, V, F will constitute a partition of P, the set Q will be disjoint from P, ℓ will denote elementary assignments ($+p$ or $-p$), φ will be an NNF formula over P and α, β will be action descriptions which have an interpretation over P, and a will be a P-action.

Notation	Notion	Explanation/Details
	General	
$\|\varphi\|$	Models	$\{s \mid s \models \varphi\}$
$\|\alpha\|$	State transitions	$\{(s,s') \mid s' \in \alpha(s)\}$
$\mathrm{V}(\varphi)$, $\mathrm{V}(\alpha)$	Set of variables occuring in a formula or action description	
NNF	Negation Normal Form	$\varphi ::= p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi$
	Actions and Connectives	
$\bot$	Failure	$\mathrm{E}(\bot, P, s) = \emptyset$
ε	Action "skip"	$\mathrm{E}(\varepsilon, P, s) := \{\langle \emptyset, \emptyset \rangle\}$
$+p$, $-p$	Elementary assignments	$\mathrm{E}(+p, P, s) := \{\langle \{p\}, \emptyset \rangle\}$, $\mathrm{E}(-p, P, s) := \{\langle \emptyset, \{p\} \rangle\}$
$\rhd$	Conditional execution	$\mathrm{E}(\varphi \rhd \alpha, P, s) := \begin{cases} \mathrm{E}(\alpha, P, s) & \text{if } s \models \varphi, \\ \{\langle \emptyset, \emptyset \rangle\} & \text{otherwise} \end{cases}$
$\cup$	Nondeterministic choice	$\mathrm{E}(\alpha \cup \beta, P, s) := \mathrm{E}(\alpha, P, s) \cup \mathrm{E}(\beta, P, s)$
$\sqcap$	Parallel composition (mutex)	$\mathrm{E}(\alpha \sqcap \beta, P, s) := \{\langle Q_\alpha^+ \cup Q_\beta^+, Q_\alpha^- \cup Q_\beta^- \rangle \mid \langle Q_\alpha^+, Q_\alpha^- \rangle \in \mathrm{E}(\alpha, P, s), \langle Q_\beta^+, Q_\beta^- \rangle \in \mathrm{E}(\beta, P, s), Q_\alpha^+ \cap Q_\beta^- = Q_\alpha^- \cap Q_\beta^+ = \emptyset\}$
$\&$	Parallel composition ($+p$ overrides $-p$)	$\mathrm{E}(\alpha \,\&\, \beta, P, s) := \{\langle Q_\alpha^+ \cup Q_\beta^+, (Q_\alpha^- \cup Q_\beta^-) \setminus (Q_\alpha^+ \cup Q_\beta^+) \rangle \mid \langle Q_\alpha^+, Q_\alpha^- \rangle \in \mathrm{E}(\alpha, P, s), \langle Q_\beta^+, Q_\beta^- \rangle \in \mathrm{E}(\beta, P, s)\}$
;	Sequential composition	$\mathrm{E}(\alpha ; \beta, P, s) := \{\langle Q_\beta^+ \cup (Q_\alpha^+ \setminus Q_\beta^-), Q_\beta^- \cup (Q_\alpha^- \setminus Q_\beta^+) \rangle \mid \langle Q_\alpha^+, Q_\alpha^- \rangle \in \mathrm{E}(\alpha, P, s), t := (s \cup Q_\alpha^+) \setminus Q_\alpha^-, \langle Q_\beta^+, Q_\beta^- \rangle \in \mathrm{E}(\beta, P, t)\}$
$\wedge$	Action conjunction	$\mathrm{E}(\alpha \wedge \beta, P, s) := \{\langle Q_\alpha^+ \cup Q_\beta^+, Q_\alpha^- \cup Q_\beta^- \rangle \mid \langle Q_\alpha^+, Q_\alpha^- \rangle \in \mathrm{E}(\alpha, P, s), \langle Q_\beta^+, Q_\beta^- \rangle \in \mathrm{E}(\beta, P, s), (s \cup Q_\alpha^+) \setminus Q_\alpha^- = (s \cup Q_\beta^+) \setminus Q_\beta^-\}$
$\neg_{\min}$	Action negation	$\mathrm{E}(\neg_{\min} \alpha, P, s) := \{\langle s' \setminus s, s \setminus s' \rangle \mid s' \notin \alpha(s)\}$

F_X	Syntactic frame	$\mathrm{E}^{\mathrm{x}}(\mathsf{F}_X(\alpha),P,s) := \{\langle e^+,e^-,i^+\setminus X,i^-\setminus X\rangle \mid \langle e^+,e^-,i^+,i^-\rangle \in \mathrm{E}^{\mathrm{x}}(\alpha,P,s)\}$
$s' \prec^s_{X,V,F} s''$	Preference w.r.t. to X,V,F	$s'\cap F = s''\cap F$, $(s'\Delta s)\cap X \subsetneq (s''\Delta s)\cap X$
$\mathsf{C}_{X,V,F}$	Circumscription	$\mathsf{C}_{X,V,F}(\alpha)(s) := \{s' \in \alpha(s) \mid \nexists s'' \in \alpha(s) : s'' \prec^s_{X,V,F} s'\}$
$a_{\mid t}$	t-conditioning	$\forall s \subseteq P\colon a_{\mid t}(s) = \{s' \mid (s'\cup t) \in \alpha(s\cup t)\}$
	Action Languages	
$L^{\mathbf{T}}$	Tree representation of language L	
$L^{\mathbf{C}}$	Circuit representation of language L	
NNFAT	NNF action theories	$\alpha ::= p \mid \neg p \mid p' \mid \neg p' \mid \alpha\wedge\alpha \mid \alpha\vee\alpha$
O-PDDL	"original nondet. **PDDL**"	$\alpha ::= \bot \mid +p \mid -p \mid \varphi \rhd \alpha \mid \alpha\cup\alpha \mid \alpha \,\&\, \alpha$
E-PDDL	"egalitarian nondet. **PDDL**"	$\alpha ::= +p \mid -p \mid \varphi \rhd \alpha \mid \alpha\cup\alpha \mid \alpha\sqcap\alpha$
O-NCSTRIPS	"orig. nondet. STRIPS"	$\&_{i=0}^{n}\left(\varphi_i \rhd \left((\ell_i^{1,1}\,\&\ldots\&\,\ell_i^{1,j_1})\cup\ldots\cup(\ell_i^{k_i,1}\,\&\ldots\&\,\ell_i^{k_i,j_{k_i}})\right)\right)$
E-NCSTRIPS	"egalit. nondet. STRIPS"	$\sqcap_{i=0}^{n}\left(\varphi_i \rhd \left((\ell_i^{1,1}\sqcap\ldots\sqcap\ell_i^{1,j_1})\cup\ldots\cup(\ell_i^{k_i,1}\sqcap\ldots\sqcap\ell_i^{k_i,j_{k_i}})\right)\right)$
NPDDL$_{\mathbf{nf}}$	Negation-free NPDDL	$\alpha ::= \bot \mid +p \mid \varphi \rhd \alpha \mid \alpha\cup\alpha \mid \alpha\sqcap\alpha$
E-PDDL$_{\text{seq}}$	**E-PDDL** with sequence	$\alpha ::= +p \mid -p \mid \varphi \rhd \alpha \mid \alpha\cup\alpha \mid \alpha\sqcap\alpha \mid \alpha\,;\alpha$
E-PDDL$_{\text{not}}$	**E-PDDL** with negation	$\alpha ::= +p \mid -p \mid \varphi \rhd \alpha \mid \alpha\cup\alpha \mid \alpha\sqcap\alpha \mid \neg_{\min}\alpha$
E-PDDL$_{\text{and}}$	**E-PDDL** with conjunction	$\alpha ::= +p \mid -p \mid \varphi \rhd \alpha \mid \alpha\cup\alpha \mid \alpha\sqcap\alpha \mid \alpha\wedge\alpha$
O-PDDL$_{\text{seq}}$	**O-PDDL** with sequence	$\alpha ::= +p \mid -p \mid \varphi \rhd \alpha \mid \alpha\cup\alpha \mid \alpha\,\&\,\alpha \mid \alpha\,;\alpha$
O-PDDL$_{\text{not}}$	**O-PDDL** with negation	$\alpha ::= +p \mid -p \mid \varphi \rhd \alpha \mid \alpha\cup\alpha \mid \alpha\,\&\,\alpha \mid \neg_{\min}\alpha$
O-PDDL$_{\text{and}}$	**O-PDDL** with conjunction	$\alpha ::= +p \mid -p \mid \varphi \rhd \alpha \mid \alpha\cup\alpha \mid \alpha\,\&\,\alpha \mid \alpha\wedge\alpha$
NNFAT$_{\mathsf{F}}$	**NNFAT** with synt. frame	$\alpha ::= p \mid \neg p \mid p' \mid \neg p' \mid \alpha\wedge\alpha \mid \alpha\vee\alpha \mid \mathsf{F}_X(\alpha)$
NNFAT$_{\mathsf{C}}$	**NNFAT** with circumscript.	$\alpha ::= p \mid \neg p \mid p' \mid \neg p' \mid \alpha\wedge\alpha \mid \alpha\vee\alpha \mid \mathsf{C}_{X,V,F}(\alpha)$
	Queries and Transformations	
IS-SUCC	Successorship	for given α,s,s': is $s' \in \alpha(s)$?
IS-APPLIC	Applicability	for given α,s: is $\alpha(s) \neq \emptyset$?
ENTAILS	Entailment	for given $\alpha_1,\ldots,\alpha_k,s,\varphi$: is $s' \models \varphi$ for all $s' \in (\alpha_1\,;\ldots;\alpha_k)(s)$?
ST	Self-transition	for given α,s: is $s \in \alpha(s)$?
IS-DET	Determinism	for given α,s: is $\lvert\alpha(s)\rvert \leq 1$?
IS-MON	Pos. (Neg.) Monotony	for given α,s: is $s' \supseteq s$ (for neg.: $s' \subseteq s$) for all $s' \in \alpha(s)$?
CHOICE	for given α_1,α_2 compute β s.t. for all s: $\beta(s) = \alpha_1(s)\cup\alpha_2(s)$	
SEQUENCE	for given α_1,α_2 compute β s.t. for all s: $\beta(s) = \{s' \mid \exists t \text{ s.t. } t \in \alpha_1(s) \text{ and } s' \in \alpha_2(t)\}$	
NEGATION	for given α compute β s.t. for all s: $\beta(s) = \{s' \mid s' \notin \alpha(s)\}$	
EXTRACT-PRECOND	for given α compute φ s.t. for all s: $\alpha(s) \neq \emptyset \iff s \models \varphi$	
	Complexity Classes	
P/poly, NP/poly, coNP/poly	Non-uniform complexity classes (see Section 3.3, Definition 3.8)	

Compilation de connaissances pour les langages d'actions non déterministes

Dans cette thèse, nous étudions les langages formels pour représenter des actions non déterministes du point de vue de la compilation de connaissances. Nous donnons un cadre formel pour capturer les caractéristiques essentielles de langages d'actions comme le « Planning Domain Definition Language » PDDL dans le cadre propositionnel, et présentons des langages qui sont des versions abstraites non déterministes des langages d'action de la littérature, ou leurs modifications obtenues en étendant ou en restreignant les ensembles de descriptions d'actions possibles. Les extensions sont obtenues en ajoutant des connecteurs supplémentaires à la grammaire, tandis que les restrictions sont obtenues en n'autorisant que les descriptions d'actions d'une certaine structure ou en supprimant des connecteurs de la grammaire. Nous comparons ces langages par rapport à trois critères : la complexité de répondre aux requêtes, la possibilité d'effectuer des transformations sans explosion en taille superpolynomiale, et la concision. Nous donnons une image complète pour les complexités de toutes les requêtes pour tous les langages, et une image presque complète pour les transformations et les requêtes pour les langages qui ne sont pas très riches mais toujours capables d'exprimer toutes les actions. Nos résultats concordent avec l'intuition que généralement il existe un compromis entre la concision et la complexité des requêtes. Nous identifions tout de même plusieurs cas où il pourrait être utile de préférer une représentation plutôt qu'une autre.

Mots-clefs : planification, compilation de connaissances, langage d'action, non déterminisme

Knowledge Compilation for Nondeterministic Action Languages

In this thesis we study formal languages for representing nondeterministic actions from the perspective of Knowledge Compilation. We define a formal framework to capture essential features of languages like the "Planning Domain Definition Language" PDDL in a propositional setting, and present a number of languages which are either abstract nondeterministic versions of action languages from the literature or their modifications obtained by extending or restricting the sets of possible action descriptions. Extensions are obtained by allowing additional connectives in the grammar, whereas restrictions are obtained by either allowing only expressions of a certain structure or removing constructs from the grammar. We compare those languages with respect to three criteria: complexity of answering queries, possibility of performing transformations without a superpolynomial explosion in size, and relative succinctness. We give a complete picture for complexities of all queries for all languages, and an almost complete picture for transformations and queries for languages which are not very rich but still fully expressive. Our results agree with the intuition that there is a tradeoff between the succinctness and the complexity of queries. We still identify several cases where it might be useful to prefer one representation over another.

Keywords: automated planning, knowledge compilation, action language, nondeterminism