Hoare-like verification of graph transformation

Jon Haël Brenas

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Mathématiques et informatique**

Arrêté ministérial : 7 août 2006

Présentée par

Jon Haël Brenas

Thèse dirigée par **Rachid Echahed**

préparée au sein **Laboratoire d'Informatique de Grenoble**
et de **Ecole Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Hoare-like Verification of Graph Transformation

Thèse soutenue publiquement le **13/10/2016**,
devant le jury composé de :

M. Jean-Guillaume Dumas
Université Grenoble-Alpes, Président
Mme. Pascale Le Gall
Ecole Centrale Paris, Rapporteur
M. Fernando Orejas
Universitat Politècnica de Catalunya, Rapporteur
M. Martin Strecker
Université Paul Sabatier, Examinateur
M. Rachid Echahed
CNRS & Université Grenoble-Alpes, Directeur de thèse

Contents

List of Figures

Acknowledgments

The work presented here is the result of these last three years. There have been difficult and tiring moments but there have been at least as many that were great memories. And there are a lot of people that I have to thank for them.

My first batch of thanks go to Ms. Bernard who rented me the flat in which I have lived during all my PHD. I arrived on the eve of my first day of work at the flat and she was surprised when I said that I was taking it without actually doing more than entering it. All along, she has been very nice, helpful and friendly. I want to include with her the flatmates I got during these years. A lot of them, I saw for only a few weeks and I don't really came to know, a few I spent very good time with and will not forget, all I thank for their part in my life.

I would then like to thank a long list of people that made my life outside of the thesis more thrilling and helped keep my balance. I want especially to thank Gregory Avondo, all his family and the RHC Lyon for the blast I had for the two years I played with and for them. I also want to thank the HC Voiron, and even more profoundly and particularly Claude Goudy, Fabien Deprat and Vincent Ducandas, for allowing me to join them and compete once more. In particular, I want to thank all the coaches and players I have been playing with and that helped me win a few more medals in my probable last year in the sport I gave my first quarter-century.

On a more work-related note, I want to thank all the people I worked with, from Martin who was instrumental in all the results this work presents, to Murielle and Zilora without whom I would never have been able to navigate the administrative waters, to Thierry, Mehdi and Mnacho whom I didn't really work with but where always close by for a nice word or a heart-warming "Hi".

Obviously, I want to thank my family. It is thanks to their help, efforts and support that I made it through. They have always believed in me, almost as much as I did in myself, even sometimes more, no matter what and they have always made it clear that I could aim for the stars and make it. And that if I missed they would always be there to catch me before I fell. I have been far from perfect and there have been a lot of hiccups on the road but they have always had my back. I have sometimes struggled to rise to the challenges and reach their expectations they have never stopped trusting me to rise higher with the next try. I know that I have chosen a path that leads me far from them, and it is probably only the beginning, but it is the fact that I love them and

that I know that they will be here waiting for me when I return that made all my achievements possible.

Last but far from least, I want to thank Rachid. Around three years ago, during an August afternoon, I entered his office for the first time. I was coming back from Berkeley and wanted to go to CERN. I had missed an opportunity to go there and had to keep myself busy until I could try again. Doing a PHD seemed as good an occupation as any for three years. I had no actual background in graph rewriting, no experience with verification and the only logic course that I had really had was on a booklet that had destroyed itself when I had tried to read it. Yet, he made me be interested in a subject I had never thought about. More than that, he made me love what I was doing and reconsider all my plans. It is not my place to judge the success of this work but I can proudly say that insomuch as it yielded results they are all due to the fires that Rachid kindled in my mind that made me dream night after night of a job that was not at all my dream job.

Résumé

En informatique comme dans de multiples autres domaines, les graphes peuvent être trouvés partout. Ils sont utilisés pour représenter des données dans des domaines allant de la chimie à l'architecture, en tant que structures abstraites ou que modèles des données et de leurs évolutions. Un graphe est défini comme l'union d'un ensemble de noeuds et d'un ensemble d'arcs les reliants. Ces noeuds comme ces arcs peuvent être étiquetés afin d'ajouter des informations supplémentaires.

Dans tous les domaines, il est prévisible que les graphes évoluent au cours du temps que cela soit à la suite de réactions chimiques, d'une mise à jour des connaissance ou de l'exécution d'un programme. Être capable de traiter ces transformations est une tâche particulièrement importante et difficile. Il y a de multiples manières de représenter les transformations de graphes. Parmi celles-ci, on peut trouver l'utilisation de la théorie des catégories ou l'utilisation d'actions atomiques. C'est cette seconde approche que nous avons choisie.

Dans ce travail, notre objectif est d'étudier la vérification de telles transformations de graphes, c'est à dire comment prouver qu'une transformation de graphes est correcte. La correction d'une transformation est plus précisément définie comme la correction d'une spécification pour cette transformation contenant en plus de la transformation elle-même une précondition et une postcondition. La précondition comme la postcondition sont des formules logiques définissant les états initiaux et finaux de l'exécution de la transformation. Prouver que la spécification est correcte revient à montrer que depuis tout graphe satisfaisant la précondition, il n'est possible d'obtenir en effectuant la transformation que des graphes satisfaisant la postcondition.

Le premier chapitre a pour vocation d'introduire de manière informelle les problèmes que nous souhaitons traiter. Il commence par expliquer pourquoi la vérification de programme est un sujet qui, malgré le fait qu'il est souvent ignoré, est particulièrement important dans le domaine de l'informatique et de ses applications. Il introduit ensuite ce qu'est un graphe et montre que l'utilisation des graphes se fait dans toutes sortes de domaines. De la même manière que les graphes sont présents un peu partout, nombre d'applications reposent sur la modification des graphes et qu'il est donc censé de s'intéresser à la correction des transformations de graphes. Ce chapitre s'achève par un plan de ce travail expliquant le contenu et la succession des chapitres.

Chapitre 2 montre que la vérification des transformations de graphe est un

sujet qui intéresse beaucoup de monde. Ce chapitre fait un état des lieux de la recherche dans ce domaine. En particulier, plusieurs outils existent qui utilisent la vérification de modèles pour prouver que des transformations sont correctes. La vérification de modèles est une approche différente de celle que nous avons choisi. Ce chapitre traite aussi de plusieurs logiques ayant été utilisées pour décrire les graphes et qui peuvent donc être considérées comme intéressantes pour notre système de vérification. Ce chapitre observe aussi les résultats obtenus par d'autres auteurs dans leurs recherche de résultats en terme de vérification de transformations de graphes. Enfin, le problème que nous considérons est comparé à d'autres problèmes qui ont été étudiés dans la littérature.

Dans un premier temps, dans le Chapitre 3, nous introduisons la définition formelle de ce qu'est un graphe étiqueté en utilisant des formules d'une logique. Ensuite, les transformations atomiques sont introduites. Ces actions permettent d'effectuer ce que nous estimons être les composants les plus élémentaires des transformations. Elles permettent de changer l'étiquetage d'un noeud ou plusieurs noeuds, de changer l'étiquetage d'un ou plusieurs arcs, de créer ou de détruire un noeud ou un arc ou encore de rediriger tous les arcs entrant ou sortant d'un noeud. Par exemple, l'action C := C + i, en supposant que C est un prédicat atomique unaire et que i est un noeud, change l'étiquetage de i pour y rajouter l'étiquette C alors que l'action i $\gg^{in}$ j, en supposant que i et j sont des noeuds, redirige tous les arcs entrants de i vers j. Ce chapitre introduit aussi l'exemple de l'hôpital qui est utilisé comme fil rouge tout au long de ce travail pour illustrer les résultats obtenus et les difficultés rencontrées.

Chapitre 4 introduit la première approche que nous utilisons pour modifier les graphes à savoir un langage de programmation impératif. Ce langage contient plusieurs éléments usuels des langages impératifs comme les boucles "while" et les expressions conditionnelles. Ces deux expressions utilisent à nouveaux la logique pour définir des conditions ce qui prouvent une fois de plus que les trois composants, graphes, transformations et logique, sont étroitement liés. En plus de ces expressions, le langage de programmaton contient des instructions pour les actions atomiques qui permettent de modifier le graphe. Enfin, une expression "select" est introduite qui permet de choisir non-déterministiquement un noeud parmi ceux satisfaisant une certaine condition auquel appliquer les transformations. Ce "select" est particulièrement important car il permet d'instancier les variables qui permettent de modifier les graphes. Afin de pouvoir utiliser ce language, nous avons implémenté un traducteur qui réécrit les programmes vers le language Java qui peut ensuite être utilisé.

Le chapitre suivant, Chapitre 5, est le chapitre central de ce travail. Il est le premier à discuter la logique et son utilisation dans la vérification des transformations de graphes. Le chapitre commence par une discussion sur l'utilisation d'une logique pour décrire un graphe nonobstant le fait qu'il soit modifié pour l'instant. Sa conclusion est que nous nous concentrons seulement sur les logiques dont les modèles sont considérés comme étant des graphes car ce sont celles qui sont adaptées au problème. Le chapitre enchaine avec la définition de l'algorithme permettant de prouver qu'une spécification est correcte. Nous avons choisi d'utiliser un calcul à la Hoare qui est particulièrement adapté à notre ap-

proche algorithmique car il repose sur la notion de pas d'exécution. L'idée est de générer la plus faible précondition correspondant à une transformation et une postcondition. Cette plus faible précondition est la formule qu'un graphe doit satisfaire pour être certain que tous graphe généré par la transformation satisfera la postcondition. S'il est possible de prouver que la précondition implique cette plus faible précondition, alors nous pouvons être certain que la spécification est correcte. Le calcul de la plus faible précondition se fait instruction par instruction en remontant depuis la postcondition. Il est donc nécessaire de définir la plus faible précondition des action atomiques. Cette plus faible précondition est définie en utilisant des substitutions. Les substitutions sont des constructeurs ajoutés à la logique dont la signification est exactement que la formule $\phi\sigma$, pour ϕ une formule et σ la substitution associée à une action atomique a, est satisfaite par un graphe si et seulement si tout graphe obtenu en exécutant a satisfait ϕ. Un autre résultat de l'utilisation du calcul à la Hoare est que les boucles "while" doivent être étiquetées avec un invariant car il est nécessaire lors de la génération de la preuve. Ceci est usuel dans le cadre des calculs à la Hoare. Enfin, le calcul doit être capable de quantifier sur les éléments qui sont instanciés par les instructions "select". De ces observations, on peut déduire les conditions que la logique doit satisfaire pour être capable d'utiliser de manière complètement automatisée le calcul à la Hoare. La logique doit être fermée par substitution, c'est à dire qu'elle doit être capable d'exprimer les substitutions sans l'ajout de nouveaux constructeurs. Elle doit aussi être capable d'exprimer l'utilisation des variables qui permettent d'effectuer les transformations. Enfin, la logique doit être décidable afin que l'automatisation soit complète. Le chapitre examine certains constructeurs de diverses logiques qui leur permettent de satisfaire ces conditions.

Chapitre 6 introduit la première famille de logiques que nous utilisons pour décrire les graphes, les logiques de description. Ces logiques ont l'avantage d'être décidables et d'être utilisées dans des cas réels. Le chapitre commence par introduire la syntaxe de ces logiques et les divers constructeurs qui permettent de former ces logiques. Une fois que cette syntaxe a été définie, nous examinons ces logiques pour voir lesquelles sont fermées par substitutions. Nous étudions d'abord les logiques les plus expressives, c'est à dire celles qui contiennent à la fois les nominaux qui permettent de parler individuellement de chaque noeud, et un constructeur permettant d'accéder à n'importe quel noeud comme le rôle universel U ou le quantificateur @. Ces logiques sont fermées par substitutions. Cela est prouvé en utilisant un système de réécriture qui retire les substitutions des formules tout en conservant les interprétations. Nous prouvons aussi que ce système termine. Il cause par contre une explosion exponentielle de la taille de formules qui accroit encore la complexité du raisonnement. Nous nous intéressons ensuite aux logiques qui ne contiennent pas les nominaux. Ces logiques ne sont pas fermées par substitutions. Cela est prouvé en utilisant des bisimulations entre modèles. Il est possible de définir une famille de bisimulation pour chaque logique tel que des noeuds appartenant à des modèles différents sont bisimilaires si et seulement si ils sont d'accord sur tous concepts de la logique. Il est possible de trouver deux modèles bisimilaires pour chacune de ces logiques

tels que l'un est un modèle d'une formule avec substitution et pas le second. La formule avec substitution ne peut donc pas être exprimée dans la logique. Enfin, nous nous intéressons aux logiques qui ont les nominaux mais qui n'ont pas de moyen d'accéder aux noeuds distants. La méthode utilisée précédemment doit être modifiée car les modèles bisimilaires s'accordent sur les formules avec substitution. Toutefois, en modifiant la définition des bisimulations, il est possible de trouver des modèles bisimilaires qui ne s'accordent pas sur les formules avec substitutions. Il est donc possible de prouver que ces logiques ne sont pas fermées par substitutions.

Le chapitre suivant, Chapitre 8, nous permet d'introduire un autre moyen de modifier les graphes. Nous utilisons des systèmes de règles dont le côté gauche est un graphe et le côté droit est une liste d'actions qui doivent être exécutées. Une règle ne peut être appliquée à un graphe que s'il existe un sous-graphe qui correspond au côté droit de la règle. Dans un tel cas, c'est ce sous-graphe qui est modifié par la règle. Afin de mieux contrôler l'exécution de ces règles, des stratégies sont définies. Elles permettent de dire quelle règle doit être appliquée quand ainsi que d'utiliser la fermeture d'une règle, c'est à dire l'utiliser tant que possible. Une nouvelle fois, un calcul à la Hoare est introduit pour prouver la correction de ce type de transformation. Ce calcul à la Hoare ressemble beaucoup à celui obtenu pour le langage de programmation impératif défini précédemment. Ainsi, il est aussi nécessaire de définir des invariants pour les fermetures. Les substitutions sont une nouvelle fois utilisée et cette condition doit donc toujours être satisfaite par les logiques. Enfin, ce calcul permet de rendre plus explicite la condition sur l'existence de variables auxquelles appliquer la transformation. Il est en fait nécessaire de pouvoir exprimer l'existence ou l'absence d'un sous-graphe correspondant aux côté droit d'une règle.

Chapitre 7 introduit une nouvelle logique $\mathcal{C}2\mathcal{PDL}$. $\mathcal{C}2\mathcal{PDL}$ est une logique qui contient les constructeurs de la logique dynamique propositionnelle avec inverse et de la logique dynamique combinatoire. Elle permet d'exprimer deux choses qu'il était impossible d'exprimer en utilisant les logiques de description: l'existence d'un chemin entre deux noeuds et de réellement créer et détruire des noeuds. L'existence, ou l'absence, d'un chemin est exprimée en utilisant la fermeture reflexive transitive d'un role. Comme le logique permet aussi d'exprimer des roles plus complexes, comme l'inverse d'un rôle ou l'union de deux rôles, il est possible d'exprimer l'existence de chemins complexes. D'un autre côté, les logiques utilisées jusque là utilisaient un concept particulier pour marquer les noeuds existants. Plutôt que de faire cela, une classe de noeuds particuliers est utilisée qui définit les noeuds qui n'existent pas encore ou plus. Comme cela fait partie de la définition de la logique, il n'est pas nécessaire de rajouter dans les préconditions et postconditions que les noeuds qui n'existent pas ne peuvent pas être étiquetés et ne peuvent pas avoir d'arcs entrants ou sortants ce qui donne des formules plus simples. Nous prouvons, une fois de plus en utilisant un système de réécriture, que $\mathcal{C}2\mathcal{PDL}$ est fermée par substitutions. Nous prouvons aussi que $\mathcal{C}2\mathcal{PDL}$ est décidable. Cela est fait de deux manières différentes. Un système de déduction est introduit qui permet de générer toutes les formules valides. Dans le même temps, il est prouvé que toute formule satisfiable a un

modèle de taille finie. Il est donc possible de tester si une formule est valide ou insatisfiable et donc de conclure. Une autre méthode utilise une traduction vers le mu-calcul hybride qui est connu comme étant décidable. Enfin, ce chapitre s'intéresse à la condition sur l'existence ou l'absence d'un sous-graphe. Il est prouvé qu'il est toujours possible, en utilisant des nominaux, d'exprimer l'existence du sous-graphe. Il est par contre nécessaire de restreindre la classe des graphes utilisés du côté droit des règles si l'on souhaite pouvoir exprimer l'absence d'un tel sous-graphe.

Chapitre 9 étend l'étude des logique qui peuvent être utilisées pour exprimer les conditions sur les graphes. Nous nous intéressons d'abord à la logique du premier ordre elle-même malgré le fait qu'il soit bien connu que le problème de décision pour la logique du premier ordre est indécidable. En utilisant une fois de plus un système de réécriture, il est possible de prouver que la logique du premier ordre est fermée par substitutions. Il est aussi facile de prouver que, comme la logique du premier ordre permet d'utiliser des variables librement, elle permet d'exprimer l'existence et l'absence d'un sous-graphe correspondant aux côté droit d'une règle. Comme cette logique est indécidable, nous nous intéressons par la suite à des fragments de la logique du premier ordre. Le premier qui est utilisé est le fragment à deux variables avec quantificateur numériques. Cette logique étend les logiques de description que nous utilisons. En utilisant le même système de réécriture que celui utilisé pour la logique du premier ordre, il est possible de prouver que cette logique est fermée par substitutions. Il existe par contre des sous-graphes dont l'absence est impossible à exprimer comme les graphes comprenant des cycles de longueur plus grande que trois. Nous étudions ensuite le fragment de la logique du premier ordre tel que les formules écrites en forme normale prenex contiennent tous les quantificateurs existentiels suivis de tous les quantificateurs universaux. Une fois de plus cette logique est fermée par substitution en utilisant le même système de réécriture. Toutefois, il existe aussi des sous-graphes dont l'absence ne peut pas être exprimée dans ce fragment. Par exemple, dire qu'il n'existe pas de noeuds n'ayant pas de voisin revient à dire que tous les noeuds ont un voisin ce qui ne peux pas être exprimé sans avoir le mauvais ordre parmi les quantificateurs.

Chapitre 10 introduit l'implantation que nous avons réalisé. Bien qu'elle ne soit pas fermée par substitutions, la logique $\mathcal{ALCQ}$ a été choisie comme logique pour cette implantation. En utilisant le prouveur de théorème Isabelle, nous avons défini la syntaxe et la sémantique de cette logique, ce qui nous a permis de prouver l'équivalence entre formules qui sont censées être logiquement équivalentes. Nous avons aussi défini la sémantique des substitutions afin de prouver que le système utilisé pour les logiques de description les plus expressive était correct quand il était utilisé sur $\mathcal{ALCQ}$. Ensuite, le calcul à la Hoare a été défini. Il a été prouvé qu'il est correct, c'est à dire que les spécifications qui sont considérées comme correctes le sont effectivement. Tout cela permet de prouver que la théorie que nous avons mise au point est cohérente. Nous avons aussi développé une méthode de tableaux pour $\mathcal{ALCQ}$ avec substitutions qui permet de prouver qu'une formule est satisfiable. Cet algorithme est utilisé pour prouver que la négation de la formule de correction générée par le calcul

à la Hoare est insatisfiable. Dans le cas contraire, un contre-exemple est généré qui peut être affiché sous forme de graphe afin de mieux comprendre quelle partie de la spécification est incorrecte.

Enfin, nous concluons sur notre travail. En plus de résumer les résultats obtenus, cette conclusion ouvre de nouvelles voies en terme de travail future que cela soit en s'intéressant à l'ajout d'opérations sur les données dans les actions et la logique, l'utilisation de logique moins expressive mais plus efficace en restreignant les actions possibles ou de logique plus expressives mais indécidables avec des actions plus complexes ou l'implantation d'algorithme pour les autres logiques et de traduction vers des prouveurs existants pour certaines des logiques considérées.

Chapter 1

Introduction

1.1 Program verification

Program verification is the field of study of the correctness of programs. To be more precise, it is the research of methods to prove that a program behaves the way it is expected to: it outputs what it should, it halts when it is done executing and no error occurs that forces it to abort some operations. Program verification is far from being a new field of study.

During World War II, Von Neumann was asked by Goldstine to collaborate to and improve ENIAC. It was the first computer using electrical signals instead of the mechanical motion of Babbage's computer. It was used to compute firing tables telling how to direct guns to hit distant targets. During that time and after when they had both joined the Institute for Advanced Study in Princeton, they wrote several reports that where the foundation of modern computer science. Among their conclusions was that "Since coding is not a static process of translation, but rather the technique of providing a dynamic background to control the automatic evolution of a meaning, it has to be viewed as a logical problem and one that represents a new branch of formal logics." [33]

Alas this vision did not receive much attention at the time and was slowly forgotten. As Donald Knuth said "People would write code and make test runs, then find bugs and make patches, then find more bugs and make more patches, and so on until not being able to discover any further errors, yet always living in dread for fear that a new case would turn up on the next day and lead to a new type of failure." [46]

An incentive for the development of formal methods to prove the correctness of programs was the growing number of such programs that failed in an untimely, and often very disastrous, manner. Most such errors are completely human-made, in that the programs behaved exactly as it should have, given the parameters. For instance, in 1998, the Mars Climate Orbiter built by NASA's Jet Propulsion Laboratory, having cost approximately $327.6 million, was destroyed when it collided with Mars instead of starting to orbit it because different

engineering teams where using different measuring units, namely the metric and imperial systems. Each sub-program was correct but they were not able to work together. This kind of program is difficult to be solved by formal methods and requires a clearly stated project road-map and clear typing and definitions, but there are multiple errors that are caused by a lack of ability to foresee the future use of a program. For instance, let's assume that we want to write a program that computes the mean value of a vector of integers. The most natural way would look like that in pseudo-code:

```
algorithm mean is
    input: A vector V of integers
    output: The mean value of the elements of V

    sum ← 0
    count ← 0

    while V is not empty do
        count ← count + 1
        sum ← sum + head(V)
        V ← tail(V)

    return sum/count
```

This program is plagued with problems the most obvious being that if V is an empty vector the program will just crash. Going beyond that, in programming languages with weak typing as C, a character can be considered as an integer in which case performing the mean value does not even make sense, the mean is unlikely to give the expected result on most inputs as the division operation between integers in many programming languages is considered to be the euclidean division, depending on the input, the values sum and even count may create integer overflows, ... All these sources of problems are to be taken into account in order to make sure that a program is correct.

In the following, we drop most of the sources of problems that are found in this example. We focus on proving that a program does what it is expected to do. To be more precise, we prove that specifications are correct. A specification contains three elements. One is the program itself. The other two are logic formulae called respectively pre- and post-conditions. The precondition describes the state of the input before the program is started while the postcondition describes the output. Both are used to add more constraints on the data that have to be checked. There are two different notions of correctness: partial and total correctness. A specification is said to be partially correct if on all inputs satisfying the precondition, that is all inputs such that the precondition is true, applying the program will only yield states which satisfy the postcondition, that is outputs such that the postcondition is true. One could see the first caveat: if the program does not halt on the input, there is no output and thus the spec-

ification is correct. To go one step farther, a specification is said to be totally correct if, in addition to the specification being partially correct, the program always halts.

The notion of total correctness is much more satisfying but it is, alas, very complex to prove and a subject of profound research on its own. It is a widely known fact that the halting problem is undecidable[68], that is there is no algorithm that always decides whether a program will halt. As we are interested in automatic proofs that a specification is correct, we have thus to either additionally provide an external proof that the program stops or only focus on partial correctness. This is the choice that is commonly made and that we will make in this work.

There are several main ways to prove a specification correct among them model-checking[20] and Hoare logic[40]. Models are representations of the data, both as input and output and as it is modified during the execution of the program. Model-checking works by defining a transition system corresponding to the transformation and finding a characterization of the models that generate unwanted states or the models that can be generated from the initial state.

On the other hand, Hoare logic works with transformation steps. It starts from a condition and generates the weakest precondition of the statements of the program. The set of preconditions of a statement s and a postcondition Q, written $Pre(s, Q)$, is the set of the formulae such that if it is satisfied before applying s, then it has to be that Q is satisfied after applying S. The weakest precondition $wp(s, Q)$ is the smallest element of $Pre(s, Q)$ in the sense of the implication, that is $wp(s, Q) \in Pre(s, Q)$ and if $P \in Pre(s, Q)$, $P \Rightarrow wp(s, Q)$. That is every model that satisfies P also satisfies $wp(s, Q)$. This is applied until the first statement of the program is reached and one then has to check that the precondition implies the weakest precondition of the whole program. It is possible to use strongest postconditions instead of weakest preconditions and work from top to bottom. In the following, we will be exclusively working with Hoare-like logics to prove the correctness of specifications. We may nonetheless refer to other works that would use model-checking as part of alternatives.

1.2 Graphs

Up to now, we have introduced the notion of program verification. We are not actually interested in program verification in all generality but in the more specific verification of graph transformations. Yet, before being able to study the correctness of a program modifying graphs, it is mandatory to define what a graph is.

The term "graph" was first coined by J.J. Sylvester[64]. To put it simply, a graph is a set of elements, called nodes, that are linked by edges. They can take many different forms depending on what meaning one associates with each components. Figure 1.1 represents a social network where each node is a person and edges represent connections between these persons. Edges and nodes can be labeled, that is they can contain information. In the example of the social

Figure 1.1: An example graph representing a social network.

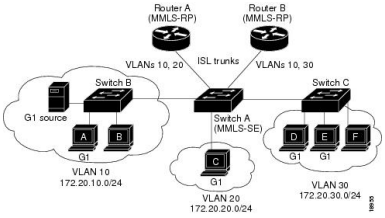


Figure 1.2: An example graph representing a computer network.

network, each node being a person, it makes sense that they are labeled with the name of that person. Edges can be labeled with the kind of connection that corresponds to the link (friend on Facebook, follower on Twitter, ...).

One of the main interests of graphs is that they are ubiquitous. They are one of the most standard ways to represent data. In addition to social networks, graphs are key modeling tools in domains as diverse as communication, where nodes can be servers and computers and edges the physical connections between them (one such example is shown in Figure 1.2), public transportation networks, where nodes are stations and edges are routes (for instance, Figure 1.3), chemistry, where nodes are atoms and edges are covalent bonds (as examplified in Figure 1.4), in genealogy to build family trees, in relational databases, ...

They can also be used to represent more abstract structures. In addition to graphs themselves, trees and pointers, and thus also arrays and such, are also data structures that can be represented as graphs. As they are, by nature, used in programs, this proves that the underlying problem that we are studying is of great interest.

Although they can represent all sorts of different data, as far as computer science is concerned, this is far from being the only use of graphs. They are also among the most widespread ways to do meta-modelling, that is modeling of the models of the graphs. Most current standards for metamodeling, like ORM[70], ER[19] or UML[43] for instance, use graphs to represent the different classes of objects and the relation between them.

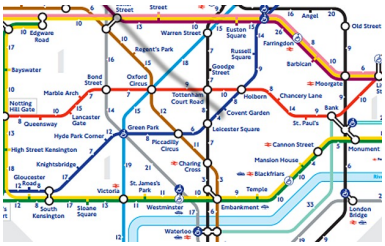


Figure 1.3: A sample of the map of the London subway network

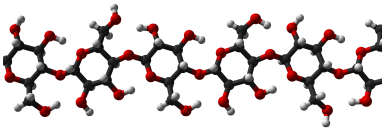


Figure 1.4: Part of a 3D representation of a cellulose molecule

Another widespread use of graphs in computer science is to represent programs themselves. Automata, as shown in Figure 1.5, and Petri nets, illustrated in Figure 1.6, are only two possible uses of graphs to model programs.

The informal definition of a graph is quite simple yet it still can represent all sorts of data. That is why it is key to be able to reason about it, to modify it and to prove that the modifications that are performed are correct. In the following, we use a slightly more elaborate notion of graph. Up to now the labels were purely descriptive: if a node is labeled with "Alice", we know that it is the name of the person associated with this node. Yet, we want to be able to express more interesting properties. To do so, instead of using lists of possible labels, we use logics. The definition of logics covers a wide array of topics and is an important field in philosophy, mathematics, computer science, linguistics and psychology among others. The main focus of logic is the systematic study of arguments. It includes, for instance, how to correctly infer a conclusion from a

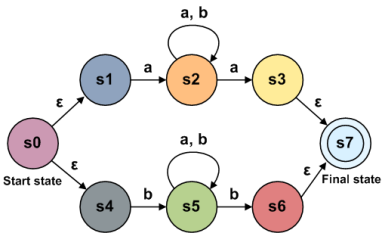


Figure 1.5: A simple automaton

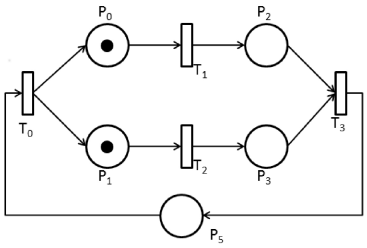


Figure 1.6: A simple Petri net

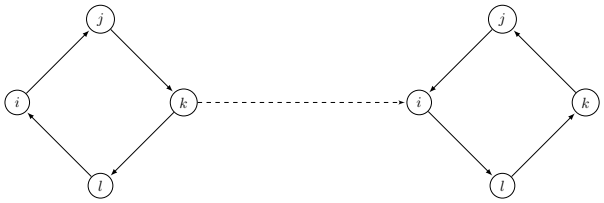


Figure 1.7: An example of graph transformation that inverts all edges in a 4-cycle.

set of assumptions. Nonetheless, for us, a logic can be understood as a language that allows to build complex labels from the previous set of atomic labels. It also come with a semantic, that is a model theory, that allows to state whether a property is true or not.

Labeling a graph with logical formulae is quite usual in Kripke structures where each node is thought to represent a possible world that contains all information that is true in this world. In this work, labeling formulae will play a role both in the transformation process and in the generation of proof obligations. Indeed, transformations will be performed depending on conditions that will be expressed in the logic while the properties intended to be proven and the weakest preconditions that computed are also formulae.

1.3 Graph transformation

Our goal is to do graph transformation verification and it is thus natural to now introduce the notion of graph transformation. This is quite a straightforward idea: one is given a graph and wants to modify it for whatever reason. Figure 1.7 shows one such dummy transformation where one wants to invert the direction of all edges.

There are several ways to describe graph transformations. One of the easiest to understand is the one used in Figure 1.7 where a graph is given as input and the resulting graph is given as output. This is a most graphic representation of

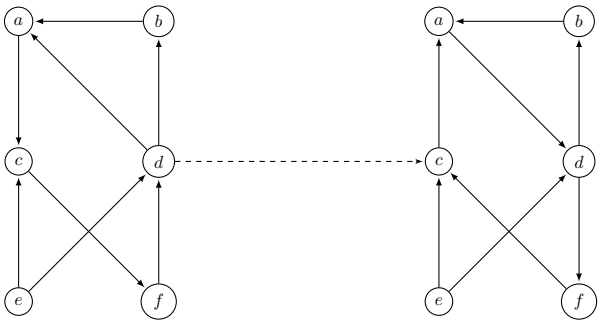


Figure 1.8: An example of application of the graph transformation of Figure 1.7.

graph transformations. Obviously, the graph that one wants to modify is not always the one that appears in such a transformation. The usual way to deal with this is to match the pattern represented by the left-hand side, say L, in the actual graph, say G. To be more precise, one looks for a sub-graph, that is a graph whose set of nodes is a subset of the set of nodes of G and accordingly for the edges, that would coincide with L. This instance of L is then replaced with the right-hand side R in G to create a new graph G'. Figure 1.8 shows such an example where the subgraph containing only the nodes a, c, f and d and the edges $(a, c), (c, f), (f, d)$ and (d, a) can be equated to the left-hand side of Figure 1.7. In the resulting graph, these 4 edges are inverted but all other edges are kept the same.

Such an understanding of graph transformations is best exemplified by the category theory approach. This approach is the most common way to deal with graph transformations. Transformations are performed by sets of rules that are a little more involved than the one shown in Figure 1.7. In addition to the pattern graph and the resulting graph, a core graph is used. These rules are classified depending on the conditions that they impose upon the transformations in term of suppression of elements.

Another approach, that is the one that we have chosen, is different. It provides a more algorithmic touch to the problem making it closer to classical programming languages. Instead of using a resulting graph, it introduces a list of actions that are performed on the matched graph. These transformations can then take the form of either rule sets[14] or imperative programs[15] depending on the needs of the user.

1.4 Statement of contribution

Now that we have made clearer what we mean by verification of graph transformations, it is key to define what is our goal and how we plan to reach it. Our goal is not, as has been done in previous works, to present a combination of a graph transformation system and a logic but to provide a way to pick one or

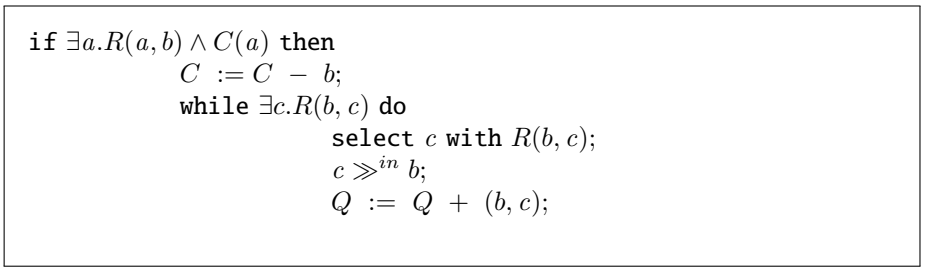


Figure 1.9: An example of transformation written as a program

several logics that can be used to prove graph transformations correct. To be more precise, we want to avoid putting the burden of devising the verification process to the user. To reach that goal, we want the logic that is chosen to express the pre- and post-conditions to be able to express everything that we need for the verification procedure so that existing decision procedures for that logic can be used to prove the transformations correct.

In order to reach this goal, we start by giving an imperative language that contains statements specifically tailored for graph transformations. This language is the one presented in [15]. To be more precise, the language introduced in Chapter 4 contains the atomic actions defined in Section 3.1 plus the usual statements of imperative languages as if-then-else statements and while loops. It also contains a non-deterministic select statement that allows to bind variables that are used in the transformations. A translation from this language to Java has been implemented to increase its usefulness and a proof system, that outputs a counter-model if the program is incorrect has also been written for one of the logics treated in Chapter 6. Such a program is given as example in Figure 1.9. This programs checks whether there exists an R-predecessor of b labeled with C in which case b is no longer labeled with C. Then, as long as b has an outgoing edge labeled R, all incoming edges of B are redirected toward the target of that edge and it is labeled with Q. However, the proof system actually does not completely meet our requirements as it uses a tableau-procedure specifically devised to be able to work with the logic that we use in that logic.

The following chapter, Chapter 5, is the centerpoint of the theory. In this chapter, the relation between the graph transformation systems and the logics is investigated more thoroughly. The previous definition of the imperative language allows us to have a concrete example of how transformations are defined and performed and is used as the basis for the reflexion. Nonetheless, we try to make as few assumptions as possible on the logic that is used, as we want to actually find out what is required from the logic, and on the transformations themselves so that the reasoning that we do can be applied even to other graph transformations. This chapter is mostly an extended version of [12].

First, the Hoare-like calculus that actually performs the verification of the graph transformations is introduced. Let's assume we are interested in proving

that for all graphs G such that a precondition P is true, any graph G' obtained by performing transformation t is such that the postcondition Q is true. The Hoare-like calculus defines the weakest precondition of t and Q, written $wp(t, Q)$. This weakest-precondition is defined by induction on the structure of the transformation. Its most atomic step takes an atomic action a and returns as its weakest precondition $wp(a, Q) = Q[a]$. This $[a]$ is called a substitution and is defined such that $Q[a]$ is the formula that is true if and only if Q is true after performing a. The Hoare-like calculus we present is proven to be sound, that is only programs that are actually correct can be proven to be so.

The Hoare-like calculus is the starting point of the study of the properties the logic should have. Several such properties are pointed out. Some are mostly relative to the fact that the logic is actually suited to speak about graphs in the first place, that is that it is able to speak about the labeling of nodes and edges and be such that its models are graphs. The other properties deal more with the transformation themselves. Each atomic transformation, as is usual in Hoare-like calculi, generates a substitution that becomes part of the formula. They are forwarded through the formulae during the computation of the weakest preconditions and the verification conditions. These substitutions need to be handled during the decision procedure for the logic. Hence, the first requirement for the logic is that it is closed under substitutions, that is that substitutions can be removed inside the logic thus allowing one to use conventional tools to test whether the correctness condition is verified or not. Secondly, transformation systems need to deal with the selection of the nodes where the transformation is performed. Being able to treat these selections also has to be something that the logic can handle so that the verification calculus can work. Satisfy these conditions is a strong requirement on the expressivity of the chosen logic and there is not one logical constructor that fits the needs of all logics. It is thus key to study what to look for in a logic so that it can meet these requirements. For each of these conditions, intuitive components of logics that allow them to be verified are provided.

The conditions introduced in Chapter 5 can appear particularly vague as they depend enormously on the chosen actions and the way they are used. In order to clarify things and give examples, Chapter 6 instantiates this framework. To be more precise, this chapter essentially focuses on the condition of closure under substitutions[13]. After introducing quickly some usual Description Logics[5], namely extensions of $\mathcal{ALC}$ without role inclusion axioms, we study which of those are closed under substitutions and why. In order to also better illustrate the way the transformations and the verification procedure work, this chapter contains an instance of the running example introduced in Section 3. The conclusion is that the logic needs to provide some way to speak about the actual nodes that are modified individually, that is in the case of Description Logic they need to allow for nominals, and a way to access remote nodes, which can be done with a universal role or the @ constructor of hybrid logics.

In order to better illustrate the problem of proving that a logic can express that a rule can be applied to a graph, we introduce a modal logic $\mathcal{C}2\mathcal{PDL}$ that

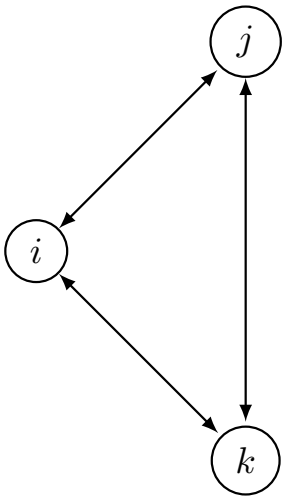


Figure 1.10: An example graph for which the existence of a match cannot be expressed in $\mathcal{C}^2$ as it requires to take into account 3 different node variables.

extends multi-modal logics with nominals, that allow us to speak about specific nodes, and a universal role on sets of nodes that allows us to handle the creation of new nodes and their deletion. In Chapter 7, this logic is formally defined. This logic benefits us in several different ways. First, its expressivity is quite different from the one of the Description Logics studied up to that point. In particular, it allows to use a closure operator on roles that can express reachability conditions that are particularly interesting in the conditions of rules. Second, it is easy to prove that it is closed under substitutions and it can be done with much less effort than in the case of Description Logics. Finally, it presents an interesting handling of the problem of expressing the existence of a match. It cannot express the existence of a match in all generality. Yet it is possible to express the existence of a match. This is not enough to be able to prove all specifications written in $\mathcal{C}2\mathcal{PDL}$. This is enough to prove that a problem is correct as long as loops are not used. Indeed, in the case of loops, one has to express that it is no longer possible to apply the rule, and thus that no match exists. We identify conditions on the rules appearing in loops so that it is possible to express that no match exists and thus restrict the set of programs for which we can effectively prove that it is correct. We also prove that $\mathcal{C}2\mathcal{PDL}$ is decidable.

The next chapter focuses on the second key condition that is that the logic should allow to express the selection of the nodes that are modified. As we felt that the exact cause for this condition was not made extremely clear by the imperative program and as we also wanted to be able to use a transformation framework closer to the rewriting systems that are used in many different incarnations of graph transformations systems, we introduced in Chapter 8 such a rewriting system[14]. As usual, the transformations in that system are handled

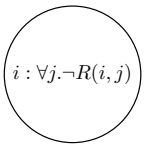


Figure 1.11: An example graph for which the absence of a match cannot be expressed in $\exists^*\forall^*$ as it would require the formula to be $\forall i.\exists j.R(i,j)$.

via rules whose left-hand side are graphs. We departed from the usual definition though by allowing the nodes and edges of those graphs to be labeled with arbitrary formulae which increases the expressivity of the rules themselves. In order to more efficiently direct the execution of the transformations, strategies are also introduced. This allows us to be as expressive as the imperative programming language defined in Chapter 5. With this system formally defined, the condition on the select can be made much clearer. So that a rule can be applied, a match between its left hand-side and the graph that is to be modified has to be found. The condition becomes that the existence of such a match can be expressed as a formula of the logic.

In Chapter 9, we look at other, more mainstream, logics that someone could propose that would fit our requirements. The first proposal is first-order logic. It is closed under substitutions and allows to express the existence of a match. Yet it is well-established that first-order logic is undecidable and this is thus a very small step in the right direction. We thus study fragments of first-order logic that are known to be decidable. Namely we focus on $\mathcal{C}^2$[34], the 2-variable fragment of first-order logic with counting. It is a fragment that has been proven to be decidable, both in the case of finite and unrestricted models. It is also known to extend Description Logics without role assertions or inclusions. It is, unsurprisingly, proven to be closed under substitution. It also allows to express the existence of a match but, once more, one has to restrict the set of rules that are allowed. Indeed, if the graph in Figure 1.10 was the left-hand side of a rule, finding a match would require to keep track of three different variables, i, j and k which is not possible in $\mathcal{C}^2$. Also, this logic does not allow to express the reachability properties. Another logic that we study after is $\exists^*\forall^*$[16]. Once again, validity of a formula of this logic is known to be decidable and it is easy to show that it is closed under substitutions. Yet there are rules where the impossibility of finding a match cannot be expressed in the logic. A rule whose left-hand side would be the graph of Figure 1.11 being an example of such case. This chapter presents results from [12].

Finally, in Chapter 11, we conclude on this work, discuss the points that are left open and introduce extensions that we are interested in.

Dealing with graph transformations is not simple and we want to make things as clear as possible. Toward that goal, we will start by giving a quick overview of the current state of the art in the field of verification of graph transformations in Chapter 2. Furthermore, in order for everything to be easier to understand,

in the next chapter, Chapter 3, we introduce the running example of a hospital. It will be the basis of most of the examples that will be used in the course of this work. It does not aim at being the perfect representation of what should be the model of a hospital but at giving an example that any reader has at least a basic understanding of while remaining sufficiently complex to be able to tackle the various problems that we unravel in this work. Before doing that, we introduce the formal definitions of a graph, of which actions we consider and of the problem we are aiming to solve.

Chapter 2

State of the art

The correctness of model transformations has attracted some attention in the last few years. One prominent approach is model checking, such as implemented by the Groove tool [31]. The idea is to carry out a symbolic exploration of the state space, starting from a given model, in order to find out whether certain invariants are maintained or certain states (*i.e.* , model configurations) are reachable. Transformations are represented as color-coded graphs and allow creation and deletion of nodes and edges. The rules created that way are then combined using control programs that contain random choice between rules and loops. As the Groove tool generates the state space, it is important to determine how this state space is explored as it can greatly influence the efficiency of the computation. The example of the Transformation 1 of Section 3.3 is given in Figure 2.1.

The Viatra tool has similar model checking capabilities [69] and in addition allows the verification of elaborate well-formedness constraints imposed on models [62]. Well-formedness is within the realm of our approach as it amounts to checking the consistency of a formula, but it is not our primary goal in this work. Indeed, all we require is a specification and we do not care whether the pre- and postconditions are well-formedness conditions, structural invariants, strong restrictions on the input, ...

The Alloy analyser [45] uses bounded model checking for exploring relational designs and transformations (see for example [9] for an application in graph transformations). Counter-examples are presented in graphical form. All the aforementioned techniques use powerful SAT- or SMT-solvers, but do not carry

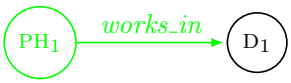


Figure 2.1: An example of graph used in GROOVE representing Transformation 1.

out a complete deductive verification. On the other hand, our main interest lies in this automatic verification and its study. Our approach is thus, though using a different kind of computation, more decisive in its conclusion as it can not yield an inconclusive result. Nonetheless, contrary to our approach up to now, it is actually working, efficient and widely used. When we happen to have developed tools with comparable support, it will be interesting to compare the efficiency of the various approaches.

Speaking of existing tools and automated provers, general-purpose program verification with systems such as AutoProof [67] and Dafny [48] becomes increasingly automated and thus interesting as push-button technology for model transformations. In this context, fragments of first-order logic have been proposed that are decidable and are useful for dealing with pointer structures [44]. They focus on linked-lists though while we are interested in graphs in general. Our goal is thus more general and it could be interesting, once we are able to produce a working tool, to compare results. One of the biggest problems though is that expressing that a graph is a finite linked-list is outside of the expressibility of the decidable logics we introduced as it requires to state that there is a head and a tail such that the head has no predecessor, the tail has no sucessor, every other element has exactly one predecessor and one sucessor and that there exists a path from head to tail. It would thus require to find a logic that would be compatible with the problem under scrutiny.

In this work, we aim at using a Hoare-like approach to program verification. We are interested in the devising of a Hoare-like calculus for the verification of graph transformations. The usual approach is to pick one logic that is able to express the graph properties and that can be used in the verification. We are thus really interested in the various logics that could be used to express graph properties.

Several different logics have been proposed over the years to describe graphs and they are natural choices when one has to tackle the problem of graph transformation verification. One of the most natural choices is first-order logic. Another widely used logic in graph transformation verification is monadic second-order logic [21, 59, 42] that allows to go beyond first-order definable properties. Nonetheless, these approaches are not flawless. They are both undecidable in general and thus either cannot be used to prove properties of graphs in an automated way or only work on limited classes of graphs. We will nonetheless consider first-order logic as it is an interesting logic that is widely used and whose undecidability is often dealt with by accepting that the prover returns an inconclusive result.

Starting from the other side of the logical spectrum, one could consider using Description Logics to describe graph properties [1, 13] that are decidable. Description Logics cover a very wide array of complexities. For instance, satisfiability of $DL - Lite$ is polynomial which is really important when dealing with graphs of huge size that are common in a lot of applications. On the other hand, the same problem is NExpTime-hard for $\mathcal{SROIQ}$. Another choice could be to use a modal logic as those logics are made to reason about programs [10]. Actually, $\mathcal{ALC}$ is equivalent to modal logic. The main difference is that

modal logic traditionally uses a possible-world semantic. Temporal logics [56] and Hybrid Logics [4] and other extensions of modal logics can also be used.

Another natural choice is to focus on decidable fragments of first-order logic. The fragment of effectively propositional logic [55], has been known for a long time to be decidable [16]. Despite the fact that it contains all Description Logics without complex role inclusions, the use of the two-variable logic [34] for the verification of model transformation is, to the best of our knowledge, introduced in this work.

Separation logic [60] is another logic that is worth considering when dealing with transformations of graphs. It has been developed especially to be able to talk about pointers in conventional programming languages. It has been applied to program verification [52, 3] in the case of such programing languages as *C* minor [49] that is a mid-level imperative programming language.

Obviously, the decidability of all these logics comes at a cost in terms of expressiveness compared to undecidable logics like the full first-order logic. The question is thus to determine what expressivity is required by any given problem and the answer to this question is thus the main indicator of which logic should be used. Even though not all of the logics that have been considered here are actually treated in the following, many are and it should be possible to work on any specific logic in order to check whether the conditions that we pinpoint for a logic to work with our proof computation are satisfied.

In this work, we proceed in an orthogonal direction. Instead of introducing a logic and advising users to tailor their problem so that it is expressible in our logic and that its models comply with the restrictions so that the verification is actually possible, we aim at providing a means for the users to decide whether the logic they have used to represent their problem will actually allow them to prove their transformations correct or whether they have to use several different systems in parallel.

Naturally, introducing a logic to express graph properties is not enough to be able to verify graph transformations. Numerous different approaches have been proposed to deal with this verification. A lot of these approaches are based on the most usual definition of graph transformations that is rooted in category theory. These approaches often introduce logics that are specially tailored for the problem under study. Among the most prominent figure GP[58, 50], that stands for Graph Programs. It uses nested conditions [37, 57] that are explicitly created to describe graph properties. Graph Programs are strategies applied to a set of rewriting rules whose right-hand sides and left-hand sides are graphs. A Hoare-like calculus is provided that allows to generate correct specifications for the graph program. It uses transformations on the rules to generate conditions for the applicability of a rule and the precondition for a rule and a postcondition. The main shortcoming of this approach is that the Hoare-like calculus is undecidable which means that the verification cannot be fully automated. On the other hand, Graph Programs allow for the use of data labels and of operations on them. This is something that we can not do as of today but that it would very interesting to further study as many actual algorithm, as for instance the computation of the shortest path between two nodes, that

modify graphs do so depending on the arithmetical properties of the labels. An extension of nested conditions using paths[53] has been proposed to give more expressive descriptions. The satisfiability of a formula has been proven sound and complete. It has not been devised in the context of verification yet to the best of our knowledge.

If one wants to compare our results to theirs, two possibilities arise. The first idea could be to look at whether it would be possible to use nested conditions as one of the logics that we consider. Nested conditions do not allow for nominals but they can be simulated as it is possible to say that there is a node with a given label and that it is the only one. Nested conditions also contain existential quantifiers that can be used for the selection of the nodes where the atomic actions are performed. It would thus seem that nested conditions satisfy the requirements we put forth. However, nested conditions are actually not closed under substitutions as they do not allow for disjunctions of graph morphisms in the first part of the conditions.

Another interesting subject of comparison is on the expressive power of the transformation. The fact is that allowing for arithmetical conditions greatly increases the transformation power and that it is not possible to express any of these transformations using our kinds of transformations. On the other hand, if one allows for arithmetical conditions on labels in one of our systems, the two transformation frameworks become comparable once more. Comparing the algorithmic and categorial approaches is not so easy yet if there were transformations that were possible only in the categorial approach, it would be possible to define an algorithmic atomic action with the same effect and one would then has to check that the logics are still closed under this added substitution. The set of atomic actions is not closed for that reason. Some of these actions have actually be studied but have not yielded enough results yet to be included in this work. For instance, cloning of nodes is interesting and can be performed in some categorial approaches. An atomic action with the same effect can be introduced and it is then possible to check which logics are closed under substitutions.

On the other hand, [8] introduces a logic closer to modal logic extended with an universal program and global and local assignments. This logic allows to express both the graph properties and the graph transformations at the same time. As the transformations are expressible in the logic as programs, it is possible to verify the transformation simply by proving that a formula of the logic is valid. This is thus not more complex than proving that a graph property is valid. Nonetheless, the logic is undecidable and it thus impossible to completely automate the verification. Trying to use this logic in any of our transformation systems does not make much sense as the logic is expressive enough to contain all transformation.

In [47], a slightly different problem is tackled. Instead of proving that a specification is correct, they aim at proving that, from a start graph, it is not possible to reach an element of a set of forbidden graphs. They thus use methods that are closer to model-checking. These forbidden graphs are defined using context-free graph grammars. Rewriting systems are then modeled as Petri graphs and the algorithm checks that there is no reachable marking of this graph

that would mean that a forbidden graph can be reached. Checking whether no marking in a set is reachable is decidable albeit complex. This can be improved using restrictions on the starting graph and the structures of the rules. This differs from our case as, in addition to using rules that are structured differently with a graph as right-hand side, it forces the user to provide a starting graph while we would like to just describe it using a precondition. It also lacks a strategy for using the rules that we are interested in both as actual strategies and as an imperative program.

The way one tackles verification of graph transformation is, obviously, quite dependent on what are the properties that one wants to express and how the transformations are performed. In [1], the possible transformations that can be performed are different from the ones we use. The transformations focus on global transformations in that they only allow to modify labels of sets of nodes instead of individuals. The logic used is $\mathcal{ALCHOIQ}br$, a Description Logic that contains, among others, nominals which means that it is possible to modify single node labeling. The transformations are performed as imperative programs without loops. A lot of loops can actually be replaced with class relabeling but loops can still increase the expressiveness of the transformation language. As $\mathcal{ALCHOIQ}br$ is included in $\mathcal{C}^2$, it can be used in combination of one of the systems we introduced yet with the same limitations that apply to the more expressive logic.

Another problem that seems to be really similar is knowledge base update [2, 28, 23] in which one tries to know whether some knowledge base will be satisfied from a given knowledge base after performing some actions that update the knowledge. An update adds or removes (in which case, it is usually qualified as erasure) pieces of knowledge. The main difference with what we consider as actions is how much change we consider occurs. In update, if one knows that every *Bird* is able to *Fly* and learns that **Chilly Willy** does not *Fly* it concludes that **Chilly Willy** is not a *Bird*, that is it aims at changing as little as possible the knowledge base. On the other hand, if the same is considered as an action, no possible outcome can be excluded and thus one would consider that every *Bird*, except maybe **Chilly Willy**, can *Fly*. As the way changes are handled is different, these two problems are quite different. It could be interesting to move farther in that direction by combining the approach we have been using of generating the weakest precondition with a set of unalterable axioms that can be used to remove alternatives and thus making the precondition stronger while still being the weakest.

Among modal logics, one can single out epistemic logics[39]. Their objective is to represent knowledge. Models are graphs whose nodes represent the various states that are considered as possible and edges are equivalence relation between indistinguishable states. For instance, let us consider a two player game where *A*lice and *B*ob each draw a ball. Each one of them knows the ball she has picked and they both know that there are only three balls two of which are *R*ed, the last one being *W*hite. They don't know which ball the other player has picked though. *A*lice has picked the white ball and *B*ob has picked a red ball. The graph presented in Figure 2.2 is their knowledge graph. Each state

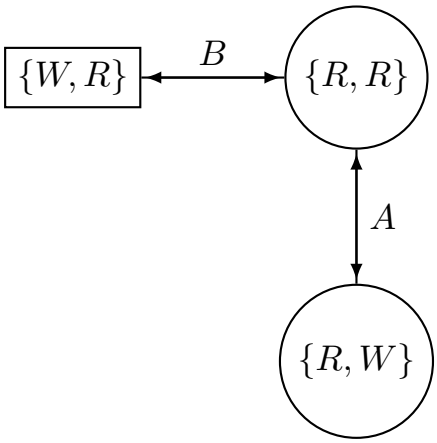


Figure 2.2: An example of epistemic model with indistinguishable states for *A*lice and *B*ob. The actual state is the rectangle.

is pair whose first element is the color of *A*lice's ball and whose second element is the color of *B*ob's. There are three possible states $(R, W), (R, R)$ and (W, R) and the actual state is (W, R). *A*lice knows that it is so as *B*ob can only have one of the two remaining balls both of which are *R*ed. On the other hand, *B*ob cannot distinguish between the states (W, R) and (R, R) as he doesn't know the color of *A*lice's ball. Were *A*lice holding a *R*ed ball, she would not be able to discriminate between the states (R, R) and (R, W) either.

Epistemic logics are particularly interesting in our case because it embedded in the conception of the logic that actions will be performed. To be more precise, it is possible to alter models by using public announcements[30]. Only something true in the actual state can be publicly announced and its effect is to remove from the graph all states which are inconsistent with the announcement. For instance, if *A*lice were to announce that she has the *W*hite ball, only (W, R) would be a possible state. If, instead of announcing that she had the *W*hite ball, *A*lice had announced that she knew the color of *B*ob's ball, the result would have been the same as if the actual state had been any of (R, R) or (R, W), she would not have been able to announce this.

Public announcements are complex actions that are more complex to emulate that one could think. Indeed, one could state that one just has to use a rule that would erase all nodes that do not satisfy the announcement. The problem is that it has to be simultaneous. Indeed, in the second case, if one were to remove the state (R, R) because it is not a case where *A*lice is unable to know the color of *B*ob's ball, she would be left with two states (W, R) and (R, W) both of which are states in which *A*lice knows the color of *B*ob's ball and thus (R, W) would not be erased. The solution is thus to mark all states that are to be erased in a first go and then erase all nodes that are marked and repeat until no node is marked.

It is possible to have more involved models too by giving various degrees of confidence in the various states and to have different actions as for instance forgetting information that ends up creating new possible states. It is also evident, as can be seen with the muddy children conundrum, that having a perfectly logical understanding of knowledge yields situations that are difficult to intuit and may not be perfectly realistic.

Another key problem that can be linked to program verification is Ontology-Based Data Access[26, 11], or OBDA. Given a knowledge base or database containing some data, one wants to know the results of a query. This is the most usual task in databases. The problem is that, when data becomes bigger, it is no longer efficient to store all the data inside the database. An ontology is thus used to reduce the size of the database without loosing actual knowledge.

For instance, the IMDB Movie Ontology[1] contains the database for the IMDB website that deals with movies. There are thousands of movies that are stored and belong to various categories. It is much more efficient to store in the database that $AM(M_0)$, ..., $AM(M_n)$, meaning that Movies M_0 to M_n are Action Movies and in the ontology that all Action Movies are Movies than to store in the database $M(M_0), ...M(M_n), AM(M_0), ..., AM(M_n)$.

OBDA may seem to have very little to do with program verification yet the two problems are closely linked. Indeed, the database can be considered as a special case of precondition, the query can be considered as a postcondition and the ontology can be treated as a program. Most ontologies are built such that every axiom as to be applied as much as possible. In our context, that means that strategies are usually unused. Yet, one could imagine ontologies that would require strategies. For instance, still in the context of the movie database, one could want to know before the Oscar Award Ceremony whether there may be, depending on who is awarded the Best Actress Oscar, a movie by a given director with 2 or more Best Actress Oscar winners. Adding to the ontology a set of rules that awards the Best Actress Oscar to one of the nominees and requiring that one and only one of the rules is applied could yield a strategized ontology that would lead to the correct answer.

[1] https://github.com/ontop/ontop/wiki/Example_MovieOntology

Chapter 3

Preliminaries and Running Example

In this chapter, our goal is to introduce several examples and definitions that we are going to use all around this work to illustrate our purpose.

First, we are going to give a few formal definitions that will form the underlying structures that will be used in all following chapters of this work. We define what is a graph as far as we are concerned, that is a pair composed of a set of nodes and a set of edges, with functions on the latter giving for each edge its source and its target and functions, for both nodes and edges, that give their labels. According to our definition, node and edge labels are formulae of a logic. We are not, for now, interested in what those formulae mean or how they are interpreted, we just use them to label things.

We then define a set of actions. They allow one to modify a graph by changing the labeling of a node ($C := C + i$, $C := C - i$) (respectively of an edge ($R := R + e$, $R := R - e$)), or a set of nodes ($C := \psi$, $C := i$) (respectively a set of edges, ($R := Q$, $R := e$)), to redirect incoming (respectively outgoing) edges from a node to another one ($i \gg^{in} j$) (respectively $i \gg^{out} j$), to create or suppress nodes and edges ($new\_node(i), del\_node(i), new\_edge(e, i, j), del\_edge(e)$). These atomic actions can then be combined into a sequence of actions.

The second part of this chapter introduces the example of a hospital. Our goal is to verify graph transformations and thus, for it to make sense, we need to be able to have an idea of what actual transformations and properties of the graphs can be used. It is a voluntarily simplistic example that is used only with the hope to be able to illustrate what is done in the various following chapters. It should deal with a subject any reader is familiar enough with to have an idea of what should be while being simple enough to be used in human-readable examples.

To go with the description of the hospital, the final part of this chapter introduces transformations of the hospital. Once more, they do not aim at

being all-encompassing but at being fairly easy to grasp and to use. Together with these transformations, several properties that make sense are introduced as part of an invariant for the transformations, that is as formulae that should always be satisfied after a transformation is performed granted that they were satisfied before.

3.1 Formal definitions

When one aims at dealing with graph transformations, it makes sense to first define formally what is considered a graph. We diverge slightly from the simplest definition in that our graphs contain nodes and edges that are labeled with formulae from a given logic. Apart from that, the definition is classical.

Definition 3.1.1 (Graph)**.** *Let $\mathcal{L}$ be a logic. A* graph alphabet *is a pair ($\mathcal{C}$, $\mathcal{R}$) of sets of elements of $\mathcal{L}$, that is $\mathcal{C} \subseteq \mathcal{L}$ and $\mathcal{R} \subseteq \mathcal{L}$. $\mathcal{C}$ is the set of* node formulae *(of $\mathcal{L}$) or* concepts *and $\mathcal{R}$ is the set of* edge formulae *(of $\mathcal{L}$) or* roles*. Subsets of $\mathcal{C}$ and $\mathcal{R}$, respectively named $\mathcal{C}_0$ and $\mathcal{R}_0$, contain basic concepts and roles respectively. A* graph *G over a graph alphabet ($\mathcal{C}$, $\mathcal{R}$) is a tuple (N, E, ϕ_N, ϕ_E, s, t) where N is a set of* nodes*, E is a set of* edges*, ϕ_N is the* node labeling *function, $\phi_N : N \to \mathcal{P}(\mathcal{C}_0)$, ϕ_E is the* edge labeling *function, $\phi_E : E \to \mathcal{P}(\mathcal{R}_0)$, s is the* source function *$s : E \to N$ and t is the* target function *$t : E \to N$.*

It is worth noting that the codomain of the labelings are defined as containing only basic concepts and roles. These definitions are extended so that the labeling is coherent with the semantics of the logic used. One can thus assume that elements are labeled with complex formulae. We call Φ_N (resp. Φ_E) this extended node labeling function (resp edge labeling function).

Definition 3.1.2. *Let $\mathcal{L}$ be a logic, G be a graph over $\{\mathcal{C}, \mathcal{R}\}$, $\mathcal{C} \cup \mathcal{R} \subseteq \mathcal{L}$, n a node of G, $\psi \in \mathcal{C}$ (resp. $Q \in \mathcal{R}$), we say that $G, n \models_{\mathcal{L}} \psi$ if $\psi \in \Phi_N^G(n)$ (resp. $Q \in \Phi_E^G(e)$).*

Once it is clear what a graph is, it is mandatory to define how to transform it. Our decision was to eschew category theory and head towards a more algorithmic approach to graph transformations. The following definition introduces several atomic actions that allow one to modify graphs. We first only give an idea of what each action does before giving the formal definition of its effects on a graph.

Definition 3.1.3 (Atomic action, action)**.** *An* atomic action*, say a, has one of the following forms:*

- *a* concept assignment *$C := \psi$ where C is a basic concept in $\mathcal{C}_0$ and ψ is a concept in $\mathcal{C}$. After performing $C := \psi$, a node n is labelled with C if and only if it is labeled with ψ.*
- *a* role assignment *$r := Q$ where r is a basic role in $\mathcal{R}_0$ and Q is a role in $\mathcal{R}$. After performing $r := Q$, an edge e is labelled with r if and only if it is labeled with Q.*

- *a* concept definition $C := i$ *where* C *is a basic concept in* $\mathcal{C}_0$ *and* i *is a node. After performing* $C := i$, i *is labelled with* C *and it is the only one.*

- *a* role definition $r := e$ *where* r *is a basic role in* $\mathcal{R}_0$ *and* e *is an edge. After performing* $r := e$, e *is labelled with* r *and it is the only one.*

- *a* concept addition $C := C + i$ *where* i *is a node and* C *is a basic concept in* $\mathcal{C}_0$. *It adds the node* i *to the valuation of the concept* C.

- *a* concept deletion $C := C - i$ *where* i *is a node and* C *is a basic concept in* $\mathcal{C}_0$. *It removes the node* i *from the valuation of the concept* C.

- *a* role addition $r := r + e$ *where* e *is an edge and* r *is a basic role (edge label) in* $\mathcal{R}_0$. *It adds the edge* e *to the valuation of the role* r.

- *a* role deletion $r := r - e$ *where* e *is an edge and* r *is a basic role (edge label) in* $\mathcal{R}_0$. *It removes the edge* e *from the valuation of the role* r.

- *a* node creation $new\_node(i)$ *where* i *is a new node. It creates the node* i. i *has no incoming nor outgoing edge and there is no basic concept (in* Φ_0*) such that* i *belongs to its valuation (resp. it deletes* i *and all its incoming and outgoing edges).*

- *a* node deletion $del\_node(i)$ *where* i *is an existing node. It deletes* i *and all its incoming and outgoing edges.*

- *an* edge creation $new\_edge(e, i, j)$ *where* e *is a new edge and* i *and* j *are existing nodes. It creates the edge* e *such that* $s(e) = i$ *and* $t(e) = j$. e *is not labeled with any basic role.*

- *an* edge deletion $del\_edge(e)$ *where* e *is an existing edge. It deletes* e.

- *a* global incoming edge redirection $i \gg^{in} j$ *where* i *and* j *are nodes. It redirects all incoming edges of* i *toward* j.

- *a* global outgoing edge redirection $i \gg^{out} j$ *where* i *and* j *are nodes. All outgoing edges of* i *becomes outgoing edges of* j.

The result of performing the atomic action α *on a graph* $G = (N^G, E^G, \phi_N^G, \phi_E^G, s^G, t^G)$, *written* $G[\alpha]$, *produces the graph* $G' = (N^{G'}, E^{G'}, \phi_N^{G'}, \phi_E^{G'}, s^{G'}, t^{G'})$. *An* action, *say* α, *is a sequence of atomic actions of the form* $\alpha = a_1; a_2; \ldots; a_n$. *The result of performing* α *on a graph* G *is written* $G[\alpha]$. $G[a; \alpha] = (G[a])[\alpha]$ *and* $G[\epsilon] = G$, ϵ *being the empty sequence.*

Let us illustrate the actual transformation associated to each atomic action α by giving the graph G' such that $G \Rightarrow_\alpha G'$.

- **If** $\alpha = C := \psi$ **then:**
  - $N^{G'} = N^G$

- $E^{G'} = E^G$
- $\phi_N^{G'}(n) = \begin{cases} \phi_N^G(n) \cup \{C\} & \textbf{if } n \models \psi \\ \phi_N^G(n) \backslash \{C\} & \textbf{otherwise} \end{cases}$
- $\phi_E^{G'} = \phi_E^G$
- $s^{G'} = s^G$
- $t^{G'} = t^G$

- **If** $\alpha = r := Q$ **then:**
  - $N^{G'} = N^G$
  - $E^{G'} = E^G$
  - $\phi_N^{G'} = \phi_N^G$
  - $\phi_E^{G'}(e) = \begin{cases} \phi_E^G(e) \cup \{r\} & \textbf{if } e \models Q \\ \phi_N^G(e) \backslash \{r\} & \textbf{otherwise} \end{cases}$
  - $s^{G'} = s^G$
  - $t^{G'} = t^G$

- **If** $\alpha = C := i$ **then:**
  - $N^{G'} = N^G$
  - $E^{G'} = E^G$
  - $\phi_N^{G'}(n) = \begin{cases} \phi_N^G(n) \cup \{C\} & \textbf{if } n = i \\ \phi_N^G(n) \backslash \{C\} & \textbf{otherwise} \end{cases}$
  - $\phi_E^{G'} = \phi_E^G$
  - $s^{G'} = s^G$
  - $t^{G'} = t^G$

- **If** $\alpha = r := e$ **then:**
  - $N^{G'} = N^G$
  - $E^{G'} = E^G$
  - $\phi_N^{G'} = \phi_N^G$
  - $\phi_E^{G'}(e') = \begin{cases} \phi_E^G(e') \cup \{r\} & \textbf{if } e' = e \\ \phi_N^G(e') \backslash \{r\} & \textbf{otherwise} \end{cases}$
  - $s^{G'} = s^G$
  - $t^{G'} = t^G$

- **If** $\alpha = C := C + i$ **then:**
  - $N^{G'} = N^G$
  - $E^{G'} = E^G$

- $\phi_N^{G'}(n) = \begin{cases} \phi_N^G(n) \cup \{C\} & \textbf{if } n = i \\ \phi_N^G(n) & \textbf{if } n \neq i \end{cases}$
- $\phi_E^{G'} = \phi_E^G$
- $s^{G'} = s^G$
- $t^{G'} = t^G$

- **If** $\alpha = C := C - i$ **then:**
  - $N^{G'} = N^G$
  - $E^{G'} = E^G$
  - $\phi_N^{G'}(n) = \begin{cases} \phi_N^G(n) \backslash \{C\} & \textbf{if } n = i \\ \phi_N^G(n) & \textbf{if } n \neq i \end{cases}$
  - $\phi_E^{G'} = \phi_E^G$
  - $s^{G'} = s^G$
  - $t^{G'} = t^G$

- **If** $\alpha = r := r + e$ **then:**
  - $N^{G'} = N^G$
  - $E^{G'} = E^G$
  - $\phi_N^{G'} = \phi_N^G$
  - $\phi_E^{G'}(e') = \begin{cases} \phi_E^G(e') \cup \{r\} & \textbf{if } e = e' \\ \phi_E^G(e') & \textbf{otherwise} \end{cases}$
  - $s^{G'} = s^G$
  - $t^{G'} = t^G$

- **If** $\alpha = r := r - e$ **then:**
  - $N^{G'} = N^G$
  - $E^{G'} = E^G$
  - $\phi_N^{G'} = \phi_N^G$
  - $\phi_E^{G'}(e') = \begin{cases} \phi_E^G(e') \backslash \{r\} & \textbf{if } e = e' \\ \phi_E^G(e') & \textbf{otherwise} \end{cases}$
  - $s^{G'} = s^G$
  - $t^{G'} = t^G$

- **If** $\alpha = new\_node(i)$ **then:**
  - $N^{G'} = N^G \cup \{i\}$ **where** i **is a new node**
  - $E^{G'} = E^G$
  - $\phi_N^{G'}(n') = \begin{cases} \emptyset & \textbf{if } n' = n \\ \phi_N^G(n') & \textbf{if } n' \neq n \end{cases}$

- $\phi_E^{G'} = \phi_E^G$
- $s^{G'} = s^G$
- $t^{G'} = t^G$

- **If** $\alpha = del\_node(i)$ **then:**
  - $N^{G'} = N^G \backslash \{i\}$
  - $E^{G'} = E^G \backslash \{e | s^G(e) = i \vee t^G(e) = i\}$
  - $\phi_N^{G'}$ **is the restriction of** ϕ_N^G **to** $N^{G'}$
  - $\phi_E^{G'}$ **is the restriction of** ϕ_E^G **to** $E^{G'}$
  - $s^{G'}$ **is the restriction of** s^G **to** $E^{G'}$
  - $t^{G'}$ **is the restriction of** t^G **to** $E^{G'}$

- **If** $\alpha = new\_edge(e, i, j)$ **then:**
  - $N^{G'} = N^G$
  - $E^{G'} = E^G \cup \{e\}$ **where** e **is a new edge**
  - $\phi_N^{G'} = \phi_N^G$
  - $\phi_E^{G'}(e') = \begin{cases} \emptyset & \textbf{if } e' = e \\ \phi_N^G(e') & \textbf{if } e' \neq e \end{cases}$
  - $s^{G'}(e') = \begin{cases} i & \textbf{if } e' = e \\ s^G(e') & \textbf{if } e' \neq e \end{cases}$
  - $t^{G'} = \begin{cases} j & \textbf{if } e' = e \\ t^G(e') & \textbf{if } e' \neq e \end{cases}$

- **If** $\alpha = del\_edge(e)$ **then:**
  - $N^{G'} = N^G$
  - $E^{G'} = E^G \backslash \{e\}$
  - $\phi_N^{G'} = \phi_N^G$
  - $\phi_E^{G'}$ **is the restriction of** ϕ_E^G **to** $E^{G'}$
  - $s^{G'}$ **is the restriction of** s^G **to** $E^{G'}$
  - $t^{G'}$ **is the restriction of** t^G **to** $E^{G'}$

- **If** $\alpha = i \gg^{in} j$ **then** :
  - $N^{G'} = N^G$
  - $E^{G'} = E^G$
  - $\phi_N^{G'} = \phi_N^G$
  - $\phi_E^{G'} = \phi_E^G$
  - $s^{G'} = s^G$

- $t^{G'}(e) = \begin{cases} j & \textbf{if } t^G(e) = i \\ t^G(e) & \textbf{if } t^G(e) \neq i \end{cases}$

- **If** $\alpha = i \gg^{out} j$ **then** :
  - $N^{G'} = N^G$
  - $E^{G'} = E^G$
  - $\phi_N^{G'} = \phi_N^G$
  - $\phi_E^{G'} = \phi_E^G$
  - $s^{G'}(e) = \begin{cases} j & \textbf{if } s^G(e) = i \\ s^G(e) & \textbf{if } s^G(e) \neq i \end{cases}$
  - $t^{G'} = t^G$

Just defining what are atomic graph transformations is quite clearly not enough to be able to construct actual graph transformations. How to build these transformations is already an interesting subject per se and, though it is not the sole focus of this work, it is still the main subject of Chapter 4 and Chapter 8.

Once graphs and their transformations have been defined, one can go back to the overall objective of graph transformation verification, that is to prove that a graph transformation is correct.

Let s be the graph transformation that one wants to perform and which one actually wants to be able to prove correct. As in the case of programs, one could think of several definitions of what is a correct transformation. One could say that it is a transformation that does something, or that it can not fail, or that it must always stop, etc. As proving that a program halts is a difficult and worthwhile subject on its own, we concentrated on proving that programs do what they have to provided they halt. Nonetheless, one wants to be more specific than saying that the program does someting. To be able to describe what one expects the program to do, we use conditions that describe wanted properties before and after the computation. We call the union of these conditions and the transformation a specification and we prove that these specifications are correct. It is thus worth noting that the proof of the correctness is at most as good as the specification.

Definition 3.1.4 (Specification)**.** *A* specification *S is a triple $(Pre, s, Post)$ where Pre and $Post$ are formulae and s is a graph transformation.*

A specification is composed of three elements. In addition to the transformation s, the specification contains Pre, the so-called *precondition*, and $Post$, the *postcondition*. Pre is a formula that describes the conditions that one wants to be satisfied before s is performed and $Post$ describes the conditions that have to be satisfied after s is performed.

Definition 3.1.5 (Correctness)**.** *A specification S is said to be* correct *if for all graphs G satisfying Pre, any graph G' that can be obtained from G by performing s is such that G' satisfies $Post$.*

The notions of specification and correctness thereof forces us to deal with conditions that have to be expressed in a language. Those languages are logics. Determining what are the characteristics we are looking for in logics is the main focus of this work that will be dealt with in Chapter 5. We then consider what these properties mean for the logics themselves and this leads to the study of several logics, namely Description Logics in Chapter 6, an extension of Dynamic Logic in Chapter 7 and several expressive fragments of First-Order Logic in Chapter 9.

3.2 Hospital example

In order to better illustrate our purpose, an example modelling a sample of the information system of a hospital is introduced. It is the same one that we used in [12]. Figure 3.1 is the UML model of this sample.

We consider persons (shortened to PE). Some of them work in the hospital and form the medical staff (MS) and others are patients (PA). The medical staff is partitioned into physicians (PH) and nurses (NU). In addition, the hospital is split into several departments (DE) or services. Files (FI) are used to store information. Those pertaining to patients are stored in folders (FO).

Each member of the medical staff is assigned (denoted by *works_in*) to a department. The same way, each patient is hospitalized (*hospital_in*) in one of the departments. There may be several members of the medical staff that may collaborate to treat (*treats*) a patient at a given time but one of them is considered as the referent physician (*ref_phys*), that is to say she is in charge of the patient. Part of the medical staff can access the folder containing the documents about (*is_about*) a patient either to read (*read_access*) or to write (*write_access*) of information. Files belong (*belongs_to*) to a folder and can reference (*reference*) other files. One of the files is HP, the hospital policy.

3.3 Transformations and properties

The fact is the hospital is bound to evolve: new patients arrive to be cured and others leave, new medical staffers are hired and others move out. To illustrate our purpose, four possible transformations are defined below.

Transformation 1. *The first transformation is New_Ph(*PH_1, D_1*). It creates a new physician to which is associated an identifier* PH_1*. This physician will be working in the department identified with* D_1*.*

Transformation 2. *The second transformation is New_Pa(*PA_1, PH_1, FO_1*). It adds a new patient. The patient* PA_1 *is created alongside his folder* FO_1*. He is then assigned* PH_1 *as referent physician.*

Transformation 3. *The third transformation is Del_Pa(*PA_1*). It removes patient* PA_1*.*

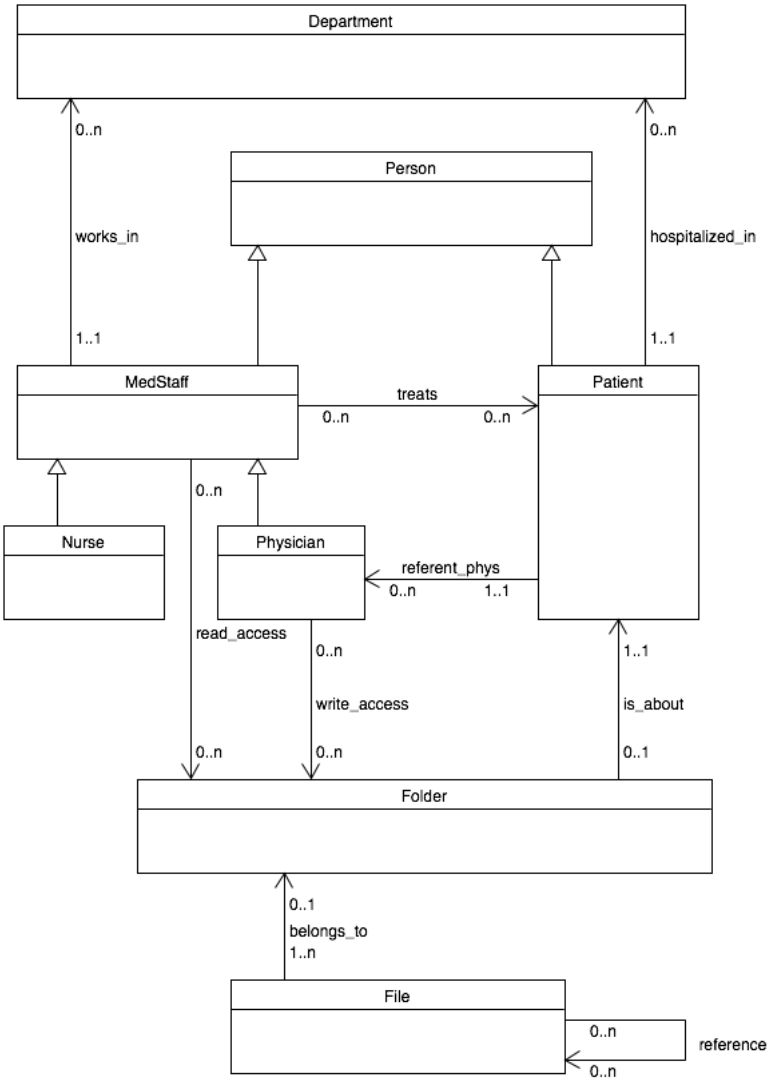


Figure 3.1: A sample UML model for the hospital example

Transformation 4. *The fourth transformation is Del_Ph(*PH_1, PH_2*). It deletes the physician* PH_1 *and forwards all his patients to the physician* PH_2. PH_1 *and* PH_2 *have to work in the same department.*

Despite the transformations, there are some properties of the hospital that should not be altered. We give a list of six such expected properties in the following.

Expected property 1. *Each member of the medical staff is either a nurse or a physician but not both.*

Expected property 2. *All patients and all medical staffers are persons.*

Expected property 3. *Each person that can write in a folder can also read it.*

Expected property 4. *Each person that can read a folder about a patient treats that patient.*

Expected property 5. *Only medical staffers can treat persons and only patients can be treated.*

Expected property 6. *Every patient has exactly one referent physician.*

Expected property 7. *Every file references, albeit maybe indirectly, the Hospital Policy file.*

3.4 Conclusion

In this chapter, we gave our formal definition of graphs and we introduced atomic transformations. These definitions are not completely usual. For instance, we decided to label edges and nodes in graphs with formulae in order to improve the expressive power of the transformations. Also we wanted a more algorithmic approach to transformation which lead us to less usual definitions of transformation. In addition, using sequences of atomic actions naturally induce the definition of execution steps that can be used as the basic block on which the Hoare-like calculus is going to work.

We also introduced a running example that we plan to use to illustrate the various results of this work. We chose the example of a hospital as it is fairly easy to understand. Additionally, it is a simplified and partial description so that, when it is used as an example, it is possible to produce both readable and yet sufficiently complex properties and transformations. We hope in this way to make clearer the most elaborate problems that we faced.

Chapter 4

An imperative language

As explained in Section 1.3, there are several ways to think about graph transformations. Using an imperative language may not be the most widespread way to deal with graph transformations but it is rather intuitive and a good way to introduce progressively the difficulties that arise when trying to prove the correctness of a program modifying graphs.

In this chapter, we introduce such an imperative language. It is the same as the one we showed in [15]. Its most simple element is an atomic action. The set of atomic actions is composed of those that have been defined in Section 1.3. They allow to modify graphs by modifying the labeling of nodes and edges. It is possible to change the label of one local element, be it a node or an edge, by adding or removing that element from those labeled by a given atomic label (by performing the action $c := c + i, c := c - i, r := r - e$ or $r := r + e$). It is also possible to modify globally the way nodes and edges are labelled by assigning a new set of elements to a label (by performing the action $c := D, r := Q, c := i$ or $r := e$). It is also possible to modify the graph without affecting the labeling by redirecting all incoming or outgoing edges of a node (by performing the action $i \gg^{in} j$ or $i \gg^{out} j$) or by adding or removing nodes and edges (by performing the action $new\_node(i), del\_node(i), new\_edge(e, i, j)$ or $del\_edge(e)$). These atomic actions are then composed together using usual constructs of imperative languages as if-then structures, while loops and sequencing of actions. A more original structure is the "select" statement that non-deterministically assigns an element satisfying a condition to a variable. These are used to handle the selection of specific nodes of the graphs where the transformations are requested to occur.

As the programming language contains conditional structures, namely the if-then structure, the while-loop and the "select" statement, a logic is needed. For this chapter, we do not worry about the logic. While next chapter, Chapter 5, is concerned with the topic of the logic and all the consequences on it that are implied by the needs of the transformation languages, this chapter only aims to introduce the imperative language and give details and examples of its workings.

Once the programming language is introduced in Section 4.1, this chapter presents some results that have been obtained by implementing a system using this programming language. To be more precise, an Eclipse plug-in has been developed to translate programs written in our imperative programming language into Java. Another program has been implemented that checks the correctness of the program and returns a counter-example when the program is incorrect. More details can be found in Section 4.2 on the implementations and in the following chapters for the proof of correctness of a program.

4.1 Example of imperative programing language for graph transformation

The programming language introduced in this chapter is an imperative language manipulating relational structures. Its distinctive features are conditions (in conditional statements and loops) that are formulae. It has a non-deterministic assignment statement `select ... with` allowing to select an element according to a formula. Traditional types (numbers, arrays, inductive types) and accompanying operations are not provided; the language thus really focuses on transformations of graphs.

Due to the "select" statement, variables are used in the transformation language. In order to deal with them, states are used instead of graphs as the structures that are modified.

Definition 4.1.1 (State)**.** *A* state *σ is a tuple $(G, \{v_1, \dots\}, A)$ where G is a graph, $\{v_1, \dots\}$ is a list of node variables and $A : \{v_1, \dots\} \to N$ is the assignment function that assigns a node to a variable.*

It is important to realize that during the program execution, nodes are accessed through these variables and the nodes that are modified are those pointed by the variables. That is when $C := C + i$ is performed, i is not a node and it is the node $A(i)$ that is modified.

Another key point is that edges are considered as pairs of nodes instead of separate entities. All possible pairs are considered as edges and those that are actually part of the graph are those that are labeled. This different view of edges, obviously, impacts the atomic actions that are used.

Definition 4.1.2. *Let C be a node label, R be an edge label, i, j be node variables, an* atomic action *is one of the following graph transformations:*

$C := C + i$	*(node labeling)*
$C := C - i$	*(node unlabeling)*
$R := R + (i, j)$	*(edge labeling)*
$R := R - (i, j)$	*(edge unlabeling)*

One can observe that not all atomic actions defined in Section 1.3 are reported in Definition 4.1.2. The creation and deletion of nodes is done in a different fashion. It is assumed that there exists a concept *Active*, unused in

the properties, that denotes nodes that exist. If this is the case, creating a new node amounts to labeling it with *Active* and deleting a node amounts to removing the *Active* label. Thus, these actions can be simulated using atomic actions as defined in Definition 4.1.2. An additional condition on node not labeled with *Active* is added that they cannot have any label and that they can not have any incoming or outgoing labeled edge. This implies that the sets of atomic node and edge labels are finite. This is obviously not an optimal translation as the formulae may become very cumbersome. The other actions are removed for the sake of simplicity for now.

Atomic actions are used as the building block of statements, that is of graph transformations.

Definition 4.1.3. Statements *of the language are then defined by the following grammar:*

stmt	::=	`a`	*(atomic action)*
	|	`select` i `with` $form$	*(assignment)*
	|	$stmt$ `;` $stmt$	*(sequence)*
	|	`if` $form$ `then` $stmt$ `else` $stmt$	*(if-then statement)*
	|	`while` $form$ `do` $stmt$	*(while loop)*

The semantics is a big-step semantics with rules of the form $\sigma \Rightarrow_{stmt} \sigma'$ expressing that executing statement *stmt* in state σ produces a new state σ'.

The rules of the semantics are given in the Figure 4.1. Beware that we overload logical symbols such as $\exists$, $\wedge$ and $\neg$ for use in the meta-syntax and as constructors of $form$.

Intuitively, the states σ manipulated by the operational semantics describe the current structure of a graph: which concepts label each node; which roles label each edge; and which variables are bound to which nodes. We write $\sigma(\phi)$ to evaluate the condition ϕ (a formula) in state σ.

The statement `select` i `with` ϕ selects an element vi that satisfies condition ϕ, and assigns it to i. i is a node variable. For instance, let us assume that *stmt* is the statement `select` i `with` $i : C$, that is select a node labeled C. i is not a node but, if there exists a node labeled C, one of them will be used as the value of i.

`select` is a generalization of a traditional assignment statement. There may be several instances that satisfy ϕ, and the expressiveness of the logic might not suffice to distinguish them. In this case, any such element is selected, non-deterministically. Let us spell out the precondition of (SelAssT): here, $\sigma^{[v:=vi]}$ is an interpretation update. What it does is modify the assignment function A such that $A(i) = vi$. It checks whether the formula ϕ would be satisfied under this choice, and if it is the case, keeps this assignment. In case no satisfying instance exists, the semantics blocks, *i.e.* the given state does not have a successor state, which can be considered as an error situation.

Now that loops have been introduced it is possible to simulate actions like $i \gg^{in}_{l_0} j$ that redirects the incoming edges of i labeled l_0 toward j as shown in Figure 4.2. Provided that there is a finite number of atomic edge labels, it

$$(\textsc{Seq})\ \frac{\sigma \Rightarrow_{c_1} \sigma'' \quad \sigma'' \Rightarrow_{c_2} \sigma'}{\sigma \Rightarrow_{c_1;c_2} \sigma'}$$

$$(\textsc{SelAssT})\ \frac{\exists vi.(\sigma' = \sigma^{[i:=vi]} \wedge \sigma'(\phi))}{\sigma \Rightarrow_{\texttt{select}\ i\ \texttt{with}\ \phi} \sigma'}$$

$$(\textsc{IfT})\ \frac{\sigma(\phi) \quad \sigma \Rightarrow_{c_1} \sigma'}{\sigma \Rightarrow_{\texttt{if}\ \phi\ \texttt{then}\ c_1\ \texttt{else}\ c_2} \sigma'} \qquad (\textsc{IfF})\ \frac{\neg\sigma(\phi) \quad \sigma \Rightarrow_{c_2} \sigma'}{\sigma \Rightarrow_{\texttt{if}\ \phi\ \texttt{then}\ c_1\ \texttt{else}\ c_2} \sigma'}$$

$$(\text{WT})\ \frac{\sigma(\phi) \quad \sigma \Rightarrow_{c_2} \sigma'' \quad \sigma'' \Rightarrow_{\texttt{while}\ \phi\ \texttt{do}\ c} \sigma'}{\sigma \Rightarrow_{\texttt{while}\ \phi\ \texttt{do}\ c} \sigma'} \qquad (\text{WF})\ \frac{\neg\sigma(\phi)}{\sigma \Rightarrow_{\texttt{while}\ \phi\ \texttt{do}\ c} \sigma}$$

Figure 4.1: Big-step semantics rules

```
while ∃d.(d, i) : l0 do
            select d with (d, i) : l0;
             l0 := l0 − (d, i);
             l0 := l0 + (d, j);
```

Figure 4.2: A statement simulating $i \gg_{l_0}^{in} j$

Pre: PH_1: PH
if $\exists d.(\text{PH}_1,d)$:*works_in*$\wedge$ (PH_2,d):*works_in* **then**
$PH := PH$ - PH_1;
while $\exists p.$ (PH_1,p):*treats* **do**
select p **with** (PH_1,p):*treats*;
treats := *treats* - (PH_1,p);
treats := *treats* + (PH_2,p);
while $\exists p.$ (p,PH_1):*ref_phys* **do**
select p **with** (p,PH_1):*ref_phys*;
ref_phys := *ref_phys* - (p,PH_1);
ref_phys := *ref_phys* + (p,PH_2);
$MS := MS$- PH_1;
Post: PH_1: $\neg PH$

Figure 4.3: An example of program for the fourth transformation

is thus possible to simulate $i \gg^{in} j$. Obviously, the same can be done with $i \gg^{out} j$.

Let us go back to the example of the hospital. One can try to express the fourth transformation in this imperative language. Obviously, that depends on the choice made for the logic. As we did not concern ourselves with logics yet, we will assume we use a first-order like logic with quantification on nodes. That is $\exists n.n : P$ means that there exist a node that is labeled with P and $\exists n_0, n_1.(n_0, n_1) : R$ means that there is an edge whose source is n_0 and whose target is n_1 labeled with R. The transformation does three main things: it checks that PH_1 and PH_2 work in the same department (this corresponds to an "if-then" statement), it removes the label PH from PH_1 (this corresponds to the atomic action) and it forwards all patient of PH_1 to PH_2(this corresponds to a "while"-loop). It is illustrated in Figure 4.3.

4.2 Translation to Java Code

Being able to use the programming language to define programs is interesting yet their is a huge gap between being able to write programs and being able to execute them. In order to cross this chasm, we translated our programs into another language, in this case Java, that can then be compile to executable code. This work was presented in [7].

Generating Java Code: For processing programs such as the one in Figure 4.3 and generating Java code, we use the Eclipse environment and, in partic-

```
public void Trans4(N PH1,PH2) {
        N p;

        if ∃d.(PH1,d):works_in∧ (PH2,d):works_in{
                g.deleteNodePH1,PH;
                while ∃p. (PH1,p):treats{
                        p = treats.selectSource(PH1);
                        g.deleteEdge(PH1,treats,p);
                        g.insertEdge(PH2,treats,p);
                }
                while ∃p. (p,PH1):ref_phys{
                        p = ref_phys.selectTarget(PH1);
                        g.deleteEdge(p,ref_phys,PH1);
                        g.addEdge(p,ref_phys,PH2);
                }
                g.deleteNode(PH1,MS);
        }
}
```

Figure 4.4: An example of Java program for the fourth transformation

ular, the Xtext[1] facilities for parsing, syntax highlighting and context-dependent help.

In order to generate Java code for programs, we parse them and then traverse the syntax tree with Xtext/Xtend, issuing calls to appropriate Java functions that manipulate a graph (which is initially the input graph provided in the program's `main` rule). Here is a glimpse at the Xtend code snippet that translates statements, in particular the `add` statement for roles:

```
def statement(Stmt s){
      switch s{
        Add_stmt: add(s.lvar,s.role,s.rvar)
        ...
      }
}
def add(String lvar,String role,String rvar)'''
       ≪graph≫.insertEdge(≪lvar≫,≪role≫,≪rvar≫);'''
```

Thus, a program fragment `R := R + (a,b);` is translated to a Java call `g.insertEdge(a, R, b);`, where the graph `g` is the current graph.

Assuming that the correct definition have been made, that is that g is defined as a graph, that all atomic concepts, atomic roles and individual names

[1] http://www.eclipse.org/Xtext/

$< rdf : RDF >$
$< rdf : Descriptionabout = "http : //www.w3.org/Home/Lassila" >$
$< s : Creatorrdf : resource = "http : //www.w3.org/staffId/85740" / >$
$< /rdf : Description >$

$< rdf : Descriptionabout = "http : //www.w3.org/staffId/85740" >$
$< v : Name > OraLassila < /v : Name >$
$< v : Email > lassila@w3.org < /v : Email >$
$< /rdf : Description >$
$< /rdf : RDF >$

Figure 4.5: An example RDF file

have been defined properly, the transformation defined in Figure 4.3 can be translated into a Java program, shown in Figure 4.4 that would be part of the class Hospital. It is worth noting that, as was true in our imperative programming language, this program is not executable as is because one also needs to instantiate the logic so that it is possible to evaluate the formulae that are used as conditions in the if and while statements. Some of the internal procedures have yet to be fully implemented. For instance, select currently only works on atomic concepts but it should not be a difficult task to extend it to more expressive conditions. As this is highly dependent on the chosen logic though, it is something that has to be defined in conjonction with the logic.

Transforming Graphs: Once a Java program has been generated for a given program, it can be compiled and linked with a library that provides graph manipulating functions such as the above-mentioned `insertEdge`. When executing this program, it remains to read an input file containing a graph description, to perform the transformation and to output the new graph. We represent graphs in the RDF [22] format or Resource Description Framework. The RDF format is the W3C recommendation for metamodelling of data since 1999. It uses URIs to identify objects. An example is given in Figure 4.5. Parsing and printing of RDF files is based on the Apache Jena framework[2].

4.3 Conclusion

In this chapter, we introduced a programming language to modify graphs. This programming language is imperative and uses classical statements of imperative programming languages, as “if-then-else” statements and “while” loops, in addition to more graph transformation-oriented statements that are the atomic

[2] http://jena.apache.org/

actions that are performed. The most interesting statement in the syntax is the non-deterministic "select" that acts as a binder for node variables used in the actions.

Although we have yet to discuss logics, which is the subject of the next chapter, this imperative programming language uses heavily logic formulae in its syntax. Both the "if-then-else" statement and the "while" loop use, as is usual, conditions that have to be Booleans and the "select" binds variables according to a formula so that only elements that make the formula true are possible values for the variable. The choice of the logic is thus of outmost importance already as it underlies the syntax of the graph transformations.

In order to be fully able to use the definition of our programming language, we had to implement ways to actually modify graphs. We give such an implementation that translates a program written in this language into a Java program. This Java program can then be executed and used to actually modify graphs.

Chapter 5

Toward a general logical framework

One way to modify graphs has been introduced and another will be shown in Chapter 8. They use logic both as part of the statements, viz. if-then statements or while loops, and as the language in which the properties are expressed. It is obvious that in these conditions, the choice of the logic, or possibly logics, that are used is key. This discussion is an extension of the one in [12].

In most cases, in order to prove the correctness of graph transformations, a logic is produced that can fit the needs of the transformations and of the properties that are under study. While finding a solution designed with a problem in mind is intuitively the best way to solve that problem, that also means that it is seldom applicable to other problems and that no clue is given on how to tailor the solution of a problem to another one.

A huge array of logics have been considered as possible solutions as the underlying structure with which transformations are built. In [8], for instance, a logic highly reminiscent of dynamic logics is introduced. Its main advantage is its expressiveness. Both complex properties of the graphs, like euclidian edge labels or the absence of cycles, and transformations can be efficiently expressed. On the other hand, its main shortcoming is that it is not decidable and thus proofs cannot be automated. Another possible choice is to use the monadic second-order logic. It is, in all generality, undecidable but, given restrictions on the structures, it becomes decidable. Furthermore, it allows to describe properties of graphs that cannot be expressed in first-order logic, say that a graph is bi-partite. In [58], GP, that stands for Graph Program, a framework to define graph transformations that is strongly linked to the categorial conception of graph transformations, is introduced. It comes with a Hoare-like system allowing to prove the correctness of a program. Still, once more, one may wish to prove programs correct without having to restrict oneself to a few given structures.

On the other side of the spectrum, in [15], $\mathcal{SROIQ}$, a decidable Description

Logic is used to reason about graph transformations. It is much less expressive but it is decidable. Its use is then restricted by conditions so that the Hoare-like calculus generates only formulae that stay in the logic. The shortcoming of having to design programs with the limitations of the logic is efficiently removed in [1] by using a more expressive logic that can take care of all transformations defined inside the logic itself. Furthermore, the article focuses on finite models which correspond, in general, better to what graphs are actually considered.

The problem though is still that all those logics have pros and cons and that, to someone who does not want to spend time looking at which logic corresponds the most closely to his needs, there is no way to take a decision on which one is to be used. On the other hand, someone could have been using a logic that is perfectly suitable to define graph properties but there is no guideline to tell whether or not this logic could actually be used to prove that the programs are correct or not. In this chapter, the goal is to take a look at and point out the requirements that the logics have to fulfill in order to be used in the verification of a program.

5.1 Logics and graphs

Our aim in this section is to discuss general requirements for a logic, say $\mathcal{L}$, that one may consider either to specify pre- and post- conditions of specifications, to label graphs and to be used as conditions in statements.

Let us first focus on the first use of $\mathcal{L}$.

Let $SP = (Pre, s, Post)$ be a specification. If SP is correct, then if a graph $G \models Pre$ and G rewrites to model G' via program $\mathcal{P}$, then $G' \models Post$. The first obvious condition is that $G \models Pre$ is actually defined. That is to say, graphs must be able to satisfy formulae. This is a condition that focuses on what the logic means, that is how a formula is interpreted. For instance, the formula $\forall x, y, z.\texttt{gcd}(x, y) = z \Rightarrow \exists a, b.a \times x + b \times y = z$ has a meaning if one tries to interpret it on $\mathbb{Z}$, the smallest ring of integers, but the meaning is much less clear if one wants to interpret it on graphs.

The delimitation between what can be interpreted on graphs and what cannot is not always that clear though. $\forall x.\texttt{odd}(x) \Rightarrow \exists y.\texttt{square}(x) = y \wedge \texttt{odd}(y)$ may also seem unadapted to be interpreted on graphs but, rewritten as $\forall x.x : \texttt{odd} \Rightarrow \exists y.(x, y) : \texttt{square} \wedge y : \texttt{odd}$, it does not seem so unnatural. Furthermore, by just changing the name of the propositions, one can get $\forall x.x : \texttt{human} \Rightarrow \exists y.(x, y) : \texttt{childOf} \wedge y : \texttt{human}$ which states that humans all have a human they are the child of which would be a perfectly fine formula on graphs representing human society.

Requirement 1. *Interpretations of Pre and Post assertions should be graphs. Thus, given a graph G and a formula ϕ, $G \models \phi$ must be defined.*

Even provided that this first requirement is respected, one has to be able to define properly graphs. This means that $\mathcal{L}$ needs to have ways to label both nodes and edges.

Requirement 2. *Interpretations of node formulae (concepts) should be nodes.*

Requirement 3. *Interpretations of edge formulae (roles) should be edges.*

Once more, this mostly means that the logic has to be tailored to deal with graphs. Essentially, what these requirements mean is that there should be unary and binary predicates (provided edges are considered as pairs of nodes) in the logic. Predicates of higher arity could be defined, say $\texttt{clique}(x, y, z)$ but they have to be interpreted on edges and nodes as there are no ternary structures in graphs. $\texttt{clique}(x, y, z)$ could be defined as $\texttt{friend}(x, y) \wedge \texttt{friend}(y, z) \wedge \texttt{friend}(z, x)$.

Going back to the arithmetic oriented formulae given previously, one of the main problems is what the connectors are. In these formulae, `gcd`, . and + are not predicates (unlike `odd`, for instance) but functions that have a result (here an integer). Obviously, functions are not always a problem. Even in these cases, = is an function that can be used as it produces a truth value. In more graph-oriented frameworks, the function `next` of lists for instance works well for edges. The issue is that functions actually need an increased arity (one for each elements and one for the result) to be expressed as predicates which means that functions of arity greater than 1 make little sense when dealing with graphs. Indeed, graphs are only composed of nodes and edges and all relations, and thus all properties, can only speak about such nodes and edges. It is obviously possible to define functions of higher arity than two but they could always be translated into formulae dealing only with edges and nodes that is only with arity 2 or less functions.

These requirements primarily check that there is a reason behind the choice of the logic. Three different arities make sense in the context of graphs: binary predicates for edges considered as pairs of nodes, unary predicates for nodes and nullary predicates for graphs.

The requirement that the labels have the good arity does not preclude the absence of such labels. For instance, one may want to use modal logic [10]. It provides no actual label for edges even though modal logic models are graphs. It is a perfectly acceptable choice of logic in which one does not discriminate between edge labels. Multi-modal logic [29], on the other hand, provides multiple labels for edges and allow to take into accounts the labeling of edges in the formulae.

5.2 Hoare-like calculus and weakest preconditions

Being able to find a logic able to express properties of the graph is, obviously, only part of the problem of finding a logic that is able to express the correctness of graph transformations. The logic must also be able to work with the verification calculus. This section focuses on the actions described in Section 4.1. When what is said can be more general, it is indicated.

As usual in Hoare logic[40], the verification calculus is reasoning upward. Let P_1 be a postcondition and s be a statement. One computes a precondition P_0 such that it is a certitude that if P_0 is true in a graph G, P_1 is true in any graph obtained from G by performing s. There may be several such P_0's so the weakest one, that is the one that implies all the others, is computed.

When applied to a specification $(Pre, s, Post)$, a precondition P_0 is generated for $Post$ to be true after having performed s. One then has to decide whether $Pre \Rightarrow P_0$.

The basic cases of the computations of weakest precondition deal with atomic actions (see Fig 5.1). To be more precise, to every atomic action is associated a so called substitution [40]. Such substitutions are the elementary building blocks allowing the verification of a program.

Definition 5.2.1. *Let a be an atomic action, as defined in Section 3.1. The substitution $[a]$ associated to a is the formula constructor that to each formula ϕ of $\mathcal{L}$ associates the formula $\phi[\alpha]$. Given a model $\mathcal{M}$, $\phi[a]$ is defined such that $\mathcal{M} \models \phi[a] \Leftrightarrow$ for all models $\mathcal{M}', \mathcal{M}' \Rightarrow_a \mathcal{M}$ implies $\mathcal{M}' \models \phi$.*

A logic $\mathcal{L}'$ is said to be closed under substitutions if for each action a, for each formula ϕ of $\mathcal{L}'$, $\phi[a]$ is also a formula of $\mathcal{L}'$.

$wp(C := \phi',\ \phi)$	$=$	$\phi[C := \phi']$
$wp(r := Q,\ \phi)$	$=$	$\phi[r := Q]$
$wp(C := i,\ \phi)$	$=$	$\phi[C := i]$
$wp(r := e,\ \phi)$	$=$	$\phi[r := e]$
$wp(C := C + i,\ \phi)$	$=$	$\phi[C := C + i]$
$wp(C := C - i,\ \phi)$	$=$	$\phi[C := C - i]$
$wp(r := r + e,\ \phi)$	$=$	$\phi[r := r + e]$
$wp(r := r - e,\ \phi)$	$=$	$\phi[r := r - e]$
$wp(new\_node(i),\ \phi)$	$=$	$\phi[new\_node(i)]$
$wp(del\_node(i),\ \phi)$	$=$	$\phi[del\_node(i)]$
$wp(new\_edge(e),\ \phi)$	$=$	$\phi[new\_edge(e)]$
$wp(del\_edge(e),\ \phi)$	$=$	$\phi[del\_edge(e)]$
$wp(i \gg^{in} j,\ \phi)$	$=$	$\phi[i \gg^{in} j]$
$wp(i \gg^{out} j,\ \phi)$	$=$	$\phi[i \gg^{out} j]$

Figure 5.1: Weakest preconditions for the actions defined in Section 3.1.

In Figure 5.1, the substitutions corresponding to the atomic actions defined in Section 3.1 are used. The computation of weakest preconditions is then extended to the statements presented in Chapter 4.1.

The weakest preconditions defined in Figure 5.2 follow usual Hoare Logic calculi [40]. The rules for statement sequence and the if-then-else statement are quite straightforward. ψ is the weakest precondition for ϕ after performing $s_0; s_1$ if it is the weakest precondition for ψ' after performing s_0 with ψ' the

$$wp(\texttt{select } i \texttt{ with } \phi',\ \phi) = \forall i.(\phi' \Rightarrow \phi) \qquad wp(s_0; s_1,\ \phi) = wp(s_0,\ wp(s_1,\ \phi))$$
$$wp(\texttt{if } \phi' \texttt{ then } s_0 \texttt{ else } s_1,\ \phi) = (\phi' \Rightarrow wp(s_0, \phi)) \wedge (\neg\phi' \Rightarrow wp(s_1, \phi))$$
$$wp(\texttt{while } \phi' \texttt{ do } \{inv\}s,\ \phi) = inv$$

Figure 5.2: Weakest preconditions for statements.

weakest precondition for ϕ after performing s_1. In a similar way, ψ is the weakest precondition for ϕ after performing `if` ϕ' `then` s_0 `else` s_1 if it is the weakest precondition for ϕ after performing s_0 knowing that ϕ' is satisfied and it is the weakest precondition for ϕ after performing s_1 knowing that ϕ' is not satisfied.

The rule for the non-deterministic select is more involved. It states that no matter which node is assigned to the variable i, whenever the condition ϕ' is satisfied so must be the post-condition as the only effect of the statement is changing the assignment of i. Finally, one may have realized that the rule for the *while* loop contained an additional part of the statement. This $\{inv\}$ is the invariant if the loop and it has to be provided by the user. The weakest precondition for the while loop is stated to be inv. Once more, it is the usual weakest precondition in Hoare logic. The weakest precondition simply checks that this invariant is satisfied before entering in the loop. Obviously, this is not sufficient to prove that the postcondition, that does not even appear in the calculus, is satisfied after the execution of the loop. Another set of conditions, namely verification condition, is introduced in Figure 5.3 to complement the weakest preconditions.

$$vc(a, \phi) = \top \qquad vc(\texttt{select } i \texttt{ with } \phi',\ \phi) = \top$$
$$vc(s_0; s_1,\ \phi) = vc(s_0,\ wp(s_1,\ \phi)) \wedge vc(s_1,\ \phi)$$
$$vc(\texttt{if } \phi' \texttt{ then } s_0 \texttt{ else } s_1,\ \phi) = vc(s_0, \phi) \wedge vc(s_1, \phi)$$
$$vc(\texttt{while } \phi' \texttt{ do } \{inv\}s,\ \phi) = (\phi' \wedge inv \Rightarrow wp(s, inv)) \wedge (\neg\phi' \wedge inv \Rightarrow \phi) \wedge vc(s, inv)$$

Figure 5.3: Verification conditions for statements.

Once more the definition of the verification conditions is usual. The goal of the verification conditions is to prove that the statements that form the core of while-loops behave as correct programs. To be more precise, let `while` ϕ `do` s be a statement whose invariant is chosen as inv. The verification condition checks that the specification $(\phi \wedge inv,\ s,\ inv)$ is correct and that if ϕ is not satisfied, that is the execution exits the loop, the postcondition must be satisfied. The other rules are simpler: the one for the if-then-else statement checks that whichever of the statements is executed, its loops behave correctly, and the one for sequence checks that each of the two statements contains only correct loops. As neither

the select statement nor any of the atomic actions may contain loops, there verification condition is always true.

These definitions may be usual and intuitively correct, one still has to prove formally that the calculus is sound.

Definition 5.2.2 (Correctness formula)**.** *We call* correctness formula *the formula* $correct(S) = (Pre \Rightarrow wp(s, Post)) \wedge vc(s, Post)$.

Theorem 5.2.1 (Soundness)**.** *Let* $S = (Pre, s, Post)$ *be a specification. If* $correct(S)$ *is valid, then for all graphs* G, G' *such that* $G \Rightarrow_s G'$, $G \models Pre$ *implies* $G' \models Post$.

This proof is done by induction on the semantic of the programming language. It is quite technical but not difficult. If not interested in this proof, one can skip to Section 5.3. We only consider here the atomic actions used in Section 4.1 in order to be coherent with the statements used.

Proof.

- Let us assume $s = C := C + i$. Then $correct(S) = vc(C := C + i,\ Post) \wedge (Pre \Rightarrow wp(C := C + i,\ Post)$. As $vc(C := C + i,\ Post) = \top$ and $wp(C := C + i,\ Post) = Post[C := C + i]$, $correct(SP) = Pre \Rightarrow Post[C := C + i]$. Let σ be a state such that $\sigma(Pre)$. As $correct(S)$ is valid, $\sigma(correct(S))$, that is $\sigma(Pre \Rightarrow Post[C := C + i])$. As $\sigma(Pre)$, $\sigma(Post[C := C+i])$. By definition of the substitutions, $\sigma(Post[C := C+i])$ implies that for any state σ' such that $\sigma \Rightarrow_{C:=C+i} \sigma'$, $\sigma'(Post)$. Thus $\sigma(Pre) \Rightarrow \sigma'(Post)$.

- Let us assume $s = C := C - i$. Then $correct(S) = vc(C := C - i,\ Post) \wedge (Pre \Rightarrow wp(C := C - i,\ Post)$. As $vc(C := C - i,\ Post) = \top$ and $wp(C := C - i,\ Post) = Post[C := C - i]$, $correct(S) = Pre \Rightarrow Post[C := C - i]$. Let σ be a state such that $\sigma(Pre)$. As $correct(S)$ is valid, $\sigma(correct(SP))$, that is $\sigma(Pre \Rightarrow Post[C := C - i])$ is. As $\sigma(Pre)$, $\sigma(Post[C := C - i])$. By definition of the substitutions, $\sigma(Post[C := C - i])$ implies that for any state σ' such that $\sigma \Rightarrow_{C:=C-i} \sigma'$, $\sigma'(Post)$. Thus $\sigma(Pre) \Rightarrow \sigma'(Post)$.

- Let us assume $s = R := R + (i, j)$. Then $correct(S) = vc(R := R + (i, j),\ Post) \wedge (Pre \Rightarrow wp(R := R + (i, j),\ Post)$. As $vc(R := R + (i, j),\ Post) = \top$ and $wp(R := R + (i, j),\ Post) = Post[R := R + (i, j)]$, $correct(S) = Pre \Rightarrow Post[R := R + (i, j)]$. Let σ be a state such that $\sigma(Pre)$. As $correct(S)$ is valid, $\sigma(correct(S))$, that is $\sigma(Pre \Rightarrow Post[R := R + (i, j)])$. As $\sigma(Pre)$, $\sigma(Post[R := R + (i, j)])$. By definition of the substitutions, $\sigma(Post[R := R + (i, j)])$ implies that for any state σ' such that $\sigma \Rightarrow_{R:=R+(i,j)} \sigma'$, $\sigma'(Post)$. Thus $\sigma(Pre) \Rightarrow \sigma'(Post)$.

- Let us assume $s = R := R - (i, j)$. Then $correct(S) = vc(R := R - (i, j),\ Post) \wedge (Pre \Rightarrow wp(R := R - (i, j),\ Post)$. As $vc(R := R - (i, j),\ Post) = \top$ and $wp(R := R - (i, j),\ Post) = Post[R := R - (i, j)]$, $correct(S) = Pre \Rightarrow Post[R := R - (i, j)]$. Let σ be a state such that $\sigma(Pre)$. As $correct(S)$ is valid, $\sigma(correct(S))$, that is $\sigma(Pre \Rightarrow Post[R :=$

$R-(i,j)]$). As $\sigma(Pre)$, $\sigma(Post[R := R-(i,j)])$. By definition of the substitutions, $\sigma(Post[R := R-(i,j))$ implies that for any state σ' such that $\sigma \Rightarrow_{R:=R-(i,j)} \sigma'$, $\sigma'(Post)$. Thus $\sigma(Pre) \Rightarrow \sigma'(Post)$.

- Let us assume $s = \texttt{select}\ i\ \texttt{with}\ \phi$. Then $correct(S) = vc(\texttt{select}\ i\ \texttt{with}\ \phi,\ Post)\ \wedge\ (Pre \Rightarrow wp(\texttt{select}\ i\ \texttt{with}\ \phi,\ Post)$. As $vc(\texttt{select}\ i\ \texttt{with}\ \phi,\ Post) = \top$ and $wp(\texttt{select}\ i\ \texttt{with}\ \phi,\ Post) = \forall i.(\phi \Rightarrow Post)$, $correct(SP) = Pre \Rightarrow \forall i.(\phi \Rightarrow Post)$. Let σ be a state such that $\sigma(Pre)$. Then $\sigma(\forall i.(\phi \Rightarrow Post))$. Let σ' be such that $\sigma \Rightarrow_{\texttt{select}\ i\ \texttt{with}\ \phi} \sigma'$. By definition, σ' is such that $\exists vi.\sigma' = \sigma^{[i:=vi]}$ and $\sigma'(\phi)$. As $\sigma'(\phi \Rightarrow Post)$; $\sigma'(Post)$. Hence $\sigma(Pre) \Rightarrow \sigma'(Post)$.

- Let us assume $s = s_0;\ s_1$. Then $correct(S) = vc(s_0;\ s_1,\ Post)\ \wedge\ (Pre \Rightarrow wp(s_0;\ s_1,\ Post)$. As $vc(s_0;\ s_1,\ Post) = vc(s_0, wp(s_1, Post)) \wedge vc(s_1, Post)$ and $wp(s_0;\ s_1,\ Post) = wp(s_0, wp(s_1, Post))$, $correct(SP) = vc(s_0, wp(s_1, Post)) \wedge vc(s_1, Post) \wedge (Pre \Rightarrow wp(s_0, wp(s_1, Post))$. Let σ be a state such that $\sigma(Pre)$. As $correct(S)$ is valid, $\sigma(correct(S))$. Let σ' be a state such that $\sigma \Rightarrow_{s_0;s_1} \sigma'$. Then there exists σ'' with $\sigma \Rightarrow_{s_0} \sigma''$ and $\sigma'' \Rightarrow_{s_1} \sigma'$. As $\sigma(Pre)$ and $\sigma(vc(s_0, wp(s_1, Post)) \wedge (Pre \Rightarrow wp(s_0, wp(s_1, Post))))$, by induction with $S_0 = (Pre, s_0, wp(s_1, Post))$, $\sigma''(wp(s_1, Post))$. As $correct(S)$, $vc(s_1, Post) \wedge (wp(s_1, Post) \Rightarrow wp(s_1, Post))$ is valid. Thus $\sigma''(vc(s_1, Post) \wedge (wp(s_1, Post) \Rightarrow wp(s_1, Post)))$. Once more, by induction with $S_1 = (wp(s_1, Post), s_1, Post)$, $\sigma'(Post)$. Thus $\sigma(Pre) \Rightarrow \sigma'(Post)$.

- Let's assume $s = \texttt{if}\ \phi\ \texttt{then}\ s_0\ \texttt{else}\ s_1$. Then $correct(S) = vc(\texttt{if}\ \phi\ \texttt{then}\ s_0\ \texttt{else}\ s_1,\ Post)\ \wedge\ (Pre \Rightarrow wp(\texttt{if}\ \phi\ \texttt{then}\ s_0\ \texttt{else}\ s_1,\ Post)$. As $vc(\texttt{if}\ \phi\ \texttt{then}\ s_0\ \texttt{else}\ s_1,\ Post) = vc(s_0, Post)) \wedge vc(s_1, Post)$ and $wp(\texttt{if}\ \phi\ \texttt{then}\ s_0\ \texttt{else}\ s_1,\ Post) = (\phi \Rightarrow wp(s_0, Post)) \wedge (\neg\phi \Rightarrow wp(s_1, Post)$, $correct(SP) = vc(s_0, Post)) \wedge vc(s_1, Post) \wedge (Pre \Rightarrow (\phi \Rightarrow wp(s_0, Post)) \wedge (\neg\phi \Rightarrow wp(s_1, Post))$. Let σ be a state such that $\sigma(Pre)$ and σ' be a state such that $\sigma \Rightarrow_{\texttt{if}\ \phi\ \texttt{then}\ s_0\ \texttt{else}\ s_1} \sigma'$, that is either $\sigma(\phi)$ and $\sigma \Rightarrow_{s_0} \sigma'$ or $\sigma(\neg\phi)$ and $\sigma \Rightarrow_{s_1} \sigma'$.

  - If $\sigma(\phi)$ then, as $correct(S)$ is valid, so is $vc(s_0, Post) \wedge (Pre \wedge \phi \Rightarrow wp(s_0, Post))$. As $\sigma \Rightarrow_{s_0} \sigma'$, by induction with $S_0 = (Pre \wedge \phi, s_0, Post)$, $\sigma'(Post))$.
  - else, as $correct(S)$ is valid, so is $vc(s_1, Post) \wedge (Pre \wedge \neg b \Rightarrow wp(s_1, Post))$. As $\sigma \Rightarrow_{s_1} \sigma'$, by induction with $S_1 = (Pre \wedge \neg\phi, s_1, Post)$, $\sigma'(Post))$.

  Thus $\sigma(Pre) \Rightarrow \sigma'(Post)$.

- Let us assume $s = \texttt{while}\ \phi\ \texttt{do}\ s'$. Then $correct(S) = vc(\texttt{while}\ \phi\ \texttt{do}\ s',\ Post)\ \wedge\ (Pre \Rightarrow wp(\texttt{while}\ \phi\ \texttt{do}\ s',\ Post)$. As $vc(\texttt{while}\ \phi\ \texttt{do}\ s, Post) = (\phi\ \wedge\ inv\ \Rightarrow wp(s', inv)) \wedge (\neg\phi\ \wedge\ inv\ \Rightarrow Post) \wedge vc(s', inv)$ and $wp(\texttt{while}\ \phi\ \texttt{do}\ s', Post) = inv$, $correct(S) = (\phi\ \wedge\ inv\ \Rightarrow wp(s', inv)) \wedge (\neg\phi\ \wedge\ inv\ \Rightarrow Post) \wedge vc(s', inv) \wedge (Pre \Rightarrow inv)$. Let σ be a state such

that $\sigma(Pre)$ and σ' be a state such that $\sigma \Rightarrow_{\texttt{while } \phi \texttt{ do } s'} \sigma'$. Then either $\sigma(\phi)$ and there exists σ'' such that $\sigma \Rightarrow_{s'} \sigma''$ and $\sigma'' \Rightarrow_{\texttt{while } \phi \texttt{ do } s'} \sigma'$ or $\sigma(\neg\phi$ and $\sigma = \sigma'$.

- If $\sigma(\phi)$, then there exist σ'' such that $\sigma \Rightarrow_{s'} \sigma''$ and $\sigma'' \Rightarrow_{\texttt{while } \phi \texttt{ do } s'} \sigma'$. As $correct(S)$ is valid, so is $vc(s', inv) \wedge (Pre \Rightarrow (\phi \wedge inv \Rightarrow wp(s, inv)))$. Thus $vc(s', inv) \wedge (Pre \wedge \phi \ \wedge \ inv \Rightarrow wp(s', inv))$ is also valid. By induction with $SP' = (Pre \wedge \phi \wedge inv, s', inv)$, $\sigma(Pre \wedge \phi \wedge inv) \Rightarrow \sigma''(inv)$. Also, as $correct(SP)$ is valid, so is $vc(s', inv) \wedge (inv \wedge \phi \Rightarrow wp(s', inv) \wedge (inv \wedge \neg\phi \Rightarrow Post) \wedge (inv \Rightarrow inv)$. Thus, by induction with $SP'' = (inv, \texttt{while } \phi \texttt{ do } s', Post)$, $\sigma''(inv) \Rightarrow \sigma'(Post)$. Thus $\sigma(Pre) \Rightarrow \sigma'(Post)$
- else, as $correct(S)$ is valid so is $(Pre \Rightarrow inv) \wedge (inv \ \wedge \ \neg \ \phi \ \Rightarrow Post)$. As $\sigma(Pre)$, $\sigma(inv)$. Furthermore, as $\sigma(\neg\phi)$, $\sigma(Post)$. But, as $\sigma' = \sigma$, $\sigma(Pre) \Rightarrow \sigma'(Post)$

Thus $\sigma(Pre) \Rightarrow \sigma'(Post)$.

□

Now that we have defined the correctness formula, we know that the verification problem can be translated to the validity problem for a given formula. The next two requirements are made to allow us to be able to deal with this problem. We chose to require that the logic be able to express the correctness formula and thus add requirements to the logic. Other solutions could be to allow for weaker logics and devise a reasoning system for the logic extended with the missing elements or to actually devise a reasoning system tailored for the specific problem of deciding the validity of the correctness formula.

In all generality, the requirements may be strongly tied to how transformations are made. For instance, the computation of the weakest precondition for the "if-then-else" statement is $wp(\texttt{if } \phi \texttt{ then } s_0 \texttt{ else } s_1,\ Q) = (\phi \Rightarrow wp(s_0, Q)) \wedge (\neg\phi \Rightarrow wp(s_1, Q))$. This means that whenever "if-then-else" statements are part of the transformation language, the logic must provide the conjunction, the negation and the implication of formulae. On the other hand, let us consider a transformation language that contains only sequences of atomic actions. In this case, the computation of the verification conditions would be moot and the computation of the weakest preconditions could not require any additional conditions on $\mathcal{L}$.

The first obvious realization is that substitutions are used and that it is thus mandatory to be able to deal with them. This is the requirement that is tackled in Section 5.3.

Another condition implied by the transformation language chosen is the weakest-precondition for the "select" statement. It introduces an universal quantifier. The presence of universal quantifiers, as there is a priori no restriction on the number of "select" statements used, in $\mathcal{L}$ is an important restriction as not all logics handle it. For instance, it is not part of modal logics. This requirement is studied in more details in Section 5.4.

5.3 Closure under substitution

The stated goal of this work is to provide general conditions on $\mathcal{L}$ and the conditions heretofore examined are highly dependent on a possibly misguided choice of transformation language. On the other hand, one clear requirement is that it is needed to be able to deal with atomic actions, that is it is needed to handle substitutions.

Requirement 4. *$\mathcal{L}$ must be closed under substitutions.*

It is obvious that if the requirement above is not satisfied, the calculus of weakest preconditions may generate a formula which is not in $\mathcal{L}$. As general as this requirement may seem to be, it is still highly dependent on what transformations are actually performed. By adding or removing atomic actions, remember that for instance the set of atomic actions used in Definition 4.1.2 is different from the one introduced in Section 3.1, the requirement that $\mathcal{L}$ be closed under substitutions changes.

Nevertheless, this may seem to be a really abstract condition and, as very few existing logics have been created with substitutions in mind, there is no straightforward way to detect logics that are closed under substitutions. One could thus be interested in what can be a clue that a logic is closed under substitutions.

The first thing one may have noticed when the atomic actions were introduced in Section 3.1 is that atomic actions deal with nodes and edges. That may seem to be insignificant but that actually means that in order to be closed under substitutions, the logic has to be able to speak about individual nodes and edges. In most cases, being able to speak about nodes is enough though as an edge e can be considered as the pair $(s(e), t(e))$. Being able to deal with individuals is not straightforward though.

Several mechanisms exist to deal with individuals. Basically, an individual is something that has one and only one instance. In first-order logic if one wants to speak about the individual i, it is possible to introduce a new unary predicate i and add to the formula that there is a node that is the individual i ($\exists x.i(x)$) and that all nodes that are labeled with i are actually the same ($\forall x, y.i(x) \wedge i(y) \Rightarrow x = y$). In other logics that lack equality or quantifiers, other mechanisms can be used. Nominals are part of some description and hybrid logics and the model checks that there exists one and only one element that is labeled with the individual.

Once again, this requirement depends enormously on the choice of atomic actions that has been made. If the atomic actions do not use individuals, say the only atomic action is $C := \phi$, this requirement becomes much less severe.

Another thing that appears quite obvious is that one needs to be able to speak about disconnected nodes. To be more precise, when one looks at the result of creating a new edge between two nodes, one has to be able to deal with both nodes even though they are not yet a priori connected. Thus the logic needs to provide a way to speak about nodes that are different from the one under current study. Node formulae are, as we have seen, essentially unary

predicates that are evaluated over one node at a time. When one is dealing with edges, in particular when creating new edges, one has to be able to deal with both the source and the target of the edge. In first order logic, the quantifiers allow to quantify over nodes and thus to consider multiple nodes at a time. In logics lacking such powerful quantifiers, it is possible to try to replace them with less potent tools like an universal role or the @ constructor of hybrid logic.

More details will be given when actual logics are studied in Chapter 6, Chapter 7 and Chapter 9.

5.4 Handling the select

In the previous section, the impact of substitutions has been looked at. Those are not the only problems that arise when trying to find a logic expressive enough to be used in the verification of graph transformations.

The weakest precondition for the select statement is $wp(\texttt{select}\ i\ \texttt{with}\ \phi,\ Q) = \forall i.(\phi \Rightarrow Q)$. It introduces a universal quantifier on variables which means that the logic has to be able to handle them. A lot of logics, say modal logic, do not allow for variables thus this may seem to restrict drastically the number of logics that can be used.

It is possible to limit the impact of this restriction though. Let us first observe that the correctness formula, provided restrictions on Pre, $Post$, the invariants and the conditions used in s, can be considered essentially universally quantified. This is a good thing for a reason that will become clear in a short while.

Definition 5.4.1. *A formula is* essentially universally quantified *if its prenex normal form contains only universal quantifiers.*
A formula is essentially existentially quantified *if its prenex normal form contains only existential quantifiers.*

One can easily see that the only rule in the computation of the weakest preconditions and the verification conditions that introduces quantifiers is the weakest precondition for the select statement. As weakest preconditions are never negated, $Corr$ is thus essentially universally quantified if no other quantifier is used.

One could, in this condition, try to extend the logic to one using quantifiers with the hope of keeping an essentially universally quantified formula. If one looks at the generation of weakest preconditions and verification conditions, it appears that conditions in if-statements and while-loops have both positive and negative occurrences. This means that, if one wants to keep an essentially universally quantified formula, one must ban quantifiers in these conditions. The same goes for the loop invariants that appear both on the left- and right-hand side of the implications in $(\phi' \wedge inv \Rightarrow wp(s, inv))$.

However, if the conditions in the select statements are essentially existentially quantified, $Corr$ will be essentially universally quantified. Let us for instance look at the statement $s = \texttt{select}\ p\ \texttt{with}\ \exists q.(p, q)\ :\ R$. Its weakest

precondition is $wp(s, Post) = \forall p.((\exists q.(p, q) : R) \Rightarrow Post)$ whose prenex normal form is $\forall p.\forall q.((p, q) : R \Rightarrow Post)$. It is essentially universally quantified. That means that it is possible to use select statements that use as many variables as one may want.

On the other hand, Pre only occurs on the left and can thus be essentially existentially quantified and still yield an essentially universally quantified $Corr$ while $Post$ only occurs on the left and can thus be essentially universally quantified. This allows to slightly extend the possibilities offered in the definition of the specifications but this can not be used very efficiently in transformations themselves.

$Corr$ being essentially universally quantified is interesting as the problem that interests us at this point is whether or not $Corr$ is valid. Instead one can look at the exact equivalent question of wether $\neg Corr$ is satisfiable. But then $\neg Corr$ is essentially existentially quantified. It is thus possible to skolemize it and replace all variables with constants. The condition on the logic is thus no longer that it contains a universal quantification on variables but that it can deal with constants.

This discussion seems to be fairly specific to the framework that we have chosen that uses an imperative programming language-like structure. As discussed in Section 1.3, this is far from the only possible approach possible. However, in Chapter 8, we will observe the case of rule-based transformations in details and see that the same kind of problems invariably appears as one has to find a way to express where transformations are to be applied. The problem would be solved if the transformation where to be applied at pre-chosen nodes and edges must it is usually mandatory to be able to use variables to consider when actions are applied in actual transformations.

5.5 Conclusion

In this chapter, we started to deal with the logic underlying the tranformations of graphs that we introduced and the verification of such transformations.

This chapter is key for several reasons. First and foremost, it introduces for the first time restrictions that one may want to enforce on the logics that one uses to express graph properties. This is the main task that has been defined as this work's aim. Some of these properties, namely that the logic is actually suited to talk about graphs, are rather intuitive and easily checked. Others are more involved and depend highly on numerous factors.

This chapter also introduces the Hoare-like calculus that is used to do the actual verification of graph transformations. As doing the verification is our ultimate goal and the Hoare-like calculus is the means that we have chosen to this goal, its importance must not be underestimated. The Hoare-like calculus presented in this chapter corresponds to the imperative language defined in Chapter 4. Another version is presented in Chapter 8 when another kind of structure for graph transformations is presented. It is important to note, even though it may seem obvious, that the calculus depends on the way transforma-

tions are handled. Nonetheless, they have much in common and it is possible to come up, as we did, with general conditions on the logics that one uses to describe graphs that make sense in several different transformation systems.

One of the most noteworthy points about the Hoare-like calculus is that it makes use of substitutions. Substitutions are short-hands that allow to insert the transformations directly into the logic. They are the first source of trouble when one wants to do program verification as, a priori, there is no reason for a logic to be able to handle them. Moreover, as each substitution is linked to one atomic action, they actually depend on the choice of atomic actions that one makes.

The first condition on logics that we underline is thus that the logic should be able to deal with substitutions, that is that it should be closed under substitutions. If a logic is closed under substitutions, it is possible, albeit maybe at a cost in term of complexity, to decide whether a formula with substitutions is satisfiable, or valid, using, possibly pre-existing, tools designed for that logic. This condition is studied further in Chapter 6 as we examine a family of logics, namely Description Logics, and prove which are closed under substitutions.

The second condition that arises is that the logic is able to deal with the "select"-statement. It may seem at first sight to be a condition that actually only exists in the case of the imperative programming language and thus an arbitrary one. We introduce, in Chapter 8, another kind of transformation systems using rules and strategies that allows us to better illustrate what this condition is. Essentially, the condition states that the logic is able to deal with the variables that occur when one tries to apply transformations on elements that are not pre-determined.

One should note that failing these tests does not mean that the logic is not suited for graph transformation verification. Indeed, the goal is to be able to use existing tools instead of requiring the user to have to program new algorithms to deal with the modified version of the logic that one wants to use. Yet it is always possible to do so as we actually did for $\mathcal{ALCQ}$, one of the logics presented in Chapter 6. Part of this implementation is shown in Chapter 10.

Chapter 6

Using Description Logics

In the previous chapters, an imperative programming language has been introduced to deal with graph transformations. After that, general requirements for logics have been introduced. Yet, until now, no logic has been formally introduced.

As the first requirements state that the logic must be tailored to speak about graphs, let us consider logics that are widely used to deal with graphs. Description Logics [5] form one of the most widely used family of logics to describe graphs. They are the underlying logics that are used in the W3C [72] defined OWL [38] standard for representing data and SPARQL [71] standard for queries. Nonetheless, these standards are only meant to define static properties about graphs and query them and do not allow to modify them.

Some DLs have been created with actions in mind though. DLs that are closely related to dynamic logics like $\mathcal{D}_{\mathcal{ALCO}@}$ introduced in [18] and $\mathcal{PDLC}$ introduced in [74] use a possible model approach. In that kind of logics, states, that in this case are graphs, are connected by actions that are described by their effects and their preconditions. On the other hand, the goal of this work is to define clear atomic actions that can be combined to produce more elaborate transformations.

Such an approach is not completely novel. Very expressive Description Logics, as $\mathcal{ALHOIQ}br$ [1], have been introduced to model graph-structured data. This logic has two main advantages: the finite satisfiability of a formula is decidable and it contains constructors that allow to write every atomic action introduced as formulae of the logic. Nonetheless, the main objective of this work is not to introduce the perfect logic to deal with graph transformations but to look at what characteristics a logic should have to be considered as a candidate to reason about graph transformations.

A great variety of Description Logics have been described depending on what is to be their use. Some of them, DL-Lite [17] for instance, have been created to be tractable. These logics are meant to be used to describe large ontologies and thus their complexities have to be as low as possible. This tractability is obtained at the cost of expressiveness. On the other hand, some of the

most expressive logics are undecidable. In particular allowing complex roles in number restrictions, for instance role composition and inverse [36] or transitivity and intersection [35], cyclic definitions of role inclusions [41] or role boxes [73] yields undecidable logics.

Testing all known Description Logics is unfeasible. Instead, the logic $\mathcal{ALC}$ is considered with some of its extensions. This is a more detailed version of the work presented in [13].

6.1 Syntax of Description Logics

In this chapter, we define the syntax of the description logics we consider. We start by the elementary logic $\mathcal{ALC}$. $\mathcal{ALC}$, and all of its extensions, are defined using a signature (**C, R, I**) where **C** is a set of *concept names*, **R** is a set of *role names* and **I** a set of *individuals*.

Let us first define the concepts, that is the node labels.

Definition 6.1.1 ($\mathcal{ALC}$ concepts)**.** *The set of concepts, $\mathcal{C}$, is defined as:*

ϕ	$::=$	$\bot$	*(empty concept)*
	$\mid$	C	*(concept name)*
	$\mid$	$\neg\, \phi$	*(negation)*
	$\mid$	$\phi \sqcap \phi$	*(conjunction)*
	$\mid$	$\phi \sqcup \phi$	*(disjunction)*
	$\mid$	$\exists R.\phi$	*(exists)*
	$\mid$	$\forall R.\phi$	*(for all)*

*where c is a concept name ($\in$ **C**) and R is a role. As usual $\top$ is defined as $\not\bot$.*

Les us now define the roles of edge labels.

Definition 6.1.2 ($\mathcal{ALC}$ roles)**.** *The set of roles is **R**.*

Very often, DL formulae are split into several parts: ABoxes, TBoxes and concepts. ABoxes are meant to store the information on particular nodes and edges: how are they labeled, which ones are known to be the same or different.

Definition 6.1.3 (ABox)**.** *An* individual assertion *A is defined as:*

A	$::=$	$i : \phi$	*(concept assertion)*
	$\mid$	$i = j$	*(equality)*
	$\mid$	$i \neq j$	*(inequality)*
	$\mid$	$i\ R\ j$	*(positive role assertion)*
	$\mid$	$i\ \neg R\ j$	*(negative role assertion)*

where ϕ is a concept, R is a role and i and j are individuals. An ABox *is a finite set of individual assertions.*

TBoxes contain the hierarchy of concepts that is which concepts are included in which others.

Definition 6.1.4 (TBox)**.** *A* TBox *is a finite set of concept inclusions of the form $\phi \subseteq \psi$, where ϕ and ψ are concepts.*

Other entities may be used, such as RBoxes. RBoxes are used to store the role hierarchy. They usually allow some role constructors (sequence, union) that are not supported in other parts of the logics. In these cases, if strong restrictions are not enforced, RBoxes can very easily lead to undecidable logics [41, 73]. Most of the logics that will be studied henceforth do not use RBoxes for this reason.

Definition 6.1.5 (RBox)**.** *A* RBox *is a finite set of role inclusions of the form* $R_0 \subseteq R_1$*, where* R_0 *and* R_1 *are roles.*

Each extension of $\mathcal{ALC}$ adds new concept or role constructors. Each constructor is associated to a letter added to the name of the logic[1]. The most popular constructors that we consider in this chapter are $\mathcal{Q}$, $\mathcal{O}$, $\mathcal{S}elf$, $\mathcal{I}$ and $\mathcal{U}$, defined as follows:

$\mathcal{Q}$	$(\geq n\ R\ \phi)$	(at least)
$\mathcal{O}$	$\{a\}$	(nominal)
$\mathcal{S}elf$	$\exists S.Self$	(local reflexivity)
$\mathcal{I}$	R^-	(inverse role)
@	$@_a\phi$	(concept assertion)
$\mathcal{U}$	U	(universal role)

where i is an individual, R is a role and ϕ is a concept.

Example 6.1.1. *For example, the concept* $\mathfrak{C} = (\{\text{PH}_1\} \sqcap (\exists\ \text{works\_in}.DE) \sqcap (\forall\ \text{ref\_phys}^-.\neg\ \text{PA}_1)) \sqcup ((<3\ \text{treats}\ \top))$ *is the concept satisfied by* $\text{PH}_1(\{\text{PH}_1\})$ *if she works in a department* $((\exists\ \text{works\_in}.DE))$ *and if all the patients whose referent physician she is are not* $\text{PA}_1(\forall\ \text{ref\_phys}^-.\neg\ \text{PA}_1)$ *or by anyone treating strictly less than 3 patients* $((<3\ \text{treats}\ \top))$*.*

Now that formula of the logics have been introduced, one has to know how to interpret them.

Definition 6.1.6 (Interpretation)**.** *An interpretation* $\mathcal{I}$ *is a pair* $\{\Delta, .^{\mathcal{I}}\}$ *where* Δ *is the universe (set of elements) and* $.^{\mathcal{I}}$*, the valuation, is a function that maps each concept name to a subset of* Δ*, each role name to a subset of* $\Delta \times \Delta$ *and each individual to an element of* Δ*.*

The way interpretations are used in the verification of graph transformations, Δ is actually the N of the graphs, that is the set of nodes. Once more, one has to note that edges are not considered as separate entities but as couples of nodes.

Below, we define the valuation of roles and concepts:

[1] see, e.g., http://www.cs.man.ac.uk/~ezolin/dl/

$$
\begin{array}{lcl}
\bot^{\mathcal{I}} & = & \emptyset \\
(\neg\ \phi)^{\mathcal{I}} & = & \Delta \backslash \phi^{\mathcal{I}} \\
(\phi\ \sqcap\ \psi)^{\mathcal{I}} & = & \phi^{\mathcal{I}} \cap \psi^{\mathcal{I}} \\
(\phi\ \sqcup\ \psi)^{\mathcal{I}} & = & \phi^{\mathcal{I}} \cup \psi^{\mathcal{I}} \\
(\geq n\ R\ \phi)^{\mathcal{I}} & = & \{\delta \in \Delta | \#(R^{\mathcal{I}}_{\phi}(\delta)\}) \geq n\} \\
(< n\ S\ \phi)^{\mathcal{I}} & = & \{\delta \in \Delta | \#(R^{\mathcal{I}}_{\phi}(\delta)) < n\} \\
(\exists R.\phi)^{\mathcal{I}} & = & \{\delta \in \Delta | R^{\mathcal{I}}_{\phi}(\delta) \geq 1\} \\
(\forall R.\phi)^{\mathcal{I}} & = & \{\delta \in \Delta | R^{\mathcal{I}}_{\neg\phi}(\delta) < 1\} \\
\{i\}^{\mathcal{I}} & = & i^{\mathcal{I}} \\
(\exists R.Self)^{\mathcal{I}} & = & \{\delta \in \Delta | (\delta, \delta) \in R^{\mathcal{I}}\} \\
(@_i\phi)^{\mathcal{I}} & = & \Delta \text{ if } i \in \phi^{\mathcal{I}}, \emptyset \text{ otherwise} \\
(\mathrm{U})^{\mathcal{I}} & = & \Delta \times \Delta \\
(R^-)^{\mathcal{I}} & = & \{(\delta, \delta') \in \Delta \times \Delta | (\delta', \delta) \in R^{\mathcal{I}}\}
\end{array}
$$

where $R^{\mathcal{I}}_{\phi}(\delta)$ is $\{\delta' \in \Delta | (\delta, \delta') \in R^{\mathcal{I}} \wedge \delta' \in \phi^{\mathcal{I}}\}$ and $\#(V)$ stands for the cardinal of V.

Definition 6.1.7 (Model)**.** *We say that an interpretation* $\mathcal{I}$ satisfies *an assertion* $i : \phi$ *(resp.* $i = j$*,* $i \neq j$*,* $i\ R\ j$*,* $i\ \neg R\ j$*) if* $i^{\mathcal{I}} \in \phi^{\mathcal{I}}$ *(resp.* $i^{\mathcal{I}} = j^{\mathcal{I}}$*,* $i^{\mathcal{I}} \neq j^{\mathcal{I}}$*,* $(i^{\mathcal{I}}, j^{\mathcal{I}}) \in R^{\mathcal{I}}$*,* $(i^{\mathcal{I}}, j^{\mathcal{I}}) \notin R^{\mathcal{I}}$*). We say that an interpretation* $\mathcal{I}$ *is a* model *of an ABox* $\mathcal{A}$ *if* $\mathcal{I}$ *satisfies all the assertions of* $\mathcal{A}$*. We say that an interpretation* $\mathcal{I}$ satisfies *a concept inclusion* $\phi \subseteq \psi$ *if* $\phi^{\mathcal{I}} \subseteq \psi^{\mathcal{I}}$*. We say that an interpretation* $\mathcal{I}$ *is a* model *of a TBox* $\mathcal{T}$ *if* $\mathcal{I}$ *satisfies all the concept inclusions of* $\mathcal{T}$*. We say that an interpretation* $\mathcal{I}$ satisfies *a concept* ϕ *if* $\phi^{\mathcal{I}} \neq \emptyset$*.*

Let us assume that, from now on, using the atomic actions defined in Section 3.1, substitutions are defined.

Substitutions define an additional concept constructor. We use σ in logic names to signify that substitutions are allowed as constructors. It is important to note that this constructor can only be used to build the concept part of the considered formulae and not in assertions nor in concept inclusions. We also identify edges e with the pair $(s(e), t(e))$ in order to be able to identify them uniquely. Thus actions like $R := e$ become $R := (i, j)$.

$\sigma \qquad \phi\theta \qquad$ (explicit substitution)

where ϕ is a concept and θ is a substitution.

For instance, $\phi[R := R + (i, j)]$ means that ϕ is satisfied after adding the pair (i, j) to the valuation of R. The interpretation is thus modified accordingly.

Definition 6.1.8 (Valuation of explicit substitutions)**.** *Let* C *be a concept name,* i *and* j *be individuals,* ϕ *be a concept and* R *be a role name, the valuation of explicit substitutions is defined as follows:*

- $(\phi\epsilon)^{\mathcal{I}} = \phi^{\mathcal{I}}$
- $(\phi[C := C + i])^{\mathcal{I}} = \phi^{\mathcal{J}}$ *where* $\Delta^{\mathcal{J}} = \Delta^{\mathcal{I}}$*,* $\forall D \in \mathbf{C}, D \neq C, D^{\mathcal{J}} = D^{\mathcal{I}}$*,* $\forall R \in \mathbf{R}, R^{\mathcal{J}} = R^{\mathcal{I}}$ *and* $C^{\mathcal{J}} = C^{\mathcal{I}} \cup \{i^{\mathcal{I}}\}$*.*
- $(\phi[C := C - i])^{\mathcal{I}} = \phi^{\mathcal{J}}$ *where* $\Delta^{\mathcal{J}} = \Delta^{\mathcal{I}}$*,* $\forall D \in \mathbf{C}, D \neq C, D^{\mathcal{J}} = D^{\mathcal{I}}$*,* $\forall R \in \mathbf{R}, R^{\mathcal{J}} = R^{\mathcal{I}}$ *and* $C^{\mathcal{J}} = C^{\mathcal{I}} \backslash \{i^{\mathcal{I}}\}$

- $(\phi[R := R + (i,j)])^{\mathcal{I}} = \phi^{\mathcal{J}}$ *where* $\Delta^{\mathcal{J}} = \Delta^{\mathcal{I}}$, $\forall C \in \mathbf{C}, c^{\mathcal{J}} = c^{\mathcal{I}}$, $\forall P \in \mathbf{R}, P \neq R, P^{\mathcal{J}} = P^{\mathcal{I}}$ *and* $R^{\mathcal{J}} = R^{\mathcal{I}} \cup \{(i^{\mathcal{I}}, j^{\mathcal{I}})\}$
- $(\phi[R := R - (i,j)])^{\mathcal{I}} = \phi^{\mathcal{J}}$ *where* $\Delta^{\mathcal{J}} = \Delta^{\mathcal{I}}$, $\forall C \in \mathbf{C}, c^{\mathcal{J}} = c^{\mathcal{I}}$, $\forall P \in \mathbf{R}, P \neq R, P^{\mathcal{J}} = P^{\mathcal{I}}$ *and* $R^{\mathcal{J}} = R^{\mathcal{I}} \backslash \{(i^{\mathcal{I}}, j^{\mathcal{I}})\}$

In this chapter, we are interested in how the addition of substitutions to concepts affects the expressive power of a given Description Logic. For now, explicit substitutions can only be used in the definition of a concept (and thus are forbidden in ABoxes and TBoxes).

Definition 6.1.9 (Equivalence)**.** *Two concepts ϕ and ψ are said to be* equivalent *if, for every interpretation $\mathcal{I}$, $\phi^{\mathcal{I}} = \psi^{\mathcal{I}}$.*

A logic $\mathcal{L}_1$ is at most as expressive as *a logic $\mathcal{L}_2$ wrt. concepts written $\mathcal{L}_1 \leq_\phi \mathcal{L}_2$ if every concept in $\mathcal{L}_1$ has an equivalent concept in $\mathcal{L}_2$. Two logics $\mathcal{L}_1$ and $\mathcal{L}_2$ are* concept-equivalent *if $\mathcal{L}_1 \leq_\phi \mathcal{L}_2$ and $\mathcal{L}_2 \leq_\phi \mathcal{L}_1$*

Definition 6.1.10 (Closure under substitutions)**.** *A logic L is said to be* closed under substitutions *if L and $L\sigma$ are concept-equivalent.*

6.2 Closed DLs

In this section, the focus is on the most expressive of the Description Logics considered. The DLs going from $\mathcal{ALCU\mathcal{O}}$ to $\mathcal{ALCQUIO}@\mathcal{S}elf$ are proved to be closed under substitutions.

Let us notice first that in $\mathcal{ALCQUIO}@\mathcal{S}elf$ some constructs are rendundant. In particular, when a DL includes $\mathcal{O}$ (nominals), assertions can be rewritten as concept inclusions. Indeed, the individual assertion $i : \phi$ (resp. $i = j$, $i \neq j$, $i\ R\ j$, $i\ \neg R\ j$) is equivalent to the concept inclusion $\{i\} \subseteq \phi$ (resp. $\{i\} \subseteq \{j\}$, $\{i\} \subseteq \neg\{j\}$, $\{i\} \subseteq \exists R.\{j\}$, $\{i\} \subseteq \forall R.\neg\{j\}$). When a DL includes $\mathcal{U}$ (universal role), concept inclusions can be rewritten as mere concepts. Actually, the concept inclusion $\phi \subseteq \psi$ is equivalent to the concept $\forall U.(\neg\phi \sqcup \psi)$.

As both $\mathcal{O}$ and $\mathcal{U}$ are part of $\mathcal{ALCQUIO}@\mathcal{S}elf$, we consider for this chapter that the TBox and the ABox are empty. Additionally, concept definition, $[C := i]$, can be rewritten in presence of nominals into a concept assignment $[C := D]$ with $D = \{i\}$. This action will thus not be considered in the following.

Theorem 6.2.1. *$\mathcal{ALCQUIO}@\mathcal{S}elf$ is closed under substitutions.*

The proof of Theorem 6.2.1 is done by providing a terminating rewriting system consisting of concepts rewrite rules which translates concepts of $\mathcal{ALCQUIO}@\mathcal{S}elf$ into concepts of $\mathcal{ALCQUIO}\mathcal{S}elf$ (without substitutions). The rewrite system, we call $\mathcal{T}$, is given below. It consists of 90 rules. The idea is to associate to each possible pair concept-substitution an equivalent substitution-free concept. In the following, these pairs are grouped mainly by concept constructors. The definitions of the rules need some variables: let C and C' be different concept names, ϕ and ψ be concepts, R and R' be role

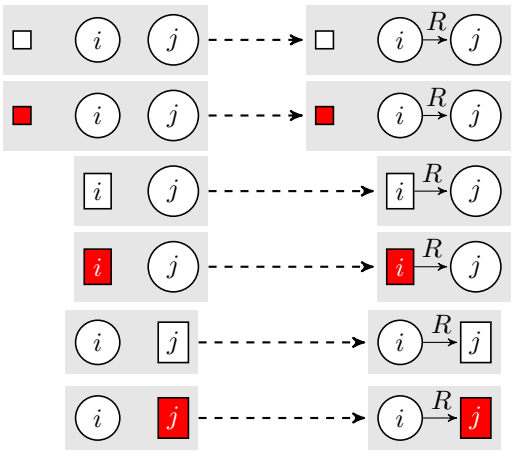


Figure 6.1: Example illustrating part of rule 3 ($C[R := R + (i,j)] \rightsquigarrow C$). The nodes satisfying C are filled in red, the current node is a rectangle. The substitution occurs when going from left to right. The interpretation of C is left unmodified.

names such that $R \neq R'$, θ be a substitution, i and j be individuals. For ease of reading, we define $\phi \Rightarrow \psi$ as $\neg\phi \sqcup \psi$. The notation $\bowtie$ stands for either $<$ or $\geq$. As usual, in presence of $\mathcal{I}$, R^{--} is understood as R.

Some intuition of why each rule is correct is given but the formal proof that the rewriting system both terminates and does not modify the interpretation is given afterwards. Some of the rules are unnecessary. For instance, rules are defined both for $(\exists R.\phi)[C := C + i]$ and for $(\neg\forall R.\neg\phi)[C := C + i]$ which are actually the same. Obviously, the interpretation of the results of both applications are the same and they were kept only for ease of understanding.

1 $\bot\ \theta \rightsquigarrow \bot$

The interpretation of $\bot$ is empty even after performing any action.

2 $\{i\}\ \theta \rightsquigarrow \{i\}$

The interpretation of the nominals does not depend on the interpretation of any concept or role name.

3 $C[R := R \pm (i,j)] \rightsquigarrow C$

4 $C[C' := C' \pm i] \rightsquigarrow C$

5 $C[R := Q] \rightsquigarrow C$

6 $C[R := (i,j)] \rightsquigarrow C$

7 $C[i \gg^{in} j] \rightsquigarrow C$

8 $C[i \gg^{out} j] \rightsquigarrow C$

The interpretation of a concept name does not depend on the interpretation of any other concept or role name. Figure 6.1 shows several possible states before the action $R := R + (i, j)]$ is performed and its result. It does not show the initial graphs where $(i, j) : R$ already, in which case nothing happens, or where $i = j$.

9 $C[C := C + i] \rightsquigarrow C \sqcup \{i\}$

After one labeled node i with the concept name C, the nodes labeled C are those that were labeled C before (those satisfying C) plus the node i (that satisfies $\{i\}$).

10 $C[C := C - i] \rightsquigarrow C \sqcap \neg\{i\}$

After one removed the label C from node i , the nodes labeled C are those that were labeled C before (those satisfying C) minus the node i (that satisfies $\{i\}$).

11 $C[C' := \psi] \rightsquigarrow C$

12 $C[C := \psi] \rightsquigarrow \psi$

If $C \neq C'$, its valuation is left unmodified. Otherwise, after assigning to C the same valuation as ψ, the nodes that satisfy C are exactly those that satisfied ψ.

13 $(\neg\phi)\ \theta \rightsquigarrow \neg(\phi\ \theta)$

14 $(\phi \sqcup \psi)\ \theta \rightsquigarrow \phi\ \theta \sqcup \psi\ \theta$

15 $(\phi \sqcap \psi)\ \theta \rightsquigarrow \phi\ \theta \sqcap \psi\ \theta$

16 $(@_a C)\ \theta \rightsquigarrow @_a(C\ \theta)$

Substitutions are propagated along boolean operators in the obvious way.

17 $\exists R.Self[C := C \pm i] \rightsquigarrow \exists R.Self$

18 $\exists R^-.Self[C := C \pm i] \rightsquigarrow \exists R^-.Self$

19 $\exists R.Self[C := \psi] \rightsquigarrow \exists R.Self$

20 $\exists R^-.Self[C := \psi] \rightsquigarrow \exists R^-.Self$

21 $\exists R.Self[R' := R' \pm (i, j)] \rightsquigarrow \exists R.Self$

22 $\exists R^-.Self[R' := R' \pm (i, j)] \rightsquigarrow \exists R^-.Self$

Substitutions $[C := C \pm i]$ and $[R' := R' \pm (i,j)]$ do not affect the interpretation of role name R, hence the rules 17 – 22.

23 $\exists R.Self[R := R + (i,j)] \rightsquigarrow (\{i\} \sqcap \{j\}) \sqcup \exists R.Self$

24 $\exists R.Self[R := R - (i,j)] \rightsquigarrow (\neg\{i\} \sqcup \neg\{j\}) \sqcap \exists R.Self$

25 $\exists R^-.Self[R := R + (i,j)] \rightsquigarrow (\{i\} \sqcap \{j\}) \sqcup \exists R^-.Self$

26 $\exists R^-.Self[R := R - (i,j)] \rightsquigarrow (\neg\{i\} \sqcup \neg\{j\}) \sqcap \exists R^-.Self$

$\exists R.Self$ is satisfied by an element, say k, after labeling the edge (i,j) with R if and only if it was already satisfied or $k = i = j$. On the other hand, $\exists R.Self$ is satisfied by an element, say k, after removing the label R from the edge (i,j) if and only if it was already satisfied and either $k \neq i$ or $k \neq j$. The direction of the self-loop being irrelevant, the translations are the same for $\exists R^-.Self$.

27 $\exists R.Self[R' := Q] \rightsquigarrow \exists R.Self$

28 $\exists R^-.Self[R' := Q] \rightsquigarrow \exists R^-.Self$

29 $\exists R.Self[R := Q] \rightsquigarrow \exists Q.Self$

30 $\exists R^-.Self[R := Q] \rightsquigarrow \exists Q^-.Self$

If $R \neq R'$, its valuation is left unmodified. Otherwise, as the valuation of R is now exactly the one of Q, th nodes where R is localy reflexive are thos where Q is. Additionally, as the only possible Q are either atomic roles or the inverse of atomic roles, these formula are well-formed.

31 $\exists R.Self[R' := (i,j)] \rightsquigarrow \exists R.Self$

32 $\exists R^-.Self[R' := (i,j)] \rightsquigarrow \exists R^-.Self$

33 $\exists R.Self[R := (i,j)] \rightsquigarrow \{i\} \sqcap \{j\}$

34 $\exists R^-.Self[R := (i,j)] \rightsquigarrow \{i\} \sqcap \{j\}$

As the valuation of R is now exactly the couple $\{i^{\mathcal{I}}, j^{\mathcal{I}}\}$, the only possibility is that both i and j are the considered node. As previously, local reflexivity does not care for the direction of the edge.

35 $\exists R.Self[i \gg^{in} j] \rightsquigarrow$
$((\{i\} \Leftrightarrow \{j\}) \Rightarrow \exists R.Self) \sqcap (\neg\{i\} \sqcap \{j\} \Rightarrow \exists R.Self \sqcup \exists R.\{i\})$

36 $\exists R^-.Self[i \gg^{in} j] \rightsquigarrow$
$((\{i\} \Leftrightarrow \{j\}) \Rightarrow \exists R.Self) \sqcap (\neg\{i\} \sqcap \{j\} \Rightarrow \exists R.Self \sqcup \exists R.\{i\})$

37 $\exists R.Self[i \gg^{out} j] \rightsquigarrow$
$((\{i\} \Leftrightarrow \{j\}) \Rightarrow \exists R.Self) \sqcap (\neg\{i\} \sqcap \{j\} \Rightarrow \exists R.Self \sqcup \exists R^-.\{i\})$

38 $\exists R^-.Self[i \gg^{out} j] \rightsquigarrow$
$((\{i\} \Leftrightarrow \{j\}) \Rightarrow \exists R.Self) \sqcap (\neg\{i\} \sqcap \{j\} \Rightarrow \exists R.Self \sqcup \exists R^-.\{i\})$

There are essentially three possibilities: either the current node is i and j or neither of them and thus it is not affected, or it is i but not j but then it is impossible to have an incoming (respectively an outgoing edge), or it is j and not i and then R is locally reflexive if it already was or there was an edge from j to i that now goes from j to j.

39 $(\bowtie\ n\ R\ \phi)[C := C \pm i] \rightsquigarrow (\bowtie\ n\ R\ \phi[C := C \pm i])$

40 $(\bowtie\ n\ R^-\ \phi)[C := C \pm i] \rightsquigarrow (\bowtie\ n\ R^-\ \phi[C := C \pm i])$

41 $(\bowtie\ n\ R\ \phi)[C := \psi] \rightsquigarrow (\bowtie\ n\ R\ \phi[C := \psi])$

42 $(\bowtie\ n\ R^-\ \phi)[C := \psi] \rightsquigarrow (\bowtie\ n\ R^-\ \phi[C := \psi])$

43 $(\bowtie\ n\ R\ \phi)[R' := R' \pm (i,j)] \rightsquigarrow (\bowtie\ n\ R\ \phi[R' := R' \pm (i,j)])$

44 $(\bowtie\ n\ R^-\ \phi)[R' := R' \pm (i,j)] \rightsquigarrow (\bowtie\ n\ R^-\ \phi[R' := R' \pm (i,j)])$

Substitutions $[C := C \pm i]$, $[R' := R' \pm (i,j)]$ and $[C := \psi]$ do not modify the interpretation of R, hence the rules 39 – 44.

45 $(\bowtie\ n\ R\ \phi)[R := R + (i,j)] \rightsquigarrow$

$((\{i\} \sqcap \exists U.(\{j\} \sqcap \phi[R := R + (i,j)]) \sqcap \forall R.\neg\{j\}) \Rightarrow$
$(\bowtie\ (n-1)\ R\ \phi[R := R + (i,j)]))$
$\sqcap\ ((\neg\{i\} \sqcup \forall U.(\neg\{j\} \sqcup \neg\phi[R := R + (i,j)]) \sqcup \exists R.\{j\}) \Rightarrow$
$(\bowtie\ n\ R\ \phi[R := R + (i,j)]))$

46 $(\bowtie\ n\ R^-\ \phi)[R := R + (i,j)] \rightsquigarrow$

$((\{j\} \sqcap \exists U.(\{i\} \sqcap \phi[R := R + (i,j)]) \sqcap \forall R^-.\neg\{i\}) \Rightarrow$
$(\bowtie\ (n-1)\ R^-\ \phi[R := R + (i,j)]))$
$\sqcap\ ((\neg\{j\} \sqcup \forall U.(\neg\{i\} \sqcup \neg\phi[R := R + (i,j)]) \sqcup \exists R^-.\{i\}) \Rightarrow$
$(\bowtie\ n\ R^-\ \phi[R := R + (i,j)]))$

47 $(\bowtie\ n\ R\ \phi)[R := R - (i,j)] \rightsquigarrow$

$((\{i\} \sqcap \exists U.(\{j\} \sqcap \phi[R := R - (i,j)]) \sqcap \exists R.\{j\}) \Rightarrow$
$(\bowtie\ (n+1)\ R\ \phi[R := R - (i,j)]))$
$\sqcap\ ((\neg\{i\} \sqcup \forall U.(\neg\{j\} \sqcup \neg\phi[R := R - (i,j)]) \sqcup \forall R.\neg\{j\}) \Rightarrow$
$(\bowtie\ n\ R\ \phi[R := R - (i,j)]))$

48 $(\bowtie\ n\ R^-\ \phi)[R := R - (i,j)] \rightsquigarrow$

$((\{j\} \sqcap \exists U.(\{i\} \sqcap \phi[R := R - (i,j)]) \sqcap \exists R^-.\{i\}) \Rightarrow$
$(\bowtie\ (n+1)\ R^-\ \phi[R := R - (i,j)]))$
$\sqcap\ ((\neg\{j\} \sqcup \forall U.(\neg\{i\} \sqcup \neg\phi[R := R - (i,j)]) \sqcup \forall R^-.\neg\{i\}) \Rightarrow$
$(\bowtie\ n\ R^-\ \phi[R := R - (i,j)]))$

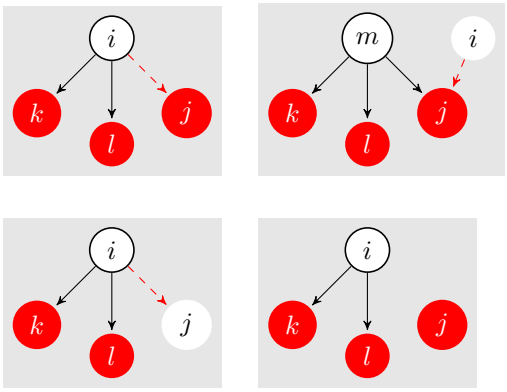


Figure 6.2: Example illustrating rule 47. The nodes satisfying $\phi[R := R+(i,j)]$ are drawn in red, the current node is circled in black. The top left graph shows the case in which the first part of the first implication is true, that is $(\geq 2R.\phi)[R := R-(i,j)]$ is true if $(\geq 3R.\phi[R := R-(i,j)])$ is. The top right graph (resp. bottom left, bottom right)shows that if $\{i\}$ is false (resp. j doesn't satisfy $\phi[R := R-(i,j)]$, (i,j) is not an R-edge), then the modification does not affect the property.

In rules 45–48, we distinguish the cases where a substitution induces a change and those where it does not. The substitution changes the interpretation of the concept under consideration at element k if $k = i$ (resp. $k = j$ when the concept uses R^-), the edge (i,j) does not already exist (resp. already exists) and j (resp. i) satisfies ϕ after the substitution. In that case, exactly one neighbour is added (resp. removed). Else, nothing has changed and thus the concept remains the same (see Figure 6.2). Let us consider rule 47, the interpretation of $(\bowtie\ n\ R\ \phi)$ will be modified by the action $R := R-(i,j)$ if the current node is i, j is labeled with $\phi[R := R-(i,j)]$ and $(i,j) : R$. The condition of the first implication $(\{i\} \sqcap \exists U.(\{j\} \sqcap \phi[R := R-(i,j)]) \sqcap \exists R.\{j\})$ is satisfied if and only if it is the case. In such a situation, the current node will lose a neighbor labeled with ϕ and thus it had to have at least $n+1$ to have n afterwards. Otherwise, the case of the second implication, the number of neighbors is still the same.

49 $(\bowtie\ n\ R\ \phi)[R' := Q] \rightsquigarrow (\bowtie\ n\ R\ \phi[R' := Q])$

50 $(\bowtie\ n\ R^-\ \phi)[R' := Q] \rightsquigarrow (\bowtie\ n\ R^-\ \phi[R' := Q])$

51 $(\bowtie\ n\ R\ \phi)[R := Q] \rightsquigarrow (\bowtie\ n\ Q\ \phi[R := Q])$

52 $(\bowtie\ n\ R^-\ \phi)[R := Q] \rightsquigarrow (\bowtie\ n\ Q^-\ \phi[R := Q])$

If $R \neq R'$, its valuation is left unmodified. Otherwise, if the valuation of R becomes equal to the one of Q, the number of R-neighbours is the one of Q-neighbours.

53 $(\bowtie\ n\ R\ \phi)[R' := (i,j)] \rightsquigarrow (\bowtie\ n\ R\ \phi[R' := (i,j)])$

54 For $n = 1$, $(<\ n\ R\ \phi)[R := (i,j)] \rightsquigarrow \neg\{i\} \sqcup \forall U.(\neg\{j\} \sqcup \neg\phi[R := (i,j)])$

55 For $n \geq 2$, $(<\ n\ R\ \phi)[R := (i,j)] \rightsquigarrow \top$

56 For $n = 1$, $(\geq\ n\ R\ \phi)[R := (i,j)] \rightsquigarrow \{i\} \sqcap \exists U.(\{j\} \sqcap \phi[R := (i,j)])$

57 For $n \geq 2$, $(\geq\ n\ R\ \phi)[R := (i,j)] \rightsquigarrow \bot$

58 For $n = 1$, $(<\ n\ R^-\ \phi)[R := (i,j)] \rightsquigarrow \neg\{j\} \sqcup \forall U.(\neg\{i\} \sqcup \neg\phi[R := (i,j)])$

59 For $n \geq 2$, $(<\ n\ R^-\ \phi)[R := (i,j)] \rightsquigarrow \top$

60 For $n = 1$, $(\geq\ n\ R^-\ \phi)[R := (i,j)] \rightsquigarrow \{j\} \sqcap \exists U.(\{i\} \sqcap \phi[R := (i,j)])$

61 For $n \geq 2$, $(\geq\ n\ R^-\ \phi)[R := (i,j)] \rightsquigarrow \bot$

If $R \neq R'$, its valuation is left unmodified. Otherwise, as the valuation of R becomes $\{(i^{\mathcal{I}}, j^{\mathcal{I}})\}$, only i can have an R-neighbour (respectively j an R^--neihbour) satisfying ϕ and it can be only j (respectively i).

62 $(\bowtie\ n\ R\ \phi)[i \gg^{in} j] \rightsquigarrow (\exists U.(\{i\} \sqcap \{j\}) \Rightarrow (\bowtie\ n\ R\ \phi[i \gg^{in} j]))$
$\sqcap(\exists U.(\{i\} \sqcap \neg\{j\}) \Rightarrow$
$(\exists R.(\{i\} \sqcap \phi[i \gg^{in} j]) \sqcap \forall R.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \neg\phi[i \gg^{in} j]) \Rightarrow$
$(\bowtie\ (n+1)\ R\ \phi[i \gg^{in} j]))$
$\sqcap(\exists R.(\{i\} \sqcap \neg\phi[i \gg^{in} j]) \sqcap \forall R.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \phi[i \gg^{in} j]) \Rightarrow$
$(\bowtie\ (n-1)\ R\ \phi[i \gg^{in} j]))$
$\sqcap(\exists R.(\{i\} \sqcap \phi[i \gg^{in} j]) \sqcap \exists R.\{j\} \Rightarrow$
$(\bowtie\ (n+1)\ R\ \phi[i \gg^{in} j]))$
$\sqcap((\forall R.\neg\{i\})$
$\sqcup(\exists R.(\{i\} \sqcap \neg\phi[i \gg^{in} j]) \sqcap \exists R.\{j\})$
$\sqcup(\exists R.(\{i\} \sqcap \phi[i \gg^{in} j]) \sqcap \forall R.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \phi[i \gg^{in} j]))$
$\sqcup(\exists R.(\{i\} \sqcap \neg\phi[i \gg^{in} j]) \sqcap \forall R.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \neg\phi[i \gg^{in} j])) \Rightarrow$
$(\bowtie\ n\ R\ \phi[i \gg^{in} j])))$

63 $(<\ n\ R^-\ \phi)[i \gg^{in} j] \rightsquigarrow (\{i\} \sqcap \neg\{j\}) \sqcup$
$(\neg\{i\} \sqcap \{j\} \Rightarrow$
$\bigsqcup_{k \in [0,n]}(<\ k\ R^-\ \phi[i \gg^{in} j]) \sqcap$
$\exists U.(\{i\} \sqcap (<\ (n-k)\ R^-\ (\phi[i \gg^{in} j] \sqcap \neg\exists R^-.\{j\}))))$
$\sqcup((\{i\} \Leftrightarrow \{j\}) \Rightarrow (<\ n\ R^-\ \phi[i \gg^{in} j]))$

64 $(\geq\ n\ R^-\ \phi)[i \gg^{in} j] \rightsquigarrow (\neg\{i\} \sqcap \{j\} \Rightarrow$
$\bigsqcup_{k \in [0,n]}(\geq\ k\ R^-\ \phi[i \gg^{in} j]) \sqcap$
$\exists U.(\{i\} \sqcap (\geq\ (n-k)\ R^-\ (\phi[i \gg^{in} j] \sqcap \exists R^-.\{j\}))))$
$\sqcap((\{i\} \Leftrightarrow \{j\}) \Rightarrow (\geq\ n\ R^-\ \phi[i \gg^{in} j]))$

65 $(<\ n\ R\ \phi)[i \gg^{out} j] \rightsquigarrow (\{i\} \sqcap \neg\{j\}) \sqcup$
$(\neg\{i\} \sqcap \{j\} \Rightarrow$
$\bigsqcup_{k \in [0,n]}(<\ k\ R\ \phi[i \gg^{out} j]) \sqcap$

$$\exists U.(\{i\} \sqcap (< \ (n-k) \ R \ (\phi[i \gg^{out} j] \sqcap \neg \exists R.\{j\}))))$$
$$\sqcup((\{i\} \Leftrightarrow \{j\}) \Rightarrow (< \ n \ R \ \phi[i \gg^{out} j]))$$

66 $(\geq \ n \ R \ \phi)[i \gg^{out} j] \rightsquigarrow (\neg\{i\} \sqcap \{j\} \Rightarrow$
$$\bigsqcup_{k \in [0,n]} (\geq \ k \ R \ \phi[i \gg^{out} j]) \sqcap$$
$$\exists U.(\{i\} \sqcap (\geq \ (n-k) \ R \ (\phi[i \gg^{out} j] \sqcap \exists R.\{j\}))))$$
$$\sqcap((\{i\} \Leftrightarrow \{j\}) \Rightarrow (\geq \ n \ R \ \phi[i \gg^{out} j]))$$

67 $(\bowtie \ n \ R^- \ \phi)[i \gg^{out} j] \rightsquigarrow (\exists U.(\{i\} \sqcap \{j\}) \Rightarrow (\bowtie \ n \ R^- \ \phi[i \gg^{out} j]))$
$$\sqcap(\exists U.(\{i\} \sqcap \neg\{j\}) \Rightarrow$$
$$(\exists R^-.(\{i\} \sqcap \phi[i \gg^{out} j]) \sqcap \forall R^-.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \neg\phi[i \gg^{out} j]) \Rightarrow$$
$$(\bowtie \ (n+1) \ R^- \ \phi[i \gg^{out} j]))$$
$$\sqcap(\exists R^-.(\{i\} \sqcap \neg\phi[i \gg^{out} j]) \sqcap \forall R^-.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \phi[i \gg^{out} j]) \Rightarrow$$
$$(\bowtie \ (n-1) \ R^- \ \phi[i \gg^{out} j]))$$
$$\sqcap(\exists R^-.(\{i\} \sqcap \phi[i \gg^{out} j]) \sqcap \exists R.\{j\} \Rightarrow$$
$$(\bowtie \ (n+1) \ R^- \ \phi[i \gg^{out} j]))$$
$$\sqcap((\forall R^-.\neg\{i\})$$
$$\sqcup(\exists R^-.(\{i\} \sqcap \neg\phi[i \gg^{out} j]) \sqcap \exists R^-.\{j\})$$
$$\sqcup(\exists R^-.(\{i\} \sqcap \phi[i \gg^{out} j]) \sqcap \forall R^-.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \phi[i \gg^{out} j]))$$
$$\sqcup(\exists R^-.(\{i\} \sqcap \neg\phi[i \gg^{out} j]) \sqcap \forall R^-.\neg\{j\}$$
$$\sqcap \exists U.(\{j\} \sqcap \neg\phi[i \gg^{out} j])) \Rightarrow$$
$$(\bowtie \ n \ R^- \ \phi[i \gg^{out} j])))$$

These are the trickiest of the rules as those substitution affect multiple edges at the same time possibly changing massively the interpretations. In all cases, if $i = j$, the action actually does nothing.

Let us consider the other different possibilities for $(\bowtie \ n \ R \ \phi)[i \gg^{in} j]$ that are illustrated in Figure 6.3:

- There exists an R-edge toward i, i satisfies $\phi[i \gg^{in} j]$, there is no R-edge toward j and j does not satisfy $\phi[i \gg^{in} j]$: in that case, the current node will have exactly one less R-neighbour satisfying $\phi[i \gg^{in} j]$.

- There exists an R-edge toward i, i does not satisfy $\phi[i \gg^{in} j]$, there is no R-edge toward j and j does not satisfy $\phi[i \gg^{in} j]$: in that case, the current node will have exactly one more R-neighbour satisfying $\phi[i \gg^{in} j]$.

- There exists an R-edge toward i, i satisfies $\phi[i \gg^{in} j]$ and there is an R-edge toward j: in that case, the current node will gave exactly one less R-neighbour satisfying $\phi[i \gg^{in} j]$.

- In all other cases, the current node is left unaffected.

Let us now look at the rules 62 and 63:

- If the current node is i and not j, it now as no incoming R-edge and thus satisfies $(< \ m \ P^- \ \psi)$ and none of the $(\geq \ m \ P^- \ \psi)$ for any $m > 0$, P or ψ

- If the current node is j and not i, it now as all its previous incoming edges plus those of i. The sum of those must thus be compared to n with special care taken not to count twice those that were R-neighbours of both
- If the current node is both i and j or neither, it is not directly affected by the action.

68 $(\exists R.\phi)[C := C \pm i] \rightsquigarrow \exists R.(\phi[C := C \pm i])$

69 $(\exists R^-.\phi)[C := C \pm i] \rightsquigarrow \exists R^-.(\phi[C := C \pm i])$

70 $(\exists R.\phi)[C := \psi] \rightsquigarrow \exists R.(\phi[C := \psi])$

71 $(\exists R^-.\phi)[C := \psi] \rightsquigarrow \exists R^-.(\phi[C := \psi])$

72 $(\exists R.\phi)[R' := R' \pm (i,j)] \rightsquigarrow \exists R.(\phi[R' := R' \pm (i,j)])$

73 $(\exists R^-.\phi)[R' := R' \pm (i,j)] \rightsquigarrow \exists R^-.(\phi[R' := R' \pm (i,j)])$

Substitutions $[C := C \pm i]$, $[C := \psi]$ and $[R' := R' \pm (i,j)]$ do not modify the interpretation of role name R, hence the rules 68 – 73.

74 $(\exists R.\phi)[R := R + (i,j)] \rightsquigarrow (\{i\} \Rightarrow$
$\exists U.(\{j\} \sqcap \phi[R := R + (i,j)]) \sqcup \exists R.\phi[R := R + (i,j)])$
$\sqcap(\neg\{i\} \Rightarrow \exists R.(\phi[R := R + (i,j)]))$

75 $(\exists R^-.\phi)[R := R + (i,j)] \rightsquigarrow (\{j\} \Rightarrow$
$\exists U.(\{i\} \sqcap \phi[R := R + (i,j)]) \sqcup \exists R^-.\phi[R := R + (i,j)])$
$\sqcap(\neg\{j\} \Rightarrow \exists R^-.(\phi[R := R + (i,j)]))$

76 $(\exists R.\phi)[R := R - (i,j)] \rightsquigarrow$
$(\{i\} \Rightarrow \exists R.(\phi[R := R - (i,j)] \sqcap \neg\{j\}))$
$\sqcap(\neg\{i\} \Rightarrow \exists R.(\phi[R := R - (i,j)]))$

77 $(\exists R^-.\phi)[R := R - (i,j)] \rightsquigarrow$
$(\{j\} \Rightarrow \exists R^-.(\phi[R := R - (i,j)] \sqcap \neg\{i\}))$
$\sqcap(\neg\{j\} \Rightarrow \exists R^-.(\phi[R := R - (i,j)]))$

Rules 74–77 are quite similar to 49–53 where $\bowtie$ is replaced by $\geq$ and $n = 1$. Let us consider the rule 74. If the current node, say k is not i, the interpretation of $\exists R.\phi$ is only affected if the interpretation of ϕ is. This is the second implication. Otherwise, $k = i$ and then either j is labeled with ϕ after the substitution and thus there exists one neighbor satisfying ϕ even if there was none before or there must exist a current neighbor that will satisfy ϕ.

78 $(\exists R.\phi)[R' := Q] \rightsquigarrow \exists R.(\phi[R' := Q])$

79 $(\exists R^-.\phi)[R' := Q] \rightsquigarrow \exists R^-.(\phi[R' := Q])$

80 $(\exists R.\phi)[R := Q] \rightsquigarrow \exists Q.(\phi[R := Q])$

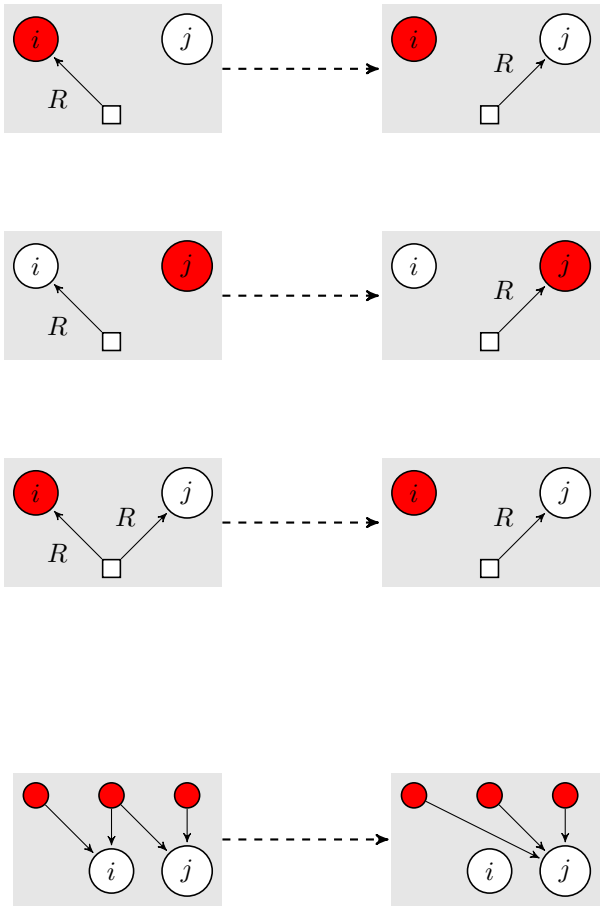


Figure 6.3: Example illustrating rule 62 to 67. The nodes satisfying ϕ are filled in red, the current node is a rectangle. The substitution occurs when going from left to right. The first three transformation correspond to rule 62 and illustrate the cases when interpretations are altered. The last one correspond to the rules 63 and 64.

81 $(\exists R^{-}.\phi)[R := Q] \rightsquigarrow \exists Q^{-}.(\phi[R := Q])$

If $R \neq R'$, its valuation is left unmodified. Otherwise, as the valuation of R becomes equal to the one of Q, there will be an R-neighbour satisfying ϕ if there is such a Q-neighbour.

82 $(\exists R.\phi)[R' := (i,j)] \rightsquigarrow \exists R.(\phi[R' := (i,j)])$

83 $(\exists R^{-}.\phi)[R' := (i,j)] \rightsquigarrow \exists R^{-}.(\phi[R' := (i,j)])$

84 $(\exists R.\phi)[R := (i,j)] \rightsquigarrow \{i\} \sqcap \exists U.(\{j\} \sqcap \phi[R := (i,j)])$

85 $(\exists R^{-}.\phi)[R := (i,j)] \rightsquigarrow \{j\} \sqcap \exists U.(\{i\} \sqcap \phi[R := (i,j)])$

As the valuation of R becomes equal to $\{(i^{\mathcal{I}}, j^{\mathcal{I}})\}$, the only node that can have an R-neighbour (resp. an R^{-}-neighbour) is i (resp. j) and this only in case j (resp. i) satisfies ϕ.

86 $(\exists R.\phi)[i \gg^{in} j)] \rightsquigarrow (\exists U.(\{i\} \sqcap \{j\}) \Rightarrow \exists R.\phi[i \gg^{in} j])$
$\sqcap(\exists U.(\{i\} \sqcap \neg\{j\}) \Rightarrow$
$(\exists R.(\{i\} \sqcap \phi[i \gg^{in} j]) \sqcap \forall R.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \neg\phi[i \gg^{in} j]) \Rightarrow$
$\exists R.(\phi[i \gg^{in} j] \sqcap \neg\{i\}))$
$\sqcap(\exists R.(\{i\} \sqcap \neg\phi[i \gg^{in} j]) \sqcap \forall R.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \phi[i \gg^{in} j]))$
$\sqcap(\exists R.(\{i\} \sqcap \phi[i \gg^{in} j]) \sqcap \exists R.\{j\} \Rightarrow$
$\exists R.(\phi[i \gg^{in} j] \sqcap \neg\{i\}))$
$\sqcap((\forall R.\neg\{i\})$
$\sqcup(\exists R.(\{i\} \sqcap \neg\phi[i \gg^{in} j]) \sqcap \exists R.\{j\})$
$\sqcup(\exists R.(\{i\} \sqcap \phi[i \gg^{in} j]) \sqcap \forall R.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \phi[i \gg^{in} j]))$
$\sqcup(\exists R.(\{i\} \sqcap \neg\phi[i \gg^{in} j]) \sqcap \forall R.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \neg\phi[i \gg^{in} j])) \Rightarrow$
$\exists R.\phi[i \gg^{in} j]))$

87 $(\exists R^{-}.\phi)[i \gg^{in} j] \rightsquigarrow (\neg\{i\} \sqcap \{j\} \Rightarrow$
$(\exists R^{-}.\phi[i \gg^{in} j] \sqcup \exists U.(\{i\} \sqcap \exists R^{-}.\phi[i \gg^{in} j])))$
$\sqcap((\{i\} \Leftrightarrow \{j\}) \Rightarrow$
$\exists R^{-}.\phi[i \gg^{in} j])$

88 $(\exists R.\phi)[i \gg^{out} j] \rightsquigarrow (\neg\{i\} \sqcap \{j\} \Rightarrow$
$\exists R.\phi[i \gg^{out} j] \sqcup \exists U.(\{i\} \sqcap \exists R.\phi[i \gg^{out} j]))$
$\sqcap((\{i\} \Leftrightarrow \{j\}) \Rightarrow$
$\exists R.\phi[i \gg^{out} j])$

89 $(\exists R^{-}.\phi)[i \gg^{out} j] \rightsquigarrow (\exists U.(\{i\} \sqcap \{j\}) \Rightarrow \exists R^{-}.\phi[i \gg^{in} j])$
$\sqcap(\exists U.(\{i\} \sqcap \neg\{j\}) \Rightarrow$
$(\exists R^{-}.(\{i\} \sqcap \phi[i \gg^{out} j]) \sqcap \forall R^{-}.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \neg\phi[i \gg^{out} j]) \Rightarrow$
$\exists R^{-}.(\phi[i \gg^{out} j] \sqcap \neg\{j\}))$
$\sqcap(\exists R^{-}.(\{i\} \sqcap \neg\phi[i \gg^{out} j]) \sqcap \forall R^{-}.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \phi[i \gg^{out} j]))$
$\sqcap(\exists R^{-}.(\{i\} \sqcap \phi[i \gg^{out} j]) \sqcap \exists R.\{j\} \Rightarrow$
$\exists R^{-}.(\phi[i \gg^{out} j] \sqcap \neg\{i\}))$
$\sqcap((\forall R^{-}.\neg\{i\})$

$$\sqcup(\exists R^-.(\{i\} \sqcap \neg\phi[i \gg^{out} j]) \sqcap \exists R^-.\{j\})$$
$$\sqcup(\exists R^-.(\{i\} \sqcap \phi[i \gg^{out} j]) \sqcap \forall R^-.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \phi[i \gg^{out} j]))$$
$$\sqcup(\exists R^-.(\{i\} \sqcap \neg\phi[i \gg^{out} j]) \sqcap \forall R^-.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \neg\phi[i \gg^{out} j])) \Rightarrow \exists R^-.\phi[i \gg^{out} j]))$$

Rules 86–89 are quite similar to 62–67 where $\bowtie$ is replaced by $\geq$ and $n = 1$. One also has to replace the fact that there must exist 2 different R- or R^--neighbours by the fact that there must be one that is not i.

90 $(\forall R.\phi)[C := C \pm i] \rightsquigarrow \forall R.(\phi[C := C \pm i])$

91 $(\forall R^-.\phi)[C := C \pm i] \rightsquigarrow \forall R^-.(\phi[C := C \pm i])$

92 $(\forall R.\phi)[C := \psi] \rightsquigarrow \forall R.(\phi[C := D])$

93 $(\forall R^-.\phi)[C := \psi] \rightsquigarrow \forall R^-.(\phi[C := D])$

94 $(\forall R.\phi)[R' := R' \pm (i,j)] \rightsquigarrow \forall R.(\phi[R' := R' \pm (i,j)])$

95 $(\forall R^-.\phi)[R' := R' \pm (i,j)] \rightsquigarrow \forall R^-.(\phi[R' := R' \pm (i,j)])$

Substitutions $[C := C \pm i]$, $[C := \psi]$ and $[R' := R' \pm (i,j)]$ do not modify the interpretation of role name R, hence the rules 90 – 95.

96 $(\forall R.\phi)[R := R + (i,j)] \rightsquigarrow (\{i\} \Rightarrow \forall R.(\phi[R := R + (i,j)]) \sqcap \exists U.(\{j\} \sqcap \phi[R := R + (i,j)])) \sqcap (\neg\{i\} \Rightarrow \forall R.(\phi[R := R + (i,j)]))$

97 $(\forall R^-.\phi)[R := R + (i,j)] \rightsquigarrow (\{j\} \Rightarrow \forall R^-.(\phi[R := R + (i,j)]) \sqcap \exists U.(\{i\} \sqcap \phi[R := R + (i,j)])) \sqcap (\neg\{j\} \Rightarrow \forall R^-.(\phi[R := R + (i,j)]))$

98 $(\forall R.\phi)[R := R - (i,j)] \rightsquigarrow (\{i\} \Rightarrow \forall R.(\phi[R := R - (i,j)] \sqcup \{j\})) \sqcap (\neg\{i\} \Rightarrow \forall R.(\phi[R := R - (i,j)]))$

99 $(\forall R^-.\phi)[R := R - (i,j)] \rightsquigarrow (\{j\} \Rightarrow \forall R^-.(\phi[R := R - (i,j)] \sqcup \{i\})) \sqcap (\neg\{j\} \Rightarrow \forall R^-.(\phi[R := R - (i,j)]))$

See Figure 6.4 for an illustration. Rules 96–99 are quite similar to 49–52 where $\bowtie$ is replaced by $<$ and $n = 1$. Let us consider rule 98. Once more there are two possibilities: either the current node is i or it is not. In the former case, all the neighbors of the current state will be labeled with ϕ after performing $R := R - (i,j)$ if and only if the only neighbor that existed before that would not satisfy ϕ after the transformation was j. In the later case, the neighbors are the same and thus one has to check that they will all be labeled with ϕ after the transformation.

100 $(\forall R.\phi)[R' := Q] \rightsquigarrow \forall R.(\phi[R' := Q])$

101 $(\forall R^-.\phi)[R' := Q] \rightsquigarrow \forall R^-.(\phi[R' := Q])$

102 $(\forall R.\phi)[R := Q] \rightsquigarrow \forall Q.(\phi[R := Q])$

103 $(\forall R^-.\phi)[R := Q] \rightsquigarrow \forall Q^-.(\phi[R := Q])$

If $R \neq R'$, its valuation is left unmodified. Otherwise, as the valuation of R becomes equal to the one of Q, all R-neighbours satisfy ϕ if all Q-neighbours do.

104 $(\forall R.\phi)[R' := (i,j)] \rightsquigarrow \forall R.(\phi[R' := (i,j)])$

105 $(\forall R^-.\phi)[R' := (i,j)] \rightsquigarrow \forall R^-.(\phi[R' := (i,j)])$

106 $(\forall R.\phi)[R := (i,j)] \rightsquigarrow (\{i\} \sqcap \exists U.(\{j\} \sqcap \phi[R := (i,j)])) \sqcup \neg\{i\}$

107 $(\forall R^-.\phi)[R := (i,j)] \rightsquigarrow (\{j\} \sqcap \exists U.(\{i\} \sqcap \phi[R := (i,j)])) \sqcup \neg\{j\}$

As the valuation of R becomes equal to $\{(i^{\mathcal{I}}, j^{\mathcal{I}})\}$, i (resp. j) is the only node that has an R-neighbour (resp. an R^--neighbour). Thus the concepts are satisfied if and only if the current node is i (resp. j) and j (resp. i) satisfies ϕ or if the current node is not i (resp. j) .

108 $(\forall R.\phi)[i \gg^{in} j] \rightsquigarrow (\exists U.(\{i\} \sqcap \{j\}) \Rightarrow \forall R.\phi[i \gg^{in} j])$
$\sqcap(\exists U.(\{i\} \sqcap \neg\{j\}) \Rightarrow$
$(\exists R.(\{i\} \sqcap \phi[i \gg^{in} j]) \sqcap \forall R.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \neg\phi[i \gg^{in} j]) \Rightarrow$
$\forall R.(\phi[i \gg^{in} j] \sqcup \{i\}))$
$\sqcap(\exists R.(\{i\} \sqcap \phi[i \gg^{in} j]) \sqcap \exists R.\{j\} \Rightarrow$
$\forall R.(\phi[i \gg^{in} j] \sqcup \{i\}))$
$\sqcap((\forall R.\neg\{i\})$
$\sqcup(\exists R.(\{i\} \sqcap \neg\phi[i \gg^{in} j]) \sqcap \exists R.\{j\})$
$\sqcup(\exists R.(\{i\} \sqcap \phi[i \gg^{in} j]) \sqcap \forall R.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \phi[i \gg^{in} j])) \Rightarrow$
$\forall R.\phi[i \gg^{in} j]))$

109 $(\forall R^-.\phi)[i \gg^{in} j] \rightsquigarrow (\{i\} \sqcap \neg\{j\})$
$\sqcup(\neg\{i\} \sqcap \{j\} \Rightarrow \forall R^-.\phi[i \gg^{in} j] \sqcap \exists U.(\{i\} \sqcap \forall R^-.\phi[i \gg^{in} j]))$
$\sqcup((\{i\} \Leftrightarrow \{j\}) \Rightarrow \forall R^-.\phi[i \gg^{in} j])$

110 $(\forall R.\phi)[i \gg^{out} j] \rightsquigarrow (\{i\} \sqcap \neg\{j\})$
$\sqcup(\neg\{i\} \sqcap \{j\} \Rightarrow \forall R.\phi[i \gg^{out} j] \sqcap \exists U.(\{i\} \sqcap \forall R.\phi[i \gg^{out} j]))$
$\sqcup((\{i\} \Leftrightarrow \{j\}) \Rightarrow \forall R.\phi[i \gg^{out} j])$

111 $(\forall R^-.\phi)[i \gg^{out} j] \rightsquigarrow (\exists U.(\{i\} \sqcap \{j\}) \Rightarrow \forall R^-.\phi[i \gg^{out} j])$
$\sqcap(\exists U.(\{i\} \sqcap \neg\{j\}) \Rightarrow$
$(\exists R^-.(\{i\} \sqcap \neg\phi[i \gg^{out} j]) \sqcap \forall R^-.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \phi[i \gg^{out} j])$
$\Rightarrow \forall R^-.(\phi[i \gg^{out} j] \sqcup \{i\}))$
$\sqcap(\exists R^-.(\{i\} \sqcap \neg\phi[i \gg^{out} j]) \sqcap \exists R.\{j\} \Rightarrow$
$\forall R^-.(\phi[i \gg^{out} j] \sqcup \{i\}))$

$$\sqcap((\forall R^-.\neg\{i\}) \\ \sqcup(\exists R^-.(\{i\} \sqcap \phi[i \gg^{out} j]) \sqcap \exists R^-.\{j\}) \\ \sqcup(\exists R^-.(\{i\} \sqcap \neg\phi[i \gg^{out} j]) \sqcap \forall R^-.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \phi[i \gg^{out} j])) \Rightarrow \\ \forall R^-.\phi[i \gg^{out} j]))$$

Rules 108–111 are quite similar to 62–67 where $\bowtie$ is replaced by $\geq$ and $n = 1$. One also has to replace the fact that there must be at most 1 R- or R^--neighbours not satisfying ϕ by the fact that the only neighbour that could not satisfy ϕ is i.

Even though the previous intuitions may be helpful in understanding why the rules are that way, they do not constitute proofs that $\mathcal{T}$ is correct. In order to prove that it is, the following two lemmata are introduced. They state three properties of the system $\mathcal{T}$, namely (1) that the interpretations are conserved, that is $\mathcal{T}$ does what one might expect it to do, (2) that it terminates, (3) that the result is substitution free. The proofs of these lemmata are not complicated. Those that want to skip them can go to Section 6.3.

Lemma 1. *Let Σ be a signature, $\mathcal{I}$ an interpretation over Σ, $L \rightsquigarrow R$ one of the rewrite rules of $\mathcal{T}$, then $L^\mathcal{I} = R^\mathcal{I}$.*

Proof. **1** $(\bot\ \theta)^\mathcal{I} = \emptyset = \bot^\mathcal{I}$ as it does not depend on the valuation of any concept or role.

2 As the valuation of a nominal is independent of the valuations of all roles and concepts, $(\{i\}\ \theta)^\mathcal{I} = i^\mathcal{I}$.

3 $(C[R := R \pm (i,j)])^\mathcal{I} = c^\mathcal{I}$ as $c^\mathcal{I}$ does not depend on the valuation of R.

4 $(C[C' := C' \pm i])^\mathcal{I} = C^\mathcal{I}$ as $C^\mathcal{I}$ does not depend on the valuation of C'.

5 $(C[R := Q])^\mathcal{I} = C^\mathcal{I}$. By definition of the valuation of a substitution, $(C[R := Q])^\mathcal{I} = C^\mathcal{I}$.

6 $(C[R := (i,j)])^\mathcal{I} = C^\mathcal{I}$. By definition of the valuation of a substitution, $(C[R := (i,j)])^\mathcal{I} = C^\mathcal{I}$.

7 $(C[i \gg^{in} j])^\mathcal{I} = C^\mathcal{I}$. By definition of the valuation of a substitution, $(C[i \gg^{in} j])^\mathcal{I} = C^\mathcal{I}$.

8 $(C[i \gg^{out} j])^\mathcal{I} = C^\mathcal{I})^\mathcal{I}$. By definition of the valuation of a substitution, $(C[i \gg^{out} j])^\mathcal{I} = C^\mathcal{I}$.

9 $(C[C := C+i])^\mathcal{I} = (C \sqcup \{i\})^\mathcal{I}$. By definition of the valuation of a substitution, $(C[C := C + i])^\mathcal{I} = C^\mathcal{I} \cup \{i^\mathcal{I}\} = (C \sqcup \{i\})^\mathcal{I}$.

10 $(C[C := C - i])^\mathcal{I} = (C \sqcap \neg\{i\})^\mathcal{I}$. By definition of the valuation of a substitution, $(C[C := C - i])^\mathcal{I} = C^\mathcal{I} \backslash \{i^\mathcal{I}\} = (C \sqcap \neg\{i\})^\mathcal{I}$.

11 $(C[C' := \psi])^\mathcal{I} =$. By definition of the valuation of a substitution, $(C[C' := \psi])^\mathcal{I} = C^\mathcal{I}$.

12 $(C[C := \psi])^{\mathcal{I}} =$. By definition of the valuation of a substitution, $(C[C := \psi])^{\mathcal{I}} = \psi^{\mathcal{I}}$.

13 By definition, $((\neg\phi)[C := C+i])^{\mathcal{I}} = (\neg\phi)^{\mathcal{I}}$ where $C^{\mathcal{I}}$ is replaced by $C^{\mathcal{I}} \cup \{i^{\backslash}\}$. As $(\neg\phi)^{\mathcal{I}} = \Delta \backslash \phi^{\mathcal{I}}$, $((\neg\phi)[C := C+i])^{\mathcal{I}} = \Delta \backslash \phi'^{\mathcal{I}}$ with $\phi'^{\mathcal{I}} = \phi^{\mathcal{I}}$ where $C^{\mathcal{I}}$ is replaced by $C^{\mathcal{I}} \cup \{i\}$. This is exactly the definition of $(\neg(\phi[C := C+i]))^{\mathcal{I}}$. The same can be done with the other substitutions.

14 By definition, $((\phi \sqcup \psi)[C := C + i])^{\mathcal{I}} = (\phi \sqcup \psi)^{\mathcal{I}}$ where $C^{\mathcal{I}}$ is replaced by $C^{\mathcal{I}} \cup \{i\}$. That is $(\phi' \sqcup \psi')^{\mathcal{I}}$ with $\phi'^{\mathcal{I}} = \phi^{\mathcal{I}}$ where $C^{\mathcal{I}}$ is replaced by $C^{\mathcal{I}} \cup \{i\}$ and $\psi'^{\mathcal{I}} = \psi^{\mathcal{I}}$ where $C^{\mathcal{I}}$ is replaced by $C^{\mathcal{I}} \cup \{i\}$. This is exactly the definition of $(\phi[C := C + i] \sqcup \psi[C := C + i])^{\mathcal{I}}$. The same can be done with the other substitutions.

15 By definition, $((\phi \sqcap \psi)[C := C + i])^{\mathcal{I}} = (\phi \sqcap \psi)^{\mathcal{I}}$ where $C^{\mathcal{I}}$ is replaced by $C^{\mathcal{I}} \cup \{i\}$. That is $(\phi' \sqcap \psi')^{\mathcal{I}}$ with $\phi'^{\mathcal{I}} = \phi^{\mathcal{I}}$ where $C^{\mathcal{I}}$ is replaced by $C^{\mathcal{I}} \cup \{i\}$ and $\psi'^{\mathcal{I}} = \psi^{\mathcal{I}}$ where $C^{\mathcal{I}}$ is replaced by $C^{\mathcal{I}} \cup \{i\}$. This is exactly the definition of $(\phi[C := C + i] \sqcap \psi[C := C + i])^{\mathcal{I}}$. The same can be done with the other substitutions.

16 By definition, $((@_a\phi)[C := C + i])^{\mathcal{I}} = (@_a\phi)^{\mathcal{I}}$ where $C^{\mathcal{I}}$ is replaced by $C^{\mathcal{I}} \cup \{i\}$. This is exactly the definition of $(@_i(\phi[C := C+i]))^{\mathcal{I}}$. The same can be done with the other substitutions.

17 $(\exists R.Self[C := C \pm i])^{\mathcal{I}} = (\exists R.Self)^{\mathcal{I}}$ as $(\exists R.Self)^{\mathcal{I}}$ is independent of the valuation of C.

18 $(\exists R^-.Self[C := C \pm i])^{\mathcal{I}} = (\exists R^-.Self)^{\mathcal{I}}$ as $(\exists R^-.Self)^{\mathcal{I}}$ is independent of the valuation of C.

19 $(\exists R.Self[C := \psi])^{\mathcal{I}} = (\exists R.Self)^{\mathcal{I}}$ as $(\exists R.Self)^{\mathcal{I}}$ is independent of the valuation of C.

20 $(\exists R^-.Self[C := \psi])^{\mathcal{I}} = (\exists R^-.Self)^{\mathcal{I}}$ as $(\exists R^-.Self)^{\mathcal{I}}$ is independent of the valuation of C.

21 $(\exists R.Self[R' := R' \pm (i,j)])^{\mathcal{I}} = (\exists R.Self)^{\mathcal{I}}$ as $(\exists R.Self)^{\mathcal{I}}$ is independent of the valuation of R'.

22 $(\exists R^-.Self[R' := R' \pm (i,j)])^{\mathcal{I}} = (\exists R^-.Self)^{\mathcal{I}}$ as $(\exists R^-.Self)^{\mathcal{I}}$ is independent of the valuation of R'.

23 As $(\exists R.Self)^{\mathcal{I}} = \{x | (x,x) \in R^{\mathcal{I}}\}$, $(\exists R.Self[R := R + (i,j)])^{\mathcal{I}} = \{x | (x,x) \in R^{\mathcal{I}} \cup \{(i,j)\}\} = (\exists R.Self)^{\mathcal{I}} \cup (\{i\} \sqcap \{j\})$ that is $(\exists R.Self[R := R + (i,j)])^{\mathcal{I}} = ((\{i\} \sqcap \{j\}) \sqcup \exists R.Self)^{\mathcal{I}}$.

24 As $(\exists R^-.Self)^{\mathcal{I}} = \{x | (x,x) \in R^{\mathcal{I}}\}$, $(\exists R^-.Self[R := R + (i,j)])^{\mathcal{I}} = \{x | (x,x) \in R^{\mathcal{I}} \cup \{(i,j)\}\} = (\exists R^-.Self)^{\mathcal{I}} \cup (\{i\} \sqcap \{j\})$ that is $(\exists R^-.Self[R := R + (i,j)])^{\mathcal{I}} = ((\{i\} \sqcap \{j\}) \sqcup \exists R^-.Self)^{\mathcal{I}}$.

25 As $(\exists R.Self)^{\mathcal{I}} = \{x|(x,x) \in R^{\mathcal{I}}\}$, $(\exists R.Self[R := R - (i,j)])^{\mathcal{I}} = \{x|(x,x) \in R^{\mathcal{I}} \backslash \{(i,j)\}\} = (\exists R.Self)^{\mathcal{I}} \backslash (\{i\} \cap \{j\})$ that is $(\exists R.Self[R := R - (i,j)])^{\mathcal{I}} = ((\neg\{i\} \sqcup \neg\{j\}) \sqcap \exists R.Self)^{\mathcal{I}}$.

26 As $(\exists R^-.Self)^{\mathcal{I}} = \{x|(x,x) \in R^{\mathcal{I}}\}$, $(\exists R^-.Self[R := R - (i,j)])^{\mathcal{I}} = \{x|(x,x) \in R^{\mathcal{I}} \cap \overline{\{(i,j)\}}\} = (\exists R^-.Self)^{\mathcal{I}} \backslash (\{i\} \cap \{j\})$ that is $(\exists R^-.Self[R := R - (i,j)])^{\mathcal{I}} = ((\neg\{i\} \sqcup \neg\{j\}) \sqcap \exists R^-.Self)^{\mathcal{I}}$.

27 As the valuation of R is left unmodified by $R' := Q$, $(\exists R.Self[R' := Q])^{\mathcal{I}} = (\exists R.Self)^{\mathcal{I}}$.

28 As the valuation of R is left unmodified by $R' := Q$, $(\exists R^-.Self[R' := Q])^{\mathcal{I}} = (\exists R^-.Self)^{\mathcal{I}}$.

29 $(\exists R.Self)^{\mathcal{I}} = \{x|(x,x) \in R^{\mathcal{I}}\}$, $(\exists R.Self[R := Q])^{\mathcal{I}} = \{x|(x,x) \in Q^{\mathcal{I}}\} = (\exists Q.Self)^{\mathcal{I}}$.

30 $(\exists R^-.Self)^{\mathcal{I}} = \{x|(x,x) \in R^{\mathcal{I}}\}$, $(\exists R^-.Self[R := Q])^{\mathcal{I}} = \{x|(x,x) \in Q^{\mathcal{I}}\} = (\exists Q^-.Self)^{\mathcal{I}}$.

31 As the valuation of R is left unmodified by $R' := (i,j)$, $(\exists R.Self[R' := (i,j)])^{\mathcal{I}} = (\exists R.Self)^{\mathcal{I}}$.

32 As the valuation of R is left unmodified by $R' := (i,j)$, $(\exists R^-.Self[R' := (i,j)])^{\mathcal{I}} = (\exists R^-.Self)^{\mathcal{I}}$.

33 $(\exists R.Self)^{\mathcal{I}} = \{x|(x,x) \in R^{\mathcal{I}}\}$, $(\exists R.Self[R := (i,j)])^{\mathcal{I}} = \{x|x = i \wedge x = j\} = (\{i\} \sqcap \{j\})^{\mathcal{I}}$.

34 $(\exists R^-.Self)^{\mathcal{I}} = \{x|(x,x) \in R^{\mathcal{I}}\}$, $(\exists R^-.Self[R := (i,j)])^{\mathcal{I}} = \{x|x = i \wedge x = j\} = (\{i\} \sqcap \{j\})^{\mathcal{I}}$.

35 As $(\exists R.Self)^{\mathcal{I}} = \{x|(x,x) \in R^{\mathcal{I}}\}$, $(\exists R.Self[i \gg^{in} j])^{\mathcal{I}} = \{x|(x,x) \in R^{\mathcal{I}} \backslash \{(y,i) \in R^{\mathcal{I}}\} \cup \{(y,j)|(y,i) \in R^{\mathcal{I}}\}\}$. Thus $(\exists R.Self[i \gg^{in} j])^{\mathcal{I}} = \{i|(i,i) \in R^{\mathcal{I}} \wedge i = j\} \cup \{j|j \neq i \wedge ((j,j) \in R^{\mathcal{I}} \vee (j,i) \in R^{\mathcal{I}})\} \cup \{x|x \neq i \wedge x \neq j \wedge (x,x) \in R^{\mathcal{I}}\}$. Thus $(\exists R.Self[i \gg^{in} j])^{\mathcal{I}} = \{x|((x = i \wedge x = j) \vee (x \neq i \wedge x \neq j)) \wedge (x,x) \in R^{\mathcal{I}}\} \cup \{x|x = j \wedge x \neq i \wedge ((x,x) \in R^{\mathcal{I}} \vee (x,i) \in R^{\mathcal{I}}\}$. That is $(\exists R.Self[i \gg^{in} j])^{\mathcal{I}} = ((((\{i\} \Leftrightarrow \{j\}) \Rightarrow \exists R.Self) \sqcup (\neg\{i\} \sqcap \{j\} \Rightarrow \exists R.Self \sqcup \exists R.\{i\}))^{\mathcal{I}}$.

36 As $(\exists R^-.Self)^{\mathcal{I}} = \{x|(x,x) \in R^{\mathcal{I}}\}$, $(\exists R^-.Self[i \gg^{in} j])^{\mathcal{I}} = \{x|(x,x) \in R^{\mathcal{I}} \backslash \{(y,i) \in R^{\mathcal{I}}\} \cup \{(y,j)|(y,i) \in R^{\mathcal{I}}\}\}$. Thus $(\exists R^-.Self[i \gg^{in} j])^{\mathcal{I}} = \{i|(i,i) \in R^{\mathcal{I}} \wedge i = j\} \cup \{j|j \neq i \wedge ((j,j) \in R^{\mathcal{I}} \vee (j,i) \in R^{\mathcal{I}})\} \cup \{x|x \neq i \wedge x \neq j \wedge (x,x) \in R^{\mathcal{I}}\}$. Thus $(\exists R^-.Self[i \gg^{in} j])^{\mathcal{I}} = \{x|((x = i \wedge x = j) \vee (x \neq i \wedge x \neq j)) \wedge (x,x) \in R^{\mathcal{I}}\} \cup \{x|x = j \wedge x \neq i \wedge ((x,x) \in R^{\mathcal{I}} \vee (x,i) \in R^{\mathcal{I}}\}$. That is $(\exists R^-.Self[i \gg^{in} j])^{\mathcal{I}} = ((((\{i\} \Leftrightarrow \{j\}) \Rightarrow \exists R.Self) \sqcup (\neg\{i\} \sqcap \{j\} \Rightarrow \exists R.Self \sqcup \exists R.\{i\}))^{\mathcal{I}}$.

37 As $(\exists R.Self)^{\mathcal{I}} = \{x|(x,x) \in R^{\mathcal{I}}\}$, $(\exists R.Self[i \gg^{out} j])^{\mathcal{I}} = \{x|(x,x) \in R^{\mathcal{I}} \backslash \{(i,y) \in R^{\mathcal{I}}\} \cup \{(j,y)|(i,y) \in R^{\mathcal{I}}\}\}\}$. Thus $(\exists R.Self[i \gg^{out} j])^{\mathcal{I}} = \{i|(i,i) \in R^{\mathcal{I}} \wedge i = j\} \cup \{j|j \neq i \wedge ((j,j) \in R^{\mathcal{I}} \vee (i,j) \in R^{\mathcal{I}})\} \cup \{x|x \neq i \wedge x \neq j \wedge (x,x) \in R^{\mathcal{I}}\}$. Thus $(\exists R.Self[i \gg^{in} j])^{\mathcal{I}} = \{x|((x = i \wedge x = j) \vee (x \neq i \wedge x \neq j)) \wedge (x,x) \in R^{\mathcal{I}}\} \cup \{x|x = j \wedge x \neq i \wedge ((x,x) \in R^{\mathcal{I}} \vee (i,x) \in R^{\mathcal{I}}\}$. That is $(\exists R.Self[i \gg^{out} j])^{\mathcal{I}} = ((\{i\} \Leftrightarrow \{j\}) \Rightarrow \exists R.Self) \sqcup (\neg\{i\} \sqcap \{j\} \Rightarrow \exists R.Self \sqcup \exists R^{-}.\{i\}))^{\mathcal{I}}$.

38 As $(\exists R^{-}.Self)^{\mathcal{I}} = \{x|(x,x) \in R^{\mathcal{I}}\}$, $(\exists R^{-}.Self[i \gg^{out} j])^{\mathcal{I}} = \{x|(x,x) \in R^{\mathcal{I}} \backslash \{(i,y) \in R^{\mathcal{I}}\} \cup \{(j,y)|(i,y) \in R^{\mathcal{I}}\}\}\}$. Thus $(\exists R.Self[i \gg^{out} j])^{\mathcal{I}} = \{i|(i,i) \in R^{\mathcal{I}} \wedge i = j\} \cup \{j|j \neq i \wedge ((j,j) \in R^{\mathcal{I}} \vee (i,j) \in R^{\mathcal{I}})\} \cup \{x|x \neq i \wedge x \neq j \wedge (x,x) \in R^{\mathcal{I}}\}$. Thus $(\exists R^{-}.Self[i \gg^{in} j])^{\mathcal{I}} = \{x|((x = i \wedge x = j) \vee (x \neq i \wedge x \neq j)) \wedge (x,x) \in R^{\mathcal{I}}\} \cup \{x|x = j \wedge x \neq i \wedge ((x,x) \in R^{\mathcal{I}} \vee (i,x) \in R^{\mathcal{I}}\}$. That is $(\exists R^{-}.Self[i \gg^{out} j])^{\mathcal{I}} = ((\{i\} \Leftrightarrow \{j\}) \Rightarrow \exists R.Self) \sqcup (\neg\{i\} \sqcap \{j\} \Rightarrow \exists R.Self \sqcup \exists R^{-}.\{i\}))^{\mathcal{I}}$.

39 As $(\bowtie\ n\ R\ \phi)^{\mathcal{I}} = \{x \mid card\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\} \bowtie n\}$, $((\bowtie\ n\ R\ \phi)[C := C \pm i])^{\mathcal{I}} = \{x \mid card\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi[C := C \pm i]^{\mathcal{I}}\} \bowtie n\}$ as $R^{\mathcal{I}}$ is independent of the valuation of C. Thus $((\bowtie\ n\ R\ \phi)[C := C \pm i])^{\mathcal{I}} = (\bowtie\ n\ R\ \phi[C := C \pm i])^{\mathcal{I}}$.

40 As $(\bowtie\ n\ R^{-}\ \phi)^{\mathcal{I}} = \{x \mid card\{y \mid (y,x) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\} \bowtie n\}$, $((\bowtie\ n\ R^{-}\ \phi)[C := C \pm i])^{\mathcal{I}} = \{x \mid card\{y \mid (y,x) \in R^{\mathcal{I}} \wedge y \in \phi[C := C \pm i]^{\mathcal{I}}\} \bowtie n\}$ as $R^{\mathcal{I}}$ is independent of the valuation of C. Thus $((\bowtie\ n\ R^{-}\ \phi)[C := C \pm i])^{\mathcal{I}} = (\bowtie\ n\ R^{-}\ \phi[C := C \pm i])^{\mathcal{I}}$.

41 As $(\bowtie\ n\ R\ \phi)^{\mathcal{I}} = \{x \mid card\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\} \bowtie n\}$, $((\bowtie\ n\ R\ \phi)[C := \psi])^{\mathcal{I}} = \{x \mid card\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi[C := \psi]^{\mathcal{I}}\} \bowtie n\}$ as $R^{\mathcal{I}}$ is independent of the valuation of C. Thus $((\bowtie\ n\ R\ \phi)[C := \psi])^{\mathcal{I}} = (\bowtie\ n\ R\ \phi[C := \psi])^{\mathcal{I}}$.

42 As $(\bowtie\ n\ R^{-}\ \phi)^{\mathcal{I}} = \{x \mid card\{y \mid (y,x) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\} \bowtie n\}$, $((\bowtie\ n\ R^{-}\ \phi)[C := \psi])^{\mathcal{I}} = \{x \mid card\{y \mid (y,x) \in R^{\mathcal{I}} \wedge y \in \phi[C := \psi]^{\mathcal{I}}\} \bowtie n\}$ as $R^{\mathcal{I}}$ is independent of the valuation of C. Thus $((\bowtie\ n\ R^{-}\ \phi)[C := \psi])^{\mathcal{I}} = (\bowtie\ n\ R^{-}\ \phi[C := \psi])^{\mathcal{I}}$.

43 As $(\bowtie\ n\ R\ \phi)^{\mathcal{I}} = \{x \mid card\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\} \bowtie n\}$, $((\bowtie\ n\ R\ \phi)[R' := R' \pm (i,j)])^{\mathcal{I}} = \{x \mid card\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi[R' := R' \pm (i,j)]^{\mathcal{I}}\} \bowtie n\}$ as $R^{\mathcal{I}}$ is independent of the valuation of R'. Thus $((\bowtie\ n\ R\ \phi)[R' := R' \pm (i,j)])^{\mathcal{I}} = (\bowtie\ n\ R\ \phi[R' := R' \pm (i,j)])^{\mathcal{I}}$.

44 As $(\bowtie\ n\ R^{-}\ \phi)^{\mathcal{I}} = \{x \mid card\{y \mid (y,x) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\} \bowtie n\}$, $((\bowtie\ n\ R^{-}\ \phi)[R' := R' \pm (i,j)])^{\mathcal{I}} = \{x \mid card\{y \mid (y,x) \in R^{\mathcal{I}} \wedge y \in \phi[R' := R' \pm (i,j)]^{\mathcal{I}}\} \bowtie n\}$ as $R^{\mathcal{I}}$ is

independent of the valuation of R'. Thus
$((\bowtie\ n\ R^-\ \phi)[R' := R' \pm (i,j)])^{\mathcal{I}} = (\bowtie\ n\ R^-\ \phi[R' := R' \pm (i,j)])^{\mathcal{I}}$.

45 As $(\bowtie\ n\ R\ \phi)^{\mathcal{I}} = \{x \mid card\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\} \bowtie n\}$,
$((\bowtie\ n\ R\ \phi)[R := R + (i,j)])^{\mathcal{I}} =$
$\{x \mid card\{y \mid (x,y) \in R[R := R+(i,j)]^{\mathcal{I}} \wedge y \in \phi[R := R+(i,j)]^{\mathcal{I}}\} \bowtie n\} =$
$\{x \mid card\{y \mid (x,y) \in R^{\mathcal{I}} \cup \{(i,j)\} \wedge y \in \phi[R := R + (i,j)]^{\mathcal{I}}\} \bowtie n\}$. We consider $\{y \mid (x,y) \in R^{\mathcal{I}} \cup \{(i,j)\} \wedge y \in \phi[R := R + (i,j)]^{\mathcal{I}}\}$ and try the possible sets:

- If $x \neq i$ or $j \notin \phi[R := R + (i,j)]^{\mathcal{I}}$ or $(i,j) \in R^{\mathcal{I}}$, then
  $\{y \mid (x,y) \in R^{\mathcal{I}} \cup \{(i,j)\} \wedge y \in \phi[R := R + (i,j)]^{\mathcal{I}}\} =$
  $\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi[R := R + (i,j)]^{\mathcal{I}}\}$,
- else, $x = i$ and $j \in \phi[R := R + (i,j)]^{\mathcal{I}}$ and $(i,j) \notin R^{\mathcal{I}}$, and thus
  $\{y \mid (x,y) \in R^{\mathcal{I}} \cup \{(i,j)\} \wedge y \in \phi[R := R + (i,j)]^{\mathcal{I}}\} =$
  $\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi[R := R + (i,j)]^{\mathcal{I}}\} \cup \{(i,j)\}$. As $\{(i,j)\}$ is disjoint from $\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi[R := R + (i,j)]^{\mathcal{I}}\}$, the cardinality of
  $\{y \mid (x,y) \in R^{\mathcal{I}} \cup \{(i,j)\} \wedge y \in \phi[R := R + (i,j)]^{\mathcal{I}}\}$ is exactly the cardinality of $\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi[R := R + (i,j)]^{\mathcal{I}}\} + 1$.
- Thus,
  $\{x \mid card\{y \mid (x,y) \in R[R := R + (i,j)]^{\mathcal{I}} \wedge y \in \phi[R := R + (i,j)]^{\mathcal{I}}\} \bowtie n\}$
  $= \{x \mid$
  $(x \neq i \vee j \notin \phi[R := R + (i,j)]^{\mathcal{I}} \vee (i,j) \in R^{\mathcal{I}} \Rightarrow$
  $card\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi[R := R + (i,j)]^{\mathcal{I}}\} \bowtie n)$
  $\wedge (x = i \wedge j \in \phi[R := R + (i,j)]^{\mathcal{I}} \wedge (i,j) \notin R^{\mathcal{I}} \Rightarrow$
  $card\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi[R := R + (i,j)]^{\mathcal{I}}\} \bowtie (n-1))\}$

Thus, $((\bowtie\ n\ R\ \phi)[R := R + (i,j)])^{\mathcal{I}} =$
$(((\{i\} \sqcap \exists U.(\{j\}\ \sqcap \phi[R := R + (i,j)]) \sqcap \forall R.\neg\{j\}) \Rightarrow$
$(\bowtie\ (n-1)\ R\ \phi[R := R + (i,j)]))$
$\sqcap ((\neg\{i\} \sqcup \forall U.(\neg\{j\} \sqcup \neg\phi[R := R + (i,j)]) \sqcup \exists R.\{j\}) \Rightarrow$
$(\bowtie\ n\ R\ \phi[R := R + (i,j)]))))^{\mathcal{I}}$

46 One can see that $(\bowtie\ n\ R^-\ \phi)[R := R + (i,j)]$ is similar to $(\bowtie\ n\ R^-\ \phi)[R^- := R^- + (j,i)]$, that is replacing R by R^- and swapping i and j in the previous case.

47 As $(\bowtie\ n\ R\ \phi)^{\mathcal{I}} = \{x \mid card\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\} \bowtie n\}$,
$((\bowtie\ n\ R\ \phi)[R := R - (i,j)])^{\mathcal{I}} =$
$\{x \mid card\{y \mid (x,y) \in R[R := R-(i,j)]^{\mathcal{I}} \wedge y \in \phi[R := R-(i,j)]^{\mathcal{I}}\} \bowtie n\} =$
$\{x \mid card\{y \mid (x,y) \in R^{\mathcal{I}} \cap \overline{\{(i,j)\}} \wedge y \in \phi[R := R - (i,j)]^{\mathcal{I}}\} \bowtie n\}$. We consider $\{y \mid (x,y) \in R^{\mathcal{I}} \cap \overline{\{(i,j)\}} \wedge y \in \phi[R := R + (i,j)]^{\mathcal{I}}\}$ and try the possible sets:

- If $x \neq i$ or $j \notin \phi[R := R - (i,j)]^{\mathcal{I}}$ or $(i,j) \notin R^{\mathcal{I}}$, then
  $\{y \mid (x,y) \in R^{\mathcal{I}} \cap \overline{\{(i,j)\}} \wedge y \in \phi[R := R - (i,j)]^{\mathcal{I}}\} =$
  $\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi[R := R - (i,j)]^{\mathcal{I}}\}$,

- else, $x = i$ and $j \in \phi[R := R - (i,j)]^{\mathcal{I}}$ and $(i,j) \in R^{\mathcal{I}}$, and thus $\{y \mid (x,y) \in R^{\mathcal{I}} \cap \underline{\{(i,j)\}} \wedge y \in \phi[R := R - \underline{(i,j)}]^{\mathcal{I}}\} = \{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi[R := R - (i,j)]^{\mathcal{I}}\} \cap \{i,j\}$. As $\{(i,j)\} \subseteq \{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \underline{\phi}[R := R - (i,j)]^{\mathcal{I}}\}$, the cardinality of $\{y \mid (x,y) \in R^{\mathcal{I}} \cap \underline{\{(i,j)\}} \wedge y \in \phi[R := R + (i,j)]^{\mathcal{I}}\}$ is exactly the cardinality of $\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi[R := R - (i,j)]^{\mathcal{I}}\} - 1$.
- Thus, $\{x \mid card\{y \mid (x,y) \in R[R := R - (i,j)]^{\mathcal{I}} \wedge y \in \phi[R := R - (i,j)]^{\mathcal{I}}\} \bowtie n\}$ $= \{x \mid$ $(x \neq i \vee j \notin \phi[R := R - (i,j)]^{\mathcal{I}} \vee (i,j) \notin R^{\mathcal{I}} \Rightarrow$ $card\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi[R := R - (i,j)]^{\mathcal{I}}\} \bowtie n)$ $\wedge (x = i \wedge j \in \phi[R := R - (i,j)]^{\mathcal{I}} \wedge (i,j) \in R^{\mathcal{I}} \Rightarrow$ $card\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi[R := R - (i,j)]^{\mathcal{I}}\} \bowtie (n+1))\}$

Thus, $((\bowtie\ n\ R\ \phi)[R := R - (i,j)])^{\mathcal{I}} =$
$((((\{i\} \sqcap \exists U.(\{j\} \sqcap \phi[R := R - (i,j)]) \sqcap \exists R.\{j\}) \Rightarrow$
$(\bowtie\ (n+1)\ R\ \phi[R := R - (i,j)]))$
$\sqcap ((\neg\{i\} \sqcup \forall U.(\neg\{j\} \sqcup \neg\phi[R := R - (i,j)]) \sqcup \forall R.\neg\{j\}) \Rightarrow$
$(\bowtie\ n\ R\ \phi[R := R - (i,j)]))))^{\mathcal{I}}$

48 One can see that $(\bowtie\ n\ R^-\ \phi)[R := R - (i,j)]$ is similar to $(\bowtie\ n\ R^-\ \phi)[R^- := R^- - (j,i)]$, that is replacing R by R^- and swapping i and j in the previous case.

49 As the valuation of R is left unmodified by $R' := Q$, $(\bowtie\ n\ R\ \phi)[R' := Q]^{\mathcal{I}} = (\bowtie\ n\ R\ \phi[R' := Q])^{\mathcal{I}}$.

50 As the valuation of R is left unmodified by $R' := Q$, $(\bowtie\ n\ R^-\ \phi)[R' := Q]^{\mathcal{I}} = (\bowtie\ n\ R^-\ \phi[R' := Q])^{\mathcal{I}}$.

51 As $(\bowtie\ n\ R\ \phi)^{\mathcal{I}} = \{x \mid card\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\} \bowtie n\}$, $(\bowtie\ n\ R\ \phi)[R := Q]^{\mathcal{I}} = \{x \mid card\{y \mid (x,y) \in Q^{\mathcal{I}} \wedge y \in \phi[R := Q]^{\mathcal{I}}\} \bowtie n\} = (\bowtie\ n\ Q\ \phi[R' := Q])^{\mathcal{I}}$.

52 As $(\bowtie\ n\ R^-\ \phi)^{\mathcal{I}} = \{x \mid card\{y \mid (y,x) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\} \bowtie n\}$, $(\bowtie\ n\ R\ \phi)[R := Q]^{\mathcal{I}} = \{x \mid card\{y \mid (y,x) \in Q^{\mathcal{I}} \wedge y \in \phi[R := Q]^{\mathcal{I}}\} \bowtie n\} = (\bowtie\ n\ Q^-\ \phi[R' := Q])^{\mathcal{I}}$.

53 As the valuation of R is left unmodified by $R' := (i,j)$, $(\bowtie\ n\ R\ \phi)[R' := (i,j)]^{\mathcal{I}} = (\bowtie\ n\ R\ \phi[R' := (i,j)])^{\mathcal{I}}$.

54 As $(<\ 1\ R\ \phi)^{\mathcal{I}} = \{x \mid card\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\} < 1\}$, $(<\ 1\ R\ \phi)[R := (i,j)]^{\mathcal{I}} = \{x \mid card\{y \mid x = i \wedge y = j \wedge y \in \phi[R := (i,j)]^{\mathcal{I}}\} < 1\}$. That is, $(<\ 1\ R\ \phi)[R := (i,j)]^{\mathcal{I}} = \{x \mid x \neq i \vee j \notin \phi[R := (i,j)]^{\mathcal{I}}\}$. Thus $(<\ 1\ R\ \phi)[R := (i,j)]^{\mathcal{I}} = (\neg\{i\} \sqcup \forall U.(\neg\{j\} \sqcup \neg\phi[R := (i,j)]))^{\mathcal{I}}$.

55 As $(<\ n\ R\ \phi)^{\mathcal{I}} = \{x \mid card\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\} < n\}$, $(<\ n\ R\ \phi)[R := (i,j)]^{\mathcal{I}} = \{x \mid card\{y \mid x = i \wedge y = j \wedge y \in \phi[R := (i,j)]^{\mathcal{I}}\} < n\}$. But, as $\{y \mid x = i \wedge y = j \wedge y \in \phi[R := (i,j)]^{\mathcal{I}}\} \subseteq \{j\}$, $card\{y \mid x = i \wedge y = j \wedge y \in \phi[R := (i,j)]^{\mathcal{I}}\} \leq 1$ and thus $(<\ n\ R\ \phi)^{\mathcal{I}} = \top^{\mathcal{I}}$.

56 As $(\geq\ 1\ R\ \phi)^{\mathcal{I}} = \{x \mid card\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\} \geq 1\}$, $(\geq\ 1\ R\ \phi)[R := (i,j)]^{\mathcal{I}} = \{x \mid card\{y \mid x = i \wedge y = j \wedge y \in \phi[R := (i,j)]^{\mathcal{I}}\} \geq 1\}$. That is, $(\geq\ 1\ R\ \phi)[R := (i,j)]^{\mathcal{I}} = \{x \mid x = i \wedge j \in \phi[R := (i,j)]^{\mathcal{I}}\}$. Thus $(\geq\ 1\ R\ \phi)[R := (i,j)]^{\mathcal{I}} = (\{i\} \sqcap \exists U.(\{j\} \sqcap \phi[R := (i,j)]))^{\mathcal{I}}$.

57 As $(\geq\ n\ R\ \phi)^{\mathcal{I}} = \{x \mid card\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\} \geq n\}$, $(\geq\ n\ R\ \phi)[R := (i,j)]^{\mathcal{I}} = \{x \mid card\{y \mid x = i \wedge y = j \wedge y \in \phi[R := (i,j)]^{\mathcal{I}}\} \geq n\}$. But, as $\{y \mid x = i \wedge y = j \wedge y \in \phi[R := (i,j)]^{\mathcal{I}}\} \subseteq \{j\}$, $card\{y \mid x = i \wedge y = j \wedge y \in \phi[R := (i,j)]^{\mathcal{I}}\} \leq 1$ and thus $(\geq\ n\ R\ \phi)^{\mathcal{I}} = \bot^{\mathcal{I}}$.

58 As $(<\ \ 1\ R^{-}\ \phi)^{\mathcal{I}} = \{x \mid card\{y \mid (y,x) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\} < 1\}$, $(<\ 1\ R^{-}\ \phi)[R := (i,j)]^{\mathcal{I}} = \{x \mid card\{y \mid x = j \wedge y = i \wedge y \in \phi[R := (i,j)]^{\mathcal{I}}\} < 1\}$. That is, $(<\ 1\ R^{-}\ \phi)[R := (i,j)]^{\mathcal{I}} = \{x \mid x \neq j \vee i \notin \phi[R := (i,j)]^{\mathcal{I}}\}$. Thus $(<\ 1\ R^{-}\ \phi)[R := (i,j)]^{\mathcal{I}} = (\neg\{j\} \sqcup \forall U.(\neg\{i\} \sqcup \neg\phi[R := (i,j)]))^{\mathcal{I}}$.

59 As $(<\ \ n\ R^{-}\ \phi)^{\mathcal{I}} = \{x \mid card\{y \mid (y,x) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\} < n\}$, $(<\ n\ R^{-}\ \phi)[R := (i,j)]^{\mathcal{I}} = \{x \mid card\{y \mid x = j \wedge y = i \wedge y \in \phi[R := (i,j)]^{\mathcal{I}}\} < n\}$. But, as $\{y \mid x = j \wedge y = i \wedge y \in \phi[R := (i,j)]^{\mathcal{I}}\} \subseteq \{i\}$, $card\{y \mid x = j \wedge y = i \wedge y \in \phi[R := (i,j)]^{\mathcal{I}}\} \leq 1$ and thus $(<\ n\ R^{-}\ \phi)^{\mathcal{I}} = \top^{\mathcal{I}}$.

60 As $(\geq\ \ 1\ R^{-}\ \phi)^{\mathcal{I}} = \{x \mid card\{y \mid (y,x) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\} \geq 1\}$, $(\geq\ 1\ R^{-}\ \phi)[R := (i,j)]^{\mathcal{I}} = \{x \mid card\{y \mid x = j \wedge y = i \wedge y \in \phi[R := (i,j)]^{\mathcal{I}}\} \geq 1\}$. That is, $(\geq\ 1\ R^{-}\ \phi)[R := (i,j)]^{\mathcal{I}} = \{x \mid x = j \wedge i \in \phi[R := (i,j)]^{\mathcal{I}}\}$. Thus $(\geq\ 1\ R^{-}\ \phi)[R := (i,j)]^{\mathcal{I}} = (\{j\} \sqcap \exists U.(\{i\} \sqcap \phi[R := (i,j)]))^{\mathcal{I}}$.

61 As $(\geq\ \ n\ R^{-}\ \phi)^{\mathcal{I}} = \{x \mid card\{y \mid (y,x) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\} \geq n\}$, $(\geq\ n\ R^{-}\ \phi)[R := (i,j)]^{\mathcal{I}} = \{x \mid card\{y \mid x = j \wedge y = i \wedge y \in \phi[R := (i,j)]^{\mathcal{I}}\} \geq n\}$. But, as $\{y \mid x = j \wedge y = i \wedge y \in \phi[R := (i,j)]^{\mathcal{I}}\} \subseteq \{i\}$, $card\{y \mid x = j \wedge y = i \wedge y \in \phi[R := (i,j)]^{\mathcal{I}}\} \leq 1$ and thus $(\geq\ n\ R^{-}\ \phi)^{\mathcal{I}} = \bot^{\mathcal{I}}$.

62 As $(\bowtie\ \ n\ \ R\ \ \phi)^{\mathcal{I}} = \{x \mid card\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\} \bowtie n\}$, $(\bowtie\ n\ R\ \phi)[i \gg^{in} j]^{\mathcal{I}} = \{x \mid card\{y \mid ((x,y) \in R^{\mathcal{I}} \wedge y \neq i \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}) \vee ((x,i) \in R^{\mathcal{I}} \wedge j \in \phi[i \gg^{in} j]^{\mathcal{I}})\} \bowtie n\}$. We consider $\{y \mid ((x,y) \in R^{\mathcal{I}} \wedge y \neq i \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}) \vee ((x,i) \in R^{\mathcal{I}} \wedge j \in \phi[i \gg^{in} j]^{\mathcal{I}})\}$ and look at the various possibilities:

- If $i = j$, $\{y \mid ((x,y) \in R^{\mathcal{I}} \wedge y \neq i \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}) \vee ((x,i) \in R^{\mathcal{I}} \wedge j \in \phi[i \gg^{in} j]^{\mathcal{I}})\} = \{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}\}$ and thus there cardinalities are the same.
- otherwise, if $(x,i) \in R^{\mathcal{I}}$ and $i \in \phi[i \gg^{in} j]^{\mathcal{I}}$, $(x,j) \notin R^{\mathcal{I}}$ and $j \notin \phi[R := R - (i,j)]^{\mathcal{I}}$, then $card\{y \mid ((x,y) \in R^{\mathcal{I}} \wedge y \neq i \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}) \vee ((x,i) \in R^{\mathcal{I}} \wedge j \in \phi[i \gg^{in} j]^{\mathcal{I}})\} =$
  $card\{y \mid ((x,y) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}\} - 1$.
- if $(x,i) \in R^{\mathcal{I}}$ and $i \notin \phi[i \gg^{in} j]^{\mathcal{I}}$, $(x,j) \notin R^{\mathcal{I}}$ and $j \in \phi[i \gg^{in} j]^{\mathcal{I}}$, then $card\{y \mid ((x,y) \in R^{\mathcal{I}} \wedge y \neq i \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}) \vee ((x,i) \in R^{\mathcal{I}} \wedge j \in \phi[i \gg^{in} j]^{\mathcal{I}})\} =$
  $card\{y \mid ((x,y) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}\} + 1$.

- if $(x,i) \in R^{\mathcal{I}}$ and $i \in \phi[i \gg^{in} j]^{\mathcal{I}}$ and $(x,j) \in R^{\mathcal{I}}$, then $card\{y \mid ((x,y) \in R^{\mathcal{I}} \wedge y \neq i \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}) \vee ((x,i) \in R^{\mathcal{I}} \wedge j \in \phi[i \gg^{in} j]^{\mathcal{I}})\} = card\{y \mid ((x,y) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}\} - 1$.
- otherwise, that is if $(x,i) \notin R^{\mathcal{I}}$, or if $(x,i) \in R^{\mathcal{I}}$, $i \notin \phi[i \gg^{in} j]^{\mathcal{I}}$ and $(x,j) \in R^{\mathcal{I}}$, or if $(x,i) \in R^{\mathcal{I}}$, $i \in \phi[i \gg^{in} j]^{\mathcal{I}}$, $(x,j) \notin R^{\mathcal{I}}$ and $j \in \phi[i \gg^{in} j]^{\mathcal{I}}$, or if $(x,i) \in R^{\mathcal{I}}$, $i \notin \phi[i \gg^{in} j]^{\mathcal{I}}$, $(x,j) \notin R^{\mathcal{I}}$ and $j \notin \phi[i \gg^{in} j]^{\mathcal{I}}$ then $\{y \mid ((x,y) \in R^{\mathcal{I}} \wedge y \neq i \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}) \vee ((x,i) \in R^{\mathcal{I}} \wedge j \in \phi[i \gg^{in} j]^{\mathcal{I}})\} = \{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}\}$ and thus there cardinalities are the same.

Thus, $((\bowtie\ n\ R\ \phi)[i \gg^{in} j])^{\mathcal{I}} =$
$((\exists U.(\{i\} \sqcap \{j\}) \Rightarrow (\bowtie\ n\ R\ \phi[i \gg^{in} j])) \sqcap (\exists U.(\{i\} \sqcap \neg\{j\}) \Rightarrow (\exists R.(\{i\} \sqcap \phi[i \gg^{in} j]) \sqcap \forall R.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \neg\phi[i \gg^{in} j]) \Rightarrow (\bowtie\ (n+1)\ R\ \phi[i \gg^{in} j])) \sqcap (\exists R.(\{i\} \sqcap \neg\phi[i \gg^{in} j]) \sqcap \forall R.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \phi[i \gg^{in} j]) \Rightarrow (\bowtie\ (n-1)\ R\ \phi[i \gg^{in} j])) \sqcap (\exists R.(\{i\} \sqcap \phi[i \gg^{in} j]) \sqcap \exists R.\{j\} \Rightarrow (\bowtie\ (n+1)\ R\ \phi[i \gg^{in} j])) \sqcap ((\forall R.\neg\{i\}) \sqcup (\exists R.(\{i\} \sqcap \neg\phi[i \gg^{in} j]) \sqcap \exists R.\{j\}) \sqcup (\exists R.(\{i\} \sqcap \phi[i \gg^{in} j]) \sqcap \forall R.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \phi[i \gg^{in} j])) \sqcup (\exists R.(\{i\} \sqcap \neg\phi[i \gg^{in} j]) \sqcap \forall R.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \neg\phi[i \gg^{in} j])) \Rightarrow (\bowtie\ n\ R\ \phi[i \gg^{in} j])))^{\mathcal{I}}$

63 As $(<\ n\ R^{-}\ \phi)^{\mathcal{I}} = \{x \mid card\{y \mid (y,x) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\} < n\}$, $(< n\ R^{-}\ \phi)[i \gg^{in} j]^{\mathcal{I}} = \{x \mid card\{y \mid ((y,x) \in R^{\mathcal{I}} \wedge x \neq i \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}) \vee ((y,i) \in R^{\mathcal{I}} \wedge x = j \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}})\} < n\}$. We consider $\{y \mid ((y,x) \in R^{\mathcal{I}} \wedge x \neq i \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}) \vee ((y,i) \in R^{\mathcal{I}} \wedge x = j \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}})\}$ and look at the various possibilities:

- If $x = i$ and $x \neq j$, $\{y \mid ((y,x) \in R^{\mathcal{I}} \wedge x \neq i \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}) \vee ((y,i) \in R^{\mathcal{I}} \wedge x = j \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}})\} = \emptyset$ and thus its cardinality is $< n$.
- otherwise, if $x \neq i$ and $x = j$, then $card\{y \mid ((y,x) \in R^{\mathcal{I}} \wedge x \neq i \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}) \vee ((y,i) \in R^{\mathcal{I}} \wedge x = j \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}})\} = card(\{y|(y,j) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}\} \cup \{y|(y,i) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}\}) = card(\{y|(y,j) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}\}) + card(\{y|(y,i) \in R^{\mathcal{I}} \wedge (y,j) \notin R^{\mathcal{I}} \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}\})$.
- otherwise, either $x = i = j$ or $x \neq i$ and $x \neq j$, and thus $\{y \mid ((y,x) \in R^{\mathcal{I}} \wedge x \neq i \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}) \vee ((y,i) \in R^{\mathcal{I}} \wedge x = j \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}})\} = \{y \mid (y,x) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}\}$ and thus there cardinalities are the same.

Thus, $((<\ n\ R^{-}\ \phi)[i \gg^{in} j])^{\mathcal{I}} =$
$((\{i\} \sqcap \neg\{j\}) \sqcup (\neg\{i\} \sqcap \{j\} \Rightarrow \bigsqcup_{k \in [0,n]} (<\ k\ R^{-}\ \phi[i \gg^{in} j]) \sqcap \exists U.(\{i\} \sqcap (< (n-k)\ R^{-}\ (\phi[i \gg^{in} j] \sqcap \neg\exists R^{-}.\{j\})))) \sqcup ((\{i\} \Leftrightarrow \{j\}) \Rightarrow (<\ n\ R^{-}\ \phi[i \gg^{in} j])))^{\mathcal{I}}$

64 As $(\geq\ n\ R^{-}\ \phi)^{\mathcal{I}} = \{x \mid card\{y \mid (y,x) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\} \geq n\}$, $(\geq n\ R^{-}\ \phi)[i \gg^{in} j]^{\mathcal{I}} = \{x \mid card\{y \mid ((y,x) \in R^{\mathcal{I}} \wedge x \neq i \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}) \vee ((y,i) \in R^{\mathcal{I}} \wedge x = j \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}})\} \geq n\}$. We consider

$\{y \mid ((y,x) \in R^{\mathcal{I}} \wedge x \neq i \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}) \vee ((y,i) \in R^{\mathcal{I}} \wedge x = j \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}})\}$ and look at the various possibilities:

- If $x = i$ and $x \neq j$, $\{y \mid ((y,x) \in R^{\mathcal{I}} \wedge x \neq i \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}) \vee ((y,i) \in R^{\mathcal{I}} \wedge x = j \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}})\} = \emptyset$ and thus its cardinality is $< n$.
- otherwise, if $x \neq i$ and $x = j$, then $card\{y \mid ((y,x) \in R^{\mathcal{I}} \wedge x \neq i \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}) \vee ((y,i) \in R^{\mathcal{I}} \wedge x = j \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}})\} = card(\{y|(y,j) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}\} \cup \{y|(y,i) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}\}) = card(\{y|(y,j) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}\}) + card(\{y|(y,i) \in R^{\mathcal{I}} \wedge (y,j) \notin R^{\mathcal{I}} \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}\})$.
- otherwise, either $x = i = j$ or $x \neq i$ and $x \neq j$, and thus $\{y \mid ((y,x) \in R^{\mathcal{I}} \wedge x \neq i \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}) \vee ((y,i) \in R^{\mathcal{I}} \wedge x = j \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}})\} = \{y \mid (y,x) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{in} j]^{\mathcal{I}}\}$ and thus there cardinalities are the same.

Thus, $((\geq\ n\ R^{-}\ \phi)[i \gg^{in} j])^{\mathcal{I}} =$
$((\neg\{i\} \sqcap \{j\} \Rightarrow \bigsqcup_{k\in[0,n]}(\geq\ k\ R^{-}\ \phi[i \gg^{in} j]) \sqcap \exists U.(\{i\} \sqcap (\geq\ (n-k)\ R^{-}\ (\phi[i \gg^{in} j] \sqcap \exists R^{-}.\{j\})))) \sqcap ((\{i\} \Leftrightarrow \{j\}) \Rightarrow (\geq\ n\ R^{-}\ \phi[i \gg^{in} j])))^{\mathcal{I}}$

65 As $(<\ n\ R\ \phi)^{\mathcal{I}} = \{x \mid card\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\} < n\}$, $(<\ n\ R\ \phi)[i \gg^{out} j]^{\mathcal{I}} = \{x \mid card\{y \mid ((x,y) \in R^{\mathcal{I}} \wedge x \neq i \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}) \vee ((y,i) \in R^{\mathcal{I}} \wedge x = j \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}})\} < n\}$. We consider $\{y \mid ((y,x) \in R^{\mathcal{I}} \wedge x \neq i \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}) \vee ((y,i) \in R^{\mathcal{I}} \wedge x = j \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}})\}$ and look at the various possibilities:

- If $x = i$ and $x \neq j$, $\{y \mid ((x,y) \in R^{\mathcal{I}} \wedge x \neq i \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}) \vee ((i,y) \in R^{\mathcal{I}} \wedge x = j \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}})\} = \emptyset$ and thus its cardinality is $< n$.
- otherwise, if $x \neq i$ and $x = j$, then $card\{y \mid ((x,y) \in R^{\mathcal{I}} \wedge x \neq i \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}) \vee ((i,y) \in R^{\mathcal{I}} \wedge x = j \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}})\} = card(\{y|(j,y) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}\} \cup \{y|(i,y) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}\}) = card(\{y|(j,y) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}\}) + card(\{y|(i,y) \in R^{\mathcal{I}} \wedge (j,y) \notin R^{\mathcal{I}} \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}\})$.
- otherwise, either $x = i = j$ or $x \neq i$ and $x \neq j$, and thus $\{y \mid ((x,y) \in R^{\mathcal{I}} \wedge x \neq i \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}) \vee ((i,y) \in R^{\mathcal{I}} \wedge x = j \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}})\} = \{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}\}$ and thus there cardinalities are the same.

Thus, $((<\ n\ R^{-}\ \phi)[i \gg^{out} j])^{\mathcal{I}} =$
$((\{i\} \sqcap \neg\{j\}) \sqcup (\neg\{i\} \sqcap \{j\} \Rightarrow \bigsqcup_{k\in[0,n]}(<\ k\ R\ \phi[i \gg^{out} j]) \sqcap \exists U.(\{i\} \sqcap (<\ (n-k)\ R\ (\phi[i \gg^{out} j] \sqcap \neg\exists R.\{j\})))) \sqcup ((\{i\} \Leftrightarrow \{j\}) \Rightarrow (<\ n\ R\ \phi[i \gg^{out} j])))^{\mathcal{I}}$

66 As $(\geq\ n\ R\ \phi)^{\mathcal{I}} = \{x \mid card\{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\} \geq n\}$, $(\geq\ n\ R\ \phi)[i \gg^{out} j]^{\mathcal{I}} = \{x \mid card\{y \mid ((x,y) \in R^{\mathcal{I}} \wedge x \neq i \wedge y \in \phi[i \gg^{out}$

$j]^{\mathcal{I}}) \vee ((i,y) \in R^{\mathcal{I}} \wedge x = j \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}})\} \geq n\}$. We consider $\{y \mid ((x,y) \in R^{\mathcal{I}} \wedge x \neq i \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}) \vee ((i,y) \in R^{\mathcal{I}} \wedge x = j \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}})\}$ and look at the various possibilities:

- If $x = i$ and $x \neq j$, $\{y \mid ((x,y) \in R^{\mathcal{I}} \wedge x \neq i \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}) \vee ((i,y) \in R^{\mathcal{I}} \wedge x = j \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}})\} = \emptyset$ and thus its cardinality is $< n$.
- otherwise, if $x \neq i$ and $x = j$, then $card\{y \mid ((x,y) \in R^{\mathcal{I}} \wedge x \neq i \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}) \vee ((i,y) \in R^{\mathcal{I}} \wedge x = j \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}})\} = card(\{y|(j,y) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}\} \cup \{y|(i,y) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}\}) = card(\{y|(j,y) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}\}) + card(\{y|(i,y) \in R^{\mathcal{I}} \wedge (j,y) \notin R^{\mathcal{I}} \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}\})$.
- otherwise, either $x = i = j$ or $x \neq i$ and $x \neq j$, and thus $\{y \mid ((x,y) \in R^{\mathcal{I}} \wedge x \neq i \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}) \vee ((i,y) \in R^{\mathcal{I}} \wedge x = j \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}})\} = \{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}\}$ and thus there cardinalities are the same.

Thus, $((\geq\ n\ R\ \phi)[i \gg^{out} j])^{\mathcal{I}} =$
$((\neg\{i\} \sqcap \{j\} \Rightarrow \bigsqcup_{k \in [0,n]} (\geq\ k\ R\ \phi[i \gg^{out} j]) \sqcap \exists U.(\{i\} \sqcap (\geq\ (n - k)\ R^{-}\ (\phi[i \gg^{out} j] \sqcap \exists R.\{j\})))) \sqcap ((\{i\} \Leftrightarrow \{j\}) \Rightarrow (\geq\ n\ R\ \phi[i \gg^{out} j])))^{\mathcal{I}}$

67 As $(\bowtie\ n\ R^{-}\ \phi)^{\mathcal{I}} = \{x \mid card\{y \mid (y,x) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\} \bowtie n\}$, $(\bowtie\ n\ R^{-}\ \phi)[i \gg^{out} j]^{\mathcal{I}} = \{x \mid card\{y \mid ((y,x) \in R^{\mathcal{I}} \wedge y \neq i \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}) \vee ((i,x) \in R^{\mathcal{I}} \wedge j \in \phi[i \gg^{out} j]^{\mathcal{I}})\} \bowtie n\}$. We consider $\{y \mid ((y,x) \in R^{\mathcal{I}} \wedge y \neq i \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}) \vee ((i,x) \in R^{\mathcal{I}} \wedge j \in \phi[i \gg^{out} j]^{\mathcal{I}})\}$ and look at the various possibilities:

- If $i = j$, $\{y \mid ((y,x) \in R^{\mathcal{I}} \wedge y \neq i \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}) \vee ((i,x) \in R^{\mathcal{I}} \wedge j \in \phi[i \gg^{out} j]^{\mathcal{I}})\} = \{y \mid (y,x) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}\}$ and thus there cardinalities are the same.
- otherwise, if $(i,x) \in R^{\mathcal{I}}$ and $i \in \phi[i \gg^{out} j]^{\mathcal{I}}$, $(x,j) \notin R^{\mathcal{I}}$ and $j \notin \phi[i \gg^{out} j]^{\mathcal{I}}$, then $card\{y \mid ((y,x) \in R^{\mathcal{I}} \wedge y \neq i \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}) \vee ((i,x) \in R^{\mathcal{I}} \wedge j \in \phi[i \gg^{out} j]^{\mathcal{I}})\} =$
$card\{y \mid ((y,x) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}\} - 1$.
- if $(i,x) \in R^{\mathcal{I}}$ and $i \notin \phi[i \gg^{out} j]^{\mathcal{I}}$, $(j,x) \notin R^{\mathcal{I}}$ and $j \in \phi[i \gg^{out} j]^{\mathcal{I}}$, then $card\{y \mid ((y,x) \in R^{\mathcal{I}} \wedge y \neq i \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}) \vee ((i,x) \in R^{\mathcal{I}} \wedge j \in \phi[i \gg^{out} j]^{\mathcal{I}})\} =$
$card\{y \mid ((y,x) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}\} + 1$.
- if $(i,x) \in R^{\mathcal{I}}$ and $i \in \phi[i \gg^{out} j]^{\mathcal{I}}$ and $(j,x) \in R^{\mathcal{I}}$, then $card\{y \mid ((y,x) \in R^{\mathcal{I}} \wedge y \neq i \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}) \vee ((i,x) \in R^{\mathcal{I}} \wedge j \in \phi[i \gg^{out} j]^{\mathcal{I}})\} =$
$card\{y \mid ((y,x) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}\} - 1$.
- otherwise, that is if $(i,x) \notin R^{\mathcal{I}}$, or if $(i,x) \in R^{\mathcal{I}}$, $i \notin \phi[i \gg^{out} j]^{\mathcal{I}}$ and $(j,x) \in R^{\mathcal{I}}$, or if $(i,x) \in R^{\mathcal{I}}$, $i \in \phi[i \gg^{out} j]^{\mathcal{I}}$, $(j,x) \notin R^{\mathcal{I}}$ and $j \in \phi[i \gg^{out} j]^{\mathcal{I}}$, or if $(i,x) \in R^{\mathcal{I}}$, $i \notin \phi[i \gg^{out} j]^{\mathcal{I}}$, $(j,x) \notin R^{\mathcal{I}}$ and $j \notin \phi[i \gg^{out} j]^{\mathcal{I}}$ then $\{y \mid ((y,x) \in R^{\mathcal{I}} \wedge y \neq i \wedge y \in \phi[i \gg^{out}$

$j]^{\mathcal{I}}) \vee ((i,x) \in R^{\mathcal{I}} \wedge j \in \phi[i \gg^{out} j]^{\mathcal{I}})\} = \{y \mid (y,x) \in R^{\mathcal{I}} \wedge y \in \phi[i \gg^{out} j]^{\mathcal{I}}\}$ and thus there cardinalities are the same.

Thus, $((\bowtie\ n\ R^-\ \phi)[i \gg^{out} j])^{\mathcal{I}} =$
$((\exists U.(\{i\} \sqcap \{j\}) \Rightarrow (\bowtie\ \ n\ R^-\ \phi[i \gg^{out} j])) \sqcap (\exists U.(\{i\} \sqcap \neg\{j\}) \Rightarrow (\exists R^-.(\{i\} \sqcap \phi[i \gg^{out} j]) \sqcap \forall R^-.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \neg\phi[i \gg^{out} j]) \Rightarrow (\bowtie (n+1)\ R^-\ \phi[i \gg^{out} j]))\sqcap(\exists R^-.(\{i\}\sqcap\neg\phi[i \gg^{out} j])\sqcap\forall R^-.\neg\{j\}\sqcap\exists U.(\{j\}\sqcap \phi[i \gg^{out} j]) \Rightarrow (\bowtie\ \ (n-1)\ R^-\ \phi[i \gg^{out} j])) \sqcap (\exists R^-.(\{i\} \sqcap \phi[i \gg^{out} j]) \sqcap \exists R^-.\{j\} \Rightarrow (\bowtie\ (n+1)\ R^-\ \phi[i \gg^{out} j]))\sqcap((\forall R^-.\neg\{i\}) \sqcup (\exists R^-.(\{i\} \sqcap \neg\phi[i \gg^{out} j]) \sqcap \exists R^-.\{j\}) \sqcup (\exists R^-.(\{i\} \sqcap \phi[i \gg^{out} j]) \sqcap \forall R^-.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \phi[i \gg^{out} j])) \sqcup (\exists R^-.(\{i\} \sqcap \neg\phi[i \gg^{out} j]) \sqcap \forall R.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \neg\phi[i \gg^{out} j])) \Rightarrow (\bowtie\ n\ R^-\ \phi[i \gg^{out} j])))))^{\mathcal{I}}$

68 As $(\exists R.\phi)^{\mathcal{I}} = \{x \mid \exists y.(x,y) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\}$,
$((\exists R.\phi)[C := C \pm i])^{\mathcal{I}} = \{x \mid \exists y.(x,y) \in R^{\mathcal{I}} \wedge y \in \phi[C := C \pm i]^{\mathcal{I}}\}$ as $R^{\mathcal{I}}$ is independent of the valuation of C. Thus
$((\exists R.\phi)[C := C \pm i])^{\mathcal{I}} = (\exists R.(\phi[C := C \pm i]))^{\mathcal{I}}$.

69 As $(\exists R^-.\phi)^{\mathcal{I}} = \{x \mid \exists y.(y,x) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\}$,
$((\exists R^-.\phi)[C := C \pm i])^{\mathcal{I}} = \{x \mid \exists y.(y,x) \in R^{\mathcal{I}} \wedge y \in \phi[C := C \pm i]^{\mathcal{I}}\}$ as $R^{\mathcal{I}}$ is independent of the valuation of C. Thus
$((\exists R^-.\phi)[C := C \pm i])^{\mathcal{I}} = (\exists R^-.(\phi[C := C \pm i]))^{\mathcal{I}}$.

70 As $(\exists R.\phi)^{\mathcal{I}} = \{x \mid \exists y.(x,y) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\}$,
$((\exists R.\phi)[C := \psi])^{\mathcal{I}} = \{x \mid \exists y.(x,y) \in R^{\mathcal{I}} \wedge y \in \phi[C := \psi]^{\mathcal{I}}\}$ as $R^{\mathcal{I}}$ is independent of the valuation of C. Thus
$((\exists R.\phi)[C := \psi])^{\mathcal{I}} = (\exists R.(\phi[C := \psi]))^{\mathcal{I}}$.

71 As $(\exists R^-.\phi)^{\mathcal{I}} = \{x \mid \exists y.(y,x) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\}$,
$((\exists R^-.\phi)[C := \psi])^{\mathcal{I}} = \{x \mid \exists y.(y,x) \in R^{\mathcal{I}} \wedge y \in \phi[C := \psi]^{\mathcal{I}}\}$ as $R^{\mathcal{I}}$ is independent of the valuation of C. Thus
$((\exists R^-.\phi)[C := \psi])^{\mathcal{I}} = (\exists R^-.(\phi[C := \psi]))^{\mathcal{I}}$.

72 As $(\exists R.\phi)^{\mathcal{I}} = \{x \mid \exists y.(x,y) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\}$,
$((\exists R.\phi)[R' := R' \pm (i,j)])^{\mathcal{I}} = \{x \mid \exists y.(x,y) \in R^{\mathcal{I}} \wedge$
$y \in \phi[R' := R' \pm (i,j)]^{\mathcal{I}}\}$ as $R^{\mathcal{I}}$ is independent of the valuation of R'. Thus $((\exists R.\phi)[R' := R' \pm (i,j)])^{\mathcal{I}} = (\exists R.(\phi[R' := R' \pm (i,j)]))^{\mathcal{I}}$

73 As $(\exists R^-.\phi)^{\mathcal{I}} = \{x \mid \exists y.(y,x) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\}$,
$((\exists R^-.\phi)[R' := R' \pm (i,j)])^{\mathcal{I}} = \{x \mid \exists y.(y,x) \in R^{\mathcal{I}} \wedge$
$y \in \phi[R' := R' \pm (i,j)]^{\mathcal{I}}\}$ as $R^{\mathcal{I}}$ is independent of the valuation of R'. Thus $((\exists R^-.\phi)[R' := R' \pm (i,j)])^{\mathcal{I}} = (\exists R^-.(\phi[R' := R' \pm (i,j)]))^{\mathcal{I}}$

74 As $(\exists R.\phi)^{\mathcal{I}} = \{x \mid \exists y.(x,y) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\}$,
$((\exists R.\phi)[R := R + (i,j)])^{\mathcal{I}} = \{x \mid \exists y.(x,y) \in R[R := R + (i,j)]^{\mathcal{I}} \wedge$
$y \in \phi[R := R + (i,j)]^{\mathcal{I}}\}$ that is $((\exists R.\phi)[R := R + (i,j)])^{\mathcal{I}} =$
$\{x \mid \exists y.(x,y) \in R^{\mathcal{I}} \cup \{(i,j)\} \wedge y \in \phi[R := R + (i,j)]^{\mathcal{I}}\} =$
$\{x \mid (\exists y.(x,y) \in R^{\mathcal{I}} \wedge y \in \phi[R := R + (i,j)]^{\mathcal{I}}) \vee (x = i \wedge$
$j \in \phi[R := R + (i,j)])\} = \{x \mid (x = i \Rightarrow \exists y.(x,y) \in R^{\mathcal{I}} \wedge$

$y \in \phi[R := R+(i,j)]^{\mathcal{I}} \vee j \in \phi[R := R+(i,j)]) \wedge (x \neq i \Rightarrow \exists y.(x,y) \in R^{\mathcal{I}} \wedge$
$y \in \phi[R := R+(i,j)]^{\mathcal{I}})\}$. Moreover, $((\{i\} \Rightarrow \exists R.(\phi[R := R+(i,j)]) \sqcup$
$\exists U.(\{j\} \sqcap \phi[R := R+(i,j)])) \sqcap (\neg\{i\} \Rightarrow \exists R.(\phi[R := R+(i,j)])))^{\mathcal{I}} =$
$\{x \mid (x = i \Rightarrow \exists y.(x,y) \in R^{\mathcal{I}} \wedge y \in \phi[R := R+(i,j)]^{\mathcal{I}} \vee$
$j \in \phi[R := R+(i,j)]) \wedge (x \neq i \Rightarrow \exists y.(x,y) \in R^{\mathcal{I}} \wedge y \in \phi[R := R+(i,j)]^{\mathcal{I}})\}$.
Thus $((\exists R.\phi)[R := R+(i,j)])^{\mathcal{I}} =$
$((\{i\} \Rightarrow \exists R.(\phi[R := R+(i,j)]) \sqcup \exists U.(\{j\} \sqcap \phi[R := R+(i,j)])) \sqcap$
$(\neg\{i\} \Rightarrow \exists R.(\phi[R := R+(i,j)])))^{\mathcal{I}}$

75 One can see that $(\exists R^{-}.\phi)[R := R+(i,j)]$ is similar to $(\exists R^{-}.\phi)[R^{-} := R^{-}+(j,i)]$, that is replacing R by R^{-} and swapping i and j in the previous case.

76 As $(\exists R.\phi)^{\mathcal{I}} = \{x \mid \exists y.(x,y) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\}$,
$((\exists R.\phi)[R := R-(i,j)])^{\mathcal{I}} = \{x \mid \exists y.(x,y) \in R[R := R-(i,j)]^{\mathcal{I}} \wedge$
$y \in \phi[R := R-(i,j)]^{\mathcal{I}}\}$ that is $((\exists R.\phi)[R := R-(i,j)])^{\mathcal{I}} =$
$\{x \mid \exists y.(x,y) \in R^{\mathcal{I}} \cap \overline{\{(i,j)\}} \wedge y \in \phi[R := R-(i,j)]^{\mathcal{I}}\} =$
$\{x \mid \exists y.(x \neq i \vee y \neq j) \wedge ((x,y) \in R^{\mathcal{I}} \wedge y \in \phi[R := R-(i,j)]^{\mathcal{I}})\} = \{x \mid (x =$
$i \Rightarrow \exists y.(x,y) \in R^{\mathcal{I}} \wedge y \in \phi[R := R-(i,j)]^{\mathcal{I}} \wedge y \neq j) \wedge$
$(x \neq i \Rightarrow \exists y.(x,y) \in R^{\mathcal{I}} \wedge y \in \phi[R := R-(i,j)]^{\mathcal{I}})\}$. Moreover,
$((\{i\} \Rightarrow \exists R.(\phi[R := R-(i,j)] \sqcup \neg\{j\})) \sqcap (\neg\{i\} \Rightarrow \exists R.(\phi[R := R-$
$(i,j)])))^{\mathcal{I}} = \{x \mid (x = i \Rightarrow \exists y.(x,y) \in R^{\mathcal{I}} \wedge y \in \phi[R := R-(i,j)]^{\mathcal{I}} \wedge y \neq j) \wedge$
$(x \neq i \Rightarrow \exists y.(x,y) \in R^{\mathcal{I}} \wedge y \in \phi[R := R-(i,j)]^{\mathcal{I}})\}$. Thus
$((\exists R.\phi)[R := R-(i,j)])^{\mathcal{I}} =$
$((\{i\} \Rightarrow \exists R.(\phi[R := R-(i,j)] \sqcup \neg\{j\})) \sqcap (\neg\{i\} \Rightarrow \exists R.(\phi[R := R-(i,j)])))^{\mathcal{I}}$

77 One can see that $(\exists R^{-}.\phi)[R := R-(i,j)]$ is similar to $(\exists R^{-}.\phi)[R^{-} := R^{-}-(j,i)]$, that is replacing R by R^{-} and swapping i and j in the previous case.

78 As $(\exists R.\phi)^{\mathcal{I}} = \{x \mid \exists y.(x,y) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\}$,
$((\exists R.\phi)[R' := Q])^{\mathcal{I}} = \{x \mid \exists y.(x,y) \in R^{\mathcal{I}} \wedge$
$y \in \phi[R' := Q]^{\mathcal{I}}\}$ as $R^{\mathcal{I}}$ is independent of the valuation of R'. Thus
$((\exists R.\phi)[R' := Q])^{\mathcal{I}} = (\exists R.(\phi[R' := Q]))^{\mathcal{I}}$

79 As $(\exists R^{-}.\phi)^{\mathcal{I}} = \{x \mid \exists y.(y,x) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\}$,
$((\exists R^{-}.\phi)[R' := Q])^{\mathcal{I}} = \{x \mid \exists y.(y,x) \in R^{\mathcal{I}} \wedge$
$y \in \phi[R' := Q]^{\mathcal{I}}\}$ as $R^{\mathcal{I}}$ is independent of the valuation of R'. Thus
$((\exists R^{-}.\phi)[R' := Q])^{\mathcal{I}} = (\exists R^{-}.(\phi[R' := Q]))^{\mathcal{I}}$

80 As $(\exists R.\phi)^{\mathcal{I}} = \{x \mid \exists y.(x,y) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\}$,
$((\exists R.\phi)[R := Q])^{\mathcal{I}} = \{x \mid \exists y.(x,y) \in Q^{\mathcal{I}} \wedge$
$y \in \phi[R := Q]^{\mathcal{I}}\}$. Thus $((\exists R.\phi)[R := Q])^{\mathcal{I}} = (\exists Q.(\phi[R := Q]))^{\mathcal{I}}$

81 As $(\exists R^{-}.\phi)^{\mathcal{I}} = \{x \mid \exists y.(y,x) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\}$,
$((\exists R^{-}.\phi)[R := Q])^{\mathcal{I}} = \{x \mid \exists y.(y,x) \in Q^{\mathcal{I}} \wedge$
$y \in \phi[R := Q]^{\mathcal{I}}\}$. Thus $((\exists R^{-}.\phi)[R := Q])^{\mathcal{I}} = (\exists Q^{-}.(\phi[R := Q]))^{\mathcal{I}}$

82 As $(\exists R.\phi)^{\mathcal{I}} = \{x \mid \exists y.(x,y) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\}$, $((\exists R.\phi)[R' := (i,j)])^{\mathcal{I}} = \{x \mid \exists y.(x,y) \in R^{\mathcal{I}} \wedge y \in \phi[R' := (i,j)]^{\mathcal{I}}\}$ as $R^{\mathcal{I}}$ is independent of the valuation of R'. Thus $((\exists R.\phi)[R' := (i,j)])^{\mathcal{I}} = (\exists R.(\phi[R' := (i,j)]))^{\mathcal{I}}$

83 As $(\exists R^-.\phi)^{\mathcal{I}} = \{x \mid \exists y.(y,x) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\}$, $((\exists R^-.\phi)[R' := (i,j)])^{\mathcal{I}} = \{x \mid \exists y.(y,x) \in R^{\mathcal{I}} \wedge y \in \phi[R' := (i,j)]^{\mathcal{I}}\}$ as $R^{\mathcal{I}}$ is independent of the valuation of R'. Thus $((\exists R^-.\phi)[R' := (i,j)])^{\mathcal{I}} = (\exists R^-.(\phi[R' := (i,j)]))^{\mathcal{I}}$

84 As $(\exists R.\phi)^{\mathcal{I}} = \{x \mid \exists y.(x,y) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\}$, $((\exists R.\phi)[R := (i,j)])^{\mathcal{I}} = \{x \mid \exists y.x = i \wedge y = j \wedge y \in \phi[R := (i,j)]^{\mathcal{I}}\}$. Thus $((\exists R.\phi)[R := (i,j)])^{\mathcal{I}} = (\{i\} \sqcap \exists U.(\{j\} \sqcap \phi[R := (i,j)]))^{\mathcal{I}}$

85 As $(\exists R^-.\phi)^{\mathcal{I}} = \{x \mid \exists y.(y,x) \in R^{\mathcal{I}} \wedge y \in \phi^{\mathcal{I}}\}$, $((\exists R^-.\phi)[R := (i,j)])^{\mathcal{I}} = \{x \mid \exists y.x = j \wedge y = i \wedge y \in \phi[R := (i,j)]^{\mathcal{I}}\}$. Thus $((\exists R^-.\phi)[R := (i,j)])^{\mathcal{I}} = (\{j\} \sqcap \exists U.(\{i\} \sqcap \phi[R := (i,j)]))^{\mathcal{I}}$

86 As $\exists R.\phi = (\geq\ 1\ R\ \phi)$, it is possible to reuse the proof for rule 62 where $(\geq\ 2\ R\ \phi[i \gg^{in} j])$ is replaced with $\exists R.(\phi[i \gg^{in} j] \sqcap \neg\{i\})$ when it is known that $(x,i) \in R^{\mathcal{I}}$ and $i \in \phi[i \gg^{in} j]^{\mathcal{I}}$ and $(\geq\ 0\ R\ \phi[i \gg^{in} j])$ is replaced with $\top$.

87 As $\exists R^-.\phi = (\geq\ 1\ R^-\ \phi)$, it is possible to reuse the proof for rule 64 where $(\geq\ 1\ R^-\ (\phi[i \gg^{in} j] \sqcap \exists R.\{j\}))$ is replaced with $\exists R^-.\phi[i \gg^{in} j]$ as, if there exists y such that $(y,j) \in R^{\mathcal{I}}$, $(y,i) \in R^{\rangle}$ and $y \in \phi[i \gg^{in} j]$, then $\exists R^-.\phi[i \gg^{in} j]$ is satisfied in y.

88 As $\exists R.\phi = (\geq\ 1\ R\ \phi)$, it is possible to reuse the proof for rule 65 where $(\geq\ 1\ R\ (\phi[i \gg^{out} j] \sqcap \neg\{j\}))$ is replaced with $\exists R.\phi[i \gg^{out} j]$ as, if there exists y such that $(j,y) \in R^{\mathcal{I}}$, $(i,y) \in R^{\rangle}$ and $y \in \phi[i \gg^{out} j]$, then $\exists R.\phi[i \gg^{out} j]$ is satisfied in y.

89 As $\exists R^-.\phi = (\geq\ 1\ R^-\ \phi)$, it is possible to reuse the proof for rule 67 where $(\geq\ 2\ R^-\ \phi[i \gg^{out} j])$ is replaced with $\exists R^-.(\phi[i \gg^{out} j] \sqcap \neg\{i\})$ when it is known that $(x,i) \in R^{\mathcal{I}}$ and $i \in \phi[i \gg^{out} j]^{\mathcal{I}}$ and $(\geq\ 0\ R^-\ \phi[i \gg^{out} j])$ is replaced with $\top$.

90 As $(\forall R.\phi)^{\mathcal{I}} = \{x \mid \forall y.(x,y) \in R^{\mathcal{I}} \Rightarrow y \in \phi^{\mathcal{I}}\}$, $((\forall R.\phi)[C := C \pm i])^{\mathcal{I}} = \{x \mid \forall y.(x,y) \in R^{\mathcal{I}} \Rightarrow y \in \phi[C := C \pm i]^{\mathcal{I}}\}$ as $R^{\mathcal{I}}$ is independent of the valuation of C. Thus, $((\forall R.\phi)[C := C \pm i])^{\mathcal{I}} = (\forall R.(\phi[C := C \pm i]))^{\mathcal{I}}$

91 As $(\forall R^-.\phi)^{\mathcal{I}} = \{x \mid \forall y.(y,x) \in R^{\mathcal{I}} \Rightarrow y \in \phi^{\mathcal{I}}\}$, $((\forall R^-.\phi)[C := C \pm i])^{\mathcal{I}} = \{x \mid \forall y.(y,x) \in R^{\mathcal{I}} \Rightarrow y \in \phi[C := C \pm i]^{\mathcal{I}}\}$ as $R^{\mathcal{I}}$ is independent of the valuation of C. Thus, $((\forall R^-.\phi)[C := C \pm i])^{\mathcal{I}} = (\forall R^-.(\phi[C := C \pm i]))^{\mathcal{I}}$

92 As $(\forall R.\phi)^{\mathcal{I}} = \{x \mid \forall y.(x,y) \in R^{\mathcal{I}} \Rightarrow y \in \phi^{\mathcal{I}}\}$, $((\forall R.\phi)[C := \psi])^{\mathcal{I}} = \{x \mid \forall y.(x,y) \in R^{\mathcal{I}} \Rightarrow y \in \phi[C := \psi]^{\mathcal{I}}\}$ as $R^{\mathcal{I}}$ is independent of the valuation of C. Thus, $((\forall R.\phi)[C := \psi])^{\mathcal{I}} = (\forall R.(\phi[C := \psi]))^{\mathcal{I}}$

93 As $(\forall R^{-}.\phi)^{\mathcal{I}} = \{x \mid \forall y.(y,x) \in R^{\mathcal{I}} \Rightarrow y \in \phi^{\mathcal{I}}\}$, $((\forall R^{-}.\phi)[C := \psi])^{\mathcal{I}} = \{x \mid \forall y.(y,x) \in R^{\mathcal{I}} \Rightarrow y \in \phi[C := \psi]^{\mathcal{I}}\}$ as $R^{\mathcal{I}}$ is independent of the valuation of C. Thus, $((\forall R^{-}.\phi)[C := \psi])^{\mathcal{I}} = (\forall R^{-}.(\phi[C := \psi]))^{\mathcal{I}}$

94 As $(\forall R.\phi)^{\mathcal{I}} = \{x \mid \forall y.(x,y) \in R^{\mathcal{I}} \Rightarrow y \in \phi^{\mathcal{I}}\}$, $((\forall R.\phi)[R' := R' \pm (i,j)])^{\mathcal{I}} = \{x \mid \forall y.(x,y) \in R^{\mathcal{I}} \Rightarrow y \in \phi[R' := R' \pm (i,j)]^{\mathcal{I}}\}$ as $R^{\mathcal{I}}$ is independent of the valuation of R'. Thus,$((\forall R.\phi)[R' := R' \pm (i,j)])^{\mathcal{I}} = (\forall R.(\phi[R' := R' \pm (i,j)]))^{\mathcal{I}}$

95 As $(\forall R^{-}.\phi)^{\mathcal{I}} = \{x \mid \forall y.(y,x) \in R^{\mathcal{I}} \Rightarrow y \in \phi^{\mathcal{I}}\}$,
$((\forall R^{-}.\phi)[R' := R' \pm (i,j)])^{\mathcal{I}} = \{x \mid \forall y.(y,x) \in R^{\mathcal{I}} \Rightarrow$
$y \in \phi[R' := R' \pm (i,j)]^{\mathcal{I}}\}$ as $R^{\mathcal{I}}$ is independent of the valuation of R'.
Thus,$((\forall R^{-}.\phi)[R' := R' \pm (i,j)])^{\mathcal{I}} = (\forall R^{-}.(\phi[R' := R' \pm (i,j)]))^{\mathcal{I}}$

96 As $(\forall R.\phi)^{\mathcal{I}} = \{x \mid \forall y.(x,y) \in R^{\mathcal{I}} \Rightarrow y \in \phi^{\mathcal{I}}\}$,
$((\forall R.\phi)[R := R + (i,j)])^{\mathcal{I}} = \{x \mid \forall y.(x,y) \in R^{\mathcal{I}} \cup \{(i,j)\} \Rightarrow$
$y \in \phi[R := R + (i,j)]^{\mathcal{I}}\}$. That is $((\forall R.\phi)[R := R + (i,j)])^{\mathcal{I}} =$
$\{x \mid (\forall y.(x,y) \in R^{\mathcal{I}} \Rightarrow y \in \phi[R := R + (i,j)]^{\mathcal{I}}) \wedge$
$(x = i \Rightarrow j \in \phi[R := R + (i,j)]^{\mathcal{I}})\} = \{x \mid (x = i \Rightarrow (\forall y.(x,y) \in R^{\mathcal{I}} \Rightarrow$
$y \in \phi[R := R + (i,j)]^{\mathcal{I}} \wedge j \in (\phi[R := R + (i,j)])^{\mathcal{I}})) \wedge$
$(x \neq i \Rightarrow \forall y.(x,y) \in R^{\mathcal{I}} \Rightarrow y \in \phi[R := R + (i,j)]^{\mathcal{I}})\}$. Moreover,
$((\{i\} \Rightarrow \forall R.(\phi[R := R + (i,j)]) \sqcap \exists R.(\{j\} \sqcap \phi[R := R + (i,j)])) \sqcap$
$(\neg\{i\} \Rightarrow \forall R.(\phi[R := R + (i,j)]))^{\mathcal{I}} = \{x \mid (x = i \Rightarrow (\forall y.(x,y) \in R^{\mathcal{I}} \Rightarrow$
$y \in \phi[R := R + (i,j)]^{\mathcal{I}} \wedge j \in (\phi[R := R + (i,j)])^{\mathcal{I}})) \wedge (x \neq i \Rightarrow$
$\forall y.(x,y) \in R^{\mathcal{I}} \Rightarrow y \in \phi[R := R + (i,j)]^{\mathcal{I}})\}$. Thus,
$((\forall R.\phi)[R := R + (i,j)])^{\mathcal{I}} = ((\{i\} \Rightarrow \forall R.(\phi[R := R + (i,j)]) \sqcap$
$\exists R.(\{j\} \sqcap \phi[R := R + (i,j)])) \sqcap (\neg\{i\} \Rightarrow \forall R.(\phi[R := R + (i,j)]))^{\mathcal{I}}$

97 One can see that $(\forall R^{-}.\phi)[R := R + (i,j)]$ is similar to $(\forall R^{-}.\phi)[R^{-} := R^{-} + (j,i)]$, that is replacing R by R^{-} and swapping i and j in the previous case.

98 As $(\forall R.\phi)^{\mathcal{I}} = \{x \mid \forall y.(x,y) \in R^{\mathcal{I}} \Rightarrow y \in \phi^{\mathcal{I}}\}$,
$((\forall R.\phi)[R := R - (i,j)])^{\mathcal{I}} = \{x \mid \forall y.(x,y) \in R^{\mathcal{I}} \cap \overline{\{(i,j)\}} \Rightarrow$
$y \in \phi[R := R - (i,j)]^{\mathcal{I}}\}$. That is $((\forall R.\phi)[R := R - (i,j)])^{\mathcal{I}} =$
$\{x \mid (x \neq i \wedge \forall y.((x,y) \in R^{\mathcal{I}} \Rightarrow y \in \phi[R := R - (i,j)]^{\mathcal{I}})) \vee$
$(x = i \wedge \forall y.((x,y) \in R^{\mathcal{I}} \Rightarrow y \in \phi[R := R - (i,j)]^{\mathcal{I}} \vee y = j))\} =$
$\{x \mid \forall y.(x \neq i \wedge (x,y) \in R^{\mathcal{I}} \Rightarrow y \in \phi[R := R - (i,j)]^{\mathcal{I}})\} \cap$
$\{x \mid \forall y.(x = i \wedge (x,y) \in R^{\mathcal{I}} \Rightarrow y \in \phi[R := R - (i,j)]^{\mathcal{I}} \vee y = j)\}$. Moreover,
$(\{i\} \Rightarrow \forall R.(\{j\} \sqcup \phi[R := R - (i,j)]))^{\mathcal{I}} = \{x \mid \forall y.(x = i \wedge$
$(x,y) \in R^{\mathcal{I}} \Rightarrow y \in \phi[R := R - (i,j)]^{\mathcal{I}} \vee y = j)\}$ and
$(\neg\{i\} \Rightarrow \forall R.(\phi[R := R - (i,j)]))^{\mathcal{I}} = \{x \mid \forall y.(x \neq i \wedge (x,y) \in R^{\mathcal{I}} \Rightarrow$
$y \in \phi[R := R - (i,j)]^{\mathcal{I}})\}$. Thus, $((\forall R.\phi)[R := R - (i,j)])^{\mathcal{I}} =$
$((\{i\} \Rightarrow \forall R.(\phi[R := R - (i,j)] \sqcup \{j\})) \sqcap (\neg\{i\} \Rightarrow \forall R.(\phi[R := R - (i,j)])))^{\mathcal{I}}$

99 One can see that $(\forall R^-.\phi)[R := R - (i,j)]$ is similar to $(\forall R^-.\phi)[R^- := R^- - (j,i)]$, that is replacing R by R^- and swapping i and j in the previous case.

100 As $(\forall R.\phi)^{\mathcal{I}} = \{x \mid \forall y.(x,y) \in R^{\mathcal{I}} \Rightarrow y \in \phi^{\mathcal{I}}\}$, $((\forall R.\phi)[R' := Q])^{\mathcal{I}} = \{x \mid \forall y.(x,y) \in R^{\mathcal{I}} \Rightarrow y \in \phi[R' := Q]^{\mathcal{I}}\}$ as $R^{\mathcal{I}}$ is independent of the valuation of R'. Thus $((\forall R.\phi)[R' := Q])^{\mathcal{I}} = (\forall R.(\phi[R' := Q]))^{\mathcal{I}}$

101 As $(\forall R^-.\phi)^{\mathcal{I}} = \{x \mid \forall y.(y,x) \in R^{\mathcal{I}} \Rightarrow y \in \phi^{\mathcal{I}}\}$, $((\forall R^-.\phi)[R' := Q])^{\mathcal{I}} = \{x \mid \forall y.(y,x) \in R^{\mathcal{I}} \Rightarrow y \in \phi[R' := Q]^{\mathcal{I}}\}$ as $R^{\mathcal{I}}$ is independent of the valuation of R'. Thus $((\forall R^-.\phi)[R' := Q])^{\mathcal{I}} = (\forall R^-.(\phi[R' := Q]))^{\mathcal{I}}$

102 As $(\forall R.\phi)^{\mathcal{I}} = \{x \mid \forall y.(x,y) \in R^{\mathcal{I}} \Rightarrow y \in \phi^{\mathcal{I}}\}$, $((\forall R.\phi)[R := Q])^{\mathcal{I}} = \{x \mid \forall y.(x,y) \in Q^{\mathcal{I}} \Rightarrow y \in \phi[R := Q]^{\mathcal{I}}\}$. Thus $((\forall R.\phi)[R := Q])^{\mathcal{I}} = (\forall Q.(\phi[R := Q]))^{\mathcal{I}}$

103 As $(\forall R^-.\phi)^{\mathcal{I}} = \{x \mid \forall y.(y,x) \in R^{\mathcal{I}} \Rightarrow y \in \phi^{\mathcal{I}}\}$, $((\forall R^-.\phi)[R := Q])^{\mathcal{I}} = \{x \mid \forall y.(y,x) \in Q^{\mathcal{I}} \Rightarrow y \in \phi[R := Q]^{\mathcal{I}}\}$. Thus $((\forall R^-.\phi)[R := Q])^{\mathcal{I}} = (\forall Q^-.(\phi[R := Q]))^{\mathcal{I}}$

104 As $(\forall R.\phi)^{\mathcal{I}} = \{x \mid \forall y.(x,y) \in R^{\mathcal{I}} \Rightarrow y \in \phi^{\mathcal{I}}\}$, $((\forall R.\phi)[R' := (i,j)])^{\mathcal{I}} = \{x \mid \forall y.(x,y) \in R^{\mathcal{I}} \Rightarrow y \in \phi[R' := (i,j)]^{\mathcal{I}}\}$ as $R^{\mathcal{I}}$ is independent of the valuation of R'. Thus $((\forall R.\phi)[R' := (i,j)])^{\mathcal{I}} = (\forall R.(\phi[R' := (i,j)]))^{\mathcal{I}}$

105 As $(\forall R^-.\phi)^{\mathcal{I}} = \{x \mid \forall y.(y,x) \in R^{\mathcal{I}} \Rightarrow y \in \phi^{\mathcal{I}}\}$, $((\forall R^-.\phi)[R' := (i,j)])^{\mathcal{I}} = \{x \mid \forall y.(y,x) \in R^{\mathcal{I}} \Rightarrow y \in \phi[R' := (i,j)]^{\mathcal{I}}\}$ as $R^{\mathcal{I}}$ is independent of the valuation of R'. Thus $((\forall R^-.\phi)[R' := (i,j)])^{\mathcal{I}} = (\forall R^-.(\phi[R' := (i,j)]))^{\mathcal{I}}$

106 As $(\forall R.\phi)^{\mathcal{I}} = \{x \mid \forall y.(x,y) \in R^{\mathcal{I}} \Rightarrow y \in \phi^{\mathcal{I}}\}$, $((\forall R.\phi)[R := (i,j)])^{\mathcal{I}} = \{x \mid \forall y.(x = i \wedge y = j) \Rightarrow y \in \phi[R := (i,j)]^{\mathcal{I}}\}$. Thus $((\forall R.\phi)[R := (i,j)])^{\mathcal{I}} = ((\{i\} \sqcap \exists U.(\{j\} \sqcap \phi[R := (i,j)])) \sqcup \neg\{i\})^{\mathcal{I}}$

107 As $(\forall R^-.\phi)^{\mathcal{I}} = \{x \mid \forall y.(y,x) \in R^{\mathcal{I}} \Rightarrow y \in \phi^{\mathcal{I}}\}$, $((\forall R^-.\phi)[R := (i,j)])^{\mathcal{I}} = \{x \mid \forall y.(x = j \wedge y = i) \Rightarrow y \in \phi[R := (i,j)]^{\mathcal{I}}\}$. Thus $((\forall R^-.\phi)[R := (i,j)])^{\mathcal{I}} = ((\{j\} \sqcap \exists U.(\{i\} \sqcap \phi[R := (i,j)])) \sqcup \neg\{j\})^{\mathcal{I}}$

108 As $\forall R.\phi = (<\ 1\ R\ \neg\phi)$, it is possible to reuse the proof for rule 62 where $(<\ 2\ R\ (\neg\phi)[i \gg^{in} j])$ is replaced with $\forall R.(\phi[i \gg^{in} j] \sqcup \{i\})$ when it is known that $(x,i) \in R^{\mathcal{I}}$ and $i \in \phi[i \gg^{in} j]^{\mathcal{I}}$ and $(<\ 0\ R\ \phi[i \gg^{in} j])$ is replaced with $\bot$.

109 As $\forall R^-.\phi = (<\ 1\ R^-\ \neg\phi)$, it is possible to reuse the proof for rule 63 where $(<\ 1\ R^-\ (\phi[i \gg^{in} j] \wedge \neg\exists R.\{j\}))$ is replaced with $\forall R^-.\phi[i \gg^{in} j]$

as, if there exists y such that $(y, j) \in R^{\mathcal{I}}$, $(y, i) \in R^{\rangle}$ and $y \notin \phi[i \gg^{in} j]$, then $\forall R^{-}.\phi[i \gg^{in} j]$ cannot be satisfied in y.

110 As $\forall R.\phi = (<\ 1\ R\ \neg\phi)$, it is possible to reuse the proof for rule 66 where $(<\ 1\ R\ (\phi[i \gg^{out} j] \wedge \exists R.\{j\}))$ is replaced with $\forall R.\phi[i \gg^{out} j]$ as, if there exists y such that $(j, y) \in R^{\mathcal{I}}$, $(i, y) \in R^{\rangle}$ and $y \notin \phi[i \gg^{out} j]$, then $\forall R.\phi[i \gg^{out} j]$ cannot be satisfied in y.

111 As $\forall R^{-}.\phi = (<\ 1\ R^{-}\ \neg\phi)$, it is possible to reuse the proof for rule 67 where $(<\ 2\ R^{-}\ (\neg\phi[i \gg^{out} j]))$ is replaced with $\forall R^{-}.(\phi[i \gg^{out} j] \sqcup \{i\})$ when it is known that $(x, i) \in R^{\mathcal{I}}$ and $i \in \phi[i \gg^{out} j]^{\mathcal{I}}$ and $(<\ 0\ R^{-}\ (\neg\phi)[i \gg^{out} j])$ is replaced with $\bot$.

□

Lemma 2. *Let Σ be a signature, $\mathcal{T}$ is terminating.*

Proof. To prove the termination, we introduce a pre-ordering $(\mathcal{M}, \mathcal{M}')$ defined as:

- $\mathcal{M}(\bot) = 0$
- $\mathcal{M}(C) = 0$
- $\mathcal{M}(\neg\ \phi) = \mathcal{M}(\phi)$
- $\mathcal{M}(\phi\ \sqcap\ \psi) = max(\mathcal{M}(\phi), \mathcal{M}(\psi))$
- $\mathcal{M}(@_a\phi) = \mathcal{M}(\phi)$
- $\mathcal{M}(\phi\ \sqcup\ \psi) = max(\mathcal{M}(\phi), \mathcal{M}(\psi))$
- $\mathcal{M}((\geq n\ R\ \phi)) = \mathcal{M}((\geq n\ R^{-}\ \phi)) = \mathcal{M}(\phi)$
- $\mathcal{M}((< n\ R\ \phi)) = \mathcal{M}((< n\ R^{-}\ \phi)) = \mathcal{M}(\phi)$
- $\mathcal{M}(\exists R.\phi) = \mathcal{M}(\exists R^{-}.\phi) = \mathcal{M}(\phi)$
- $\mathcal{M}(\forall R.\phi) = \mathcal{M}(\forall R^{-}.\phi) = \mathcal{M}(\phi)$
- $\mathcal{M}(o) = 0$
- $\mathcal{M}(\exists R.Self) = \mathcal{M}(\exists R^{-}.Self) = 0$
- $\mathcal{M}(\phi\theta) = \mathcal{M}(\phi) + 1$
- $\mathcal{M}'(\bot) = 0$
- $\mathcal{M}'(C) = 0$
- $\mathcal{M}'((\neg\ \phi)) = \mathcal{M}'(\phi)$
- $\mathcal{M}'(\phi\ \sqcap\ \psi) = max(\mathcal{M}'(\phi), \mathcal{M}'(\psi))$
- $\mathcal{M}'(\phi\ \sqcup\ \psi) = max(\mathcal{M}'(\phi), \mathcal{M}'(\psi))$

- $\mathcal{M}'((@_a\phi)) = \mathcal{M}(\phi)$
- $\mathcal{M}'((\geq n\ R\ \phi)) = \mathcal{M}'((\geq n\ R^-\ \phi)) = \mathcal{M}'(\phi)$
- $\mathcal{M}'((< n\ R\ \phi)) = \mathcal{M}'((< n\ R^-\ \phi)) = \mathcal{M}'(\phi)$
- $\mathcal{M}'((\exists R.\phi)) = \mathcal{M}'((\exists R^-.\phi)) = \mathcal{M}'(\phi)$
- $\mathcal{M}'((\forall R.\phi)) = \mathcal{M}'((\forall R^-.\phi)) = \mathcal{M}'(\phi)$
- $\mathcal{M}'(o) = 0$
- $\mathcal{M}'(\exists R.Self) = \mathcal{M}'(\exists R^-.Self) = 0$
- $\mathcal{M}'(\bot\theta) = 0$
- $\mathcal{M}'(c\theta) = 0$
- $\mathcal{M}'((\neg\ \phi)\theta) = \mathcal{M}'(\phi\theta) + 1$
- $\mathcal{M}'((\phi\ \sqcap\ \psi)\theta) = max(\mathcal{M}'(\phi\theta), \mathcal{M}'(\psi\theta)) + 1$
- $\mathcal{M}'((\phi\ \sqcup\ \psi)\theta) = max(\mathcal{M}'(\phi\theta), \mathcal{M}'(\psi\theta)) + 1$
- $\mathcal{M}'((@_a\phi)\theta) = \mathcal{M}(\phi\theta) + 1$
- $\mathcal{M}'((\geq n\ R\ \phi)\theta) = \mathcal{M}'((\geq n\ R^-\ \phi)\theta) = \mathcal{M}'(\phi\theta) + 1$
- $\mathcal{M}'((< n\ R\ \phi)\theta) = \mathcal{M}'((< n\ R^-\ \phi)\theta) = \mathcal{M}'(\phi\theta) + 1$
- $\mathcal{M}'((\exists R.\phi)\theta) = \mathcal{M}'((\exists^- R.\phi)\theta) = \mathcal{M}'(\phi\theta) + 1$
- $\mathcal{M}'((\forall R.\phi)\theta) = \mathcal{M}'((\forall R^-.\phi)\theta) = \mathcal{M}'(\phi\theta) + 1$
- $\mathcal{M}'(o\theta) = 0$
- $\mathcal{M}'((\exists R.Self)\theta) = \mathcal{M}'((\exists R^-.Self)\theta) = 0$

For every concept ϕ, $\mathcal{M}(\phi)$ and $\mathcal{M}'(\phi)$ are positive. We now prove that the transformations either strictly decrease $\mathcal{M}$ or keep $\mathcal{M}$ constant and strictly decrease $\mathcal{M}'$. We compute the results of the functions for the left- and the right- hand side for each transformation.

1

$$\begin{array}{rcrcl} \mathcal{M}(\bot\ \theta) & = & \mathcal{M}(\bot) + 1 & = & 1 \\ \mathcal{M}(\bot) & & & = & 0 \end{array}$$

2

$$\begin{array}{rcrcl} \mathcal{M}(o\ \theta) & = & \mathcal{M}(o) + 1 & = & 1 \\ \mathcal{M}(o) & & & = & 0 \end{array}$$

3

$$\begin{array}{rcrcl} \mathcal{M}(C[R := R \pm (i,j)]) & = & \mathcal{M}(C) + 1 & = & 1 \\ \mathcal{M}(C) & & & = & 0 \end{array}$$

4

$$\begin{array}{rcccl} \mathcal{M}(C[C' := C' \pm i]) & = & \mathcal{M}(C) + 1 & = & 1 \\ \mathcal{M}(C) & & & = & 0 \end{array}$$

5

$$\begin{array}{rcccl} \mathcal{M}(C[R := Q]) & = & \mathcal{M}(C) + 1 & = & 1 \\ \mathcal{M}(C) & & & = & 0 \end{array}$$

6

$$\begin{array}{rcccl} \mathcal{M}(C[R := (i,j)]) & = & \mathcal{M}(C) + 1 & = & 1 \\ \mathcal{M}(C) & & & = & 0 \end{array}$$

7

$$\begin{array}{rcccl} \mathcal{M}(C[i \gg^{in} j]) & = & \mathcal{M}(C) + 1 & = & 1 \\ \mathcal{M}(C) & & & = & 0 \end{array}$$

8

$$\begin{array}{rcccl} \mathcal{M}(C[i \gg^{out} j]) & = & \mathcal{M}(C) + 1 & = & 1 \\ \mathcal{M}(C) & & & = & 0 \end{array}$$

9

$$\begin{array}{rcccl} \mathcal{M}(C[C := C + i]) & = & \mathcal{M}(C) + 1 & = & 1 \\ \mathcal{M}(C \sqcup \{i\}) & = & max(\mathcal{M}(C), \mathcal{M}(\{i\})) & = & 0 \end{array}$$

10

$$\begin{array}{rcccl} \mathcal{M}(C[C := C - i]) & = & \mathcal{M}(C) + 1 & = & 1 \\ \mathcal{M}(C \sqcap \neg\{i\}) & = & max(\mathcal{M}(C), \mathcal{M}(\neg\{i\})) & = & 0 \end{array}$$

11

$$\begin{array}{rcccl} \mathcal{M}(C[C' := \psi]) & = & \mathcal{M}(C) + 1 & = & 1 \\ \mathcal{M}(C) & & & = & 0 \end{array}$$

12

$$\begin{array}{rcccl} \mathcal{M}(C[C := \psi]) & = & \mathcal{M}(C) + 1 & = & 1 \\ \mathcal{M}(\psi) & & & = & 0^{2} \end{array}$$

13

$$\begin{array}{rcccl} \mathcal{M}((\neg\phi)\ \theta) & = & \mathcal{M}(\neg\phi) + 1 & = & \mathcal{M}(\phi) + 1 \\ \mathcal{M}(\neg(\phi\ \theta)) & = & \mathcal{M}(\phi\ \theta) & = & \mathcal{M}(\phi) + 1 \\ \mathcal{M}'((\neg\phi)\ \theta) & = & \mathcal{M}'(\phi\theta) + 1 & & \\ \mathcal{M}'(\neg(\phi\theta)) & = & \mathcal{M}'(\phi\theta) & & \end{array}$$

14

$$\begin{array}{rcl} \mathcal{M}((\phi \sqcup \psi)\ \theta) & = & max(\mathcal{M}(\phi), \mathcal{M}(\psi)) + 1 \\ \mathcal{M}(\phi\theta \sqcup \psi\theta) & = & max(\mathcal{M}(\phi) + 1, \mathcal{M}(\psi) + 1) \\ \mathcal{M}'((\phi \sqcup \psi)\ \theta) & = & max(\mathcal{M}'(\phi\theta), \mathcal{M}'(\psi\theta)) + 1 \\ \mathcal{M}'(\phi\theta \sqcup \psi\theta) & = & max(\mathcal{M}'(\phi\theta), \mathcal{M}'(\psi\theta)) \end{array}$$

15

$$\begin{array}{rcl} \mathcal{M}((\phi \sqcap \psi)\ \theta) & = & max(\mathcal{M}(\phi), \mathcal{M}(\psi)) + 1 \\ \mathcal{M}(\phi\theta \sqcap \psi\theta) & = & max(\mathcal{M}(\phi) + 1, \mathcal{M}(\psi) + 1) \\ \mathcal{M}'((\phi \sqcap \psi)\ \theta) & = & max(\mathcal{M}'(\phi\theta), \mathcal{M}'(\psi\theta)) + 1 \\ \mathcal{M}'(\phi\theta \sqcap \psi\theta) & = & max(\mathcal{M}'(\phi\theta), \mathcal{M}'(\psi\theta)) \end{array}$$

16

$$\begin{array}{rcl} \mathcal{M}((@_a\phi)\ \theta) & = & \mathcal{M}(@_a\phi)+1 \\ & & \mathcal{M}(\phi)+1 \\ \mathcal{M}(@_a(\phi\theta)) & = & \mathcal{M}(\phi\ \theta) \\ & & \mathcal{M}(\phi)+1 \\ \mathcal{M}'((@_a\phi)\ \theta) & = & \mathcal{M}'(\phi\ \theta)+1 \\ \mathcal{M}'(@_a(\phi\theta)) & = & \mathcal{M}(\phi\ \theta) \end{array}$$

17

$$\begin{array}{rcl} \mathcal{M}(\exists R.Self[C := C \pm i]) & = & \mathcal{M}(\exists R.Self)+1 \quad = \quad 1 \\ \mathcal{M}(\exists R.Self) & & \quad = \quad 0 \end{array}$$

18 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

19

$$\begin{array}{rcl} \mathcal{M}(\exists R.Self[C := \psi]) & = & \mathcal{M}(\exists R.Self)+1 \quad = \quad 1 \\ \mathcal{M}(\exists R.Self) & & \quad = \quad 0 \end{array}$$

20 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

21

$$\begin{array}{rcl} \mathcal{M}(\exists R.Self[R' := R' \pm (i,j)]) & = & \mathcal{M}(\exists R.Self)+1 \quad = \quad 1 \\ \mathcal{M}(\exists R.Self) & & \quad = \quad 0 \end{array}$$

22 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

23

$$\begin{array}{rcl} \mathcal{M}(\exists R.Self[R := R+(i,j)]) & = & \mathcal{M}(\exists R.Self)+1 \quad = \quad 1 \\ \mathcal{M}((\{i\} \sqcap \{j\}) \sqcup \exists R.Self) & = & 0 \end{array}$$

24 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

25

$$\begin{array}{rcl} C \pm i\mathcal{M}(\exists R.Self[R := R-(i,j)]) & = & \mathcal{M}(\exists R.Self)+1 \quad = \quad 1 \\ \mathcal{M}((\neg\{i\} \sqcup \neg\{j\}) \sqcap \exists R.Self) & & \quad = \quad 0 \end{array}$$

26 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

27

$$\begin{array}{rcl} \mathcal{M}(\exists R.Self[R' := Q]) & = & \mathcal{M}(\exists R.Self)+1 \quad = \quad 1 \\ \mathcal{M}(\exists R.Self) & & \quad = \quad 0 \end{array}$$

28 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

29

$$\begin{array}{rcl} \mathcal{M}(\exists R.Self[R := Q]) & = & \mathcal{M}(\exists R.Self)+1 \quad = \quad 1 \\ \mathcal{M}(\exists Q.Self) & & \quad = \quad 0 \end{array}$$

30 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^- and Q and Q', this rule is identic to the previous one.

31

$$\begin{array}{rcl} \mathcal{M}(\exists R.Self[R' := (i,j)]) & = & \mathcal{M}(\exists R.Self) + 1 \quad = \quad 1 \\ \mathcal{M}(\exists R.Self) & & = \quad 0 \end{array}$$

32 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

33

$$\begin{array}{rcl} \mathcal{M}(\exists R.Self[R := (i,j)]) & = & \mathcal{M}(\exists R.Self) + 1 \quad = \quad 1 \\ \mathcal{M}(\{i\} \sqcap \{j\}) & & = \quad 0 \end{array}$$

34 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

35

$$\begin{array}{rcl} \mathcal{M}(\exists R.Self[i \gg^{in} j]) & = & \mathcal{M}(\exists R.Self) + 1 \\ & = & 1 \\ \mathcal{M}(RHS_{35}) & = & max(\mathcal{M}(\{i\}, \mathcal{M}(\{j\}), \mathcal{M}(\exists R.Self), \mathcal{M}(\exists R.i)) \\ & = & max(0, \mathcal{M}(\{i\})) \\ & = & 0 \end{array}$$

36 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

37

$$\begin{array}{rcl} \mathcal{M}(\exists R.Self[i \gg^{out} j]) & = & \mathcal{M}(\exists R.Self) + 1 \\ & = & 1 \\ \mathcal{M}(RHS_{37}) & = & max(\mathcal{M}(\{i\}, \mathcal{M}(\{j\}), \mathcal{M}(\exists R.Self), \mathcal{M}(\exists R^-.i)) \\ & = & max(0, \mathcal{M}(\{i\})) \\ & = & 0 \end{array}$$

38 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

39

$$\begin{array}{rcl} \mathcal{M}((\bowtie\ n\ R\ \phi)[C := C \pm i]) & = & \mathcal{M}((\bowtie\ n\ R\ \phi)) + 1 \\ & = & \mathcal{M}(\phi) + 1 \\ \mathcal{M}((\bowtie\ n\ R\ \phi[C := C \pm i])) & = & \mathcal{M}(\phi[C := C \pm i]) \\ & = & \mathcal{M}(\phi) + 1 \\ \mathcal{M}'((\bowtie\ n\ R\ \phi)[C := C \pm i]) & = & \mathcal{M}'(\phi[C := C \pm i]) + 1 \\ \mathcal{M}'((\bowtie\ n\ R\ \phi[C := C \pm i])) & = & \mathcal{M}'(\phi[C := C \pm i]) \end{array}$$

40 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

41

$$\begin{array}{rcl}
\mathcal{M}((\bowtie\ n\ R\ \phi)[C := \psi]) & = & \mathcal{M}((\bowtie\ n\ R\ \phi)) + 1 \\
& = & \mathcal{M}(\phi) \\
\mathcal{M}((\bowtie\ n\ R\ \phi[C := \psi])) & = & \mathcal{M}(\phi[C := \psi]) \\
& = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}'((\bowtie\ n\ R\ \phi)[C := C \pm i]) & = & \mathcal{M}'(\phi[C := \psi]) + 1 \\
\mathcal{M}'((\bowtie\ n\ R\ \phi[C := C \pm i])) & = & \mathcal{M}'(\phi[C := \psi])
\end{array}$$

42 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

43

$$\begin{array}{rcl}
\mathcal{M}((\bowtie\ n\ R\ \phi)[R' := R' \pm (i,j)]) & = & \mathcal{M}((\bowtie\ n\ R\ \phi)) + 1 \\
& = & \mathcal{M}(\phi) + 2 \\
\mathcal{M}((\bowtie\ n\ R\ \phi[R' := R' \pm (i,j)]) & = & \mathcal{M}(\phi[R' := R' \pm (i,j)]) + 1 \\
& = & \mathcal{M}(\phi) + 2 \\
\mathcal{M}'((\bowtie\ n\ R\ \phi)[R' := R' \pm (i,j)]) & = & \mathcal{M}'(\phi[R' := R' \pm (i,j)]) + 1 \\
\mathcal{M}'((\bowtie\ n\ R\ \phi[R' := R' \pm (i,j)])) & = & \mathcal{M}'(\phi[R' := R' \pm (i,j)])
\end{array}$$

44 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

45

$$\begin{array}{rcl}
\mathcal{M}((\bowtie\ n\ R\ \phi)[R := R + (i,j)]) & = & \mathcal{M}((\bowtie\ n\ R\ \phi) + 1 \\
& = & \mathcal{M}(\phi) + 2
\end{array}$$

$$
\begin{array}{rcl}
\mathcal{M}(c_{1,16}) & = & max(\mathcal{M}(\{j\}), \mathcal{M}(\phi[R := R + (i,j)])) \\
& = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(c_{2,16}) & = & max(\mathcal{M}(\neg\{j\}), \mathcal{M}(\neg(\phi[R := R + (i,j)]))) \\
& = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(b_{1,16}) & = & \mathcal{M}(\{i\}) \\
& = & 0 \\
\mathcal{M}(b_{2,16}) & = & \mathcal{M}(\exists U.c_{1,16}) \\
& = & \mathcal{M}(c_{1,16}) + 1 \\
& = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(b_{3,16}) & = & \mathcal{M}(\forall R.\neg\{j\}) \\
& = & \mathcal{M}(\neg\{j\}) + 1 \\
& = & 1 \\
\mathcal{M}(b_{4,16}) & = & \mathcal{M}((\bowtie\ (n-1)\ R\ \phi[R := R + (i,j)])) \\
& = & \mathcal{M}(\phi[R := R + (i,j)]) + 1 \\
& = & \mathcal{M}(\phi) + 2 \\
\mathcal{M}(b_{11,16}) & = & \mathcal{M}(\neg\{i\}) \\
& = & 0 \\
\mathcal{M}(b_{12,16}) & = & \mathcal{M}(\forall U.c_{2,16}) \\
& = & \mathcal{M}(c_{2,16}) + 1 \\
& = & \mathcal{M}(\phi) + 2 \\
\mathcal{M}(b_{13,16}) & = & \mathcal{M}(\exists R.\{j\}) \\
& = & \mathcal{M}(\{j\}) + 1 \\
& = & 1 \\
\mathcal{M}(b_{14,16}) & = & \mathcal{M}((\bowtie\ n\ R\ \phi[R := R + (i,j)])) \\
& = & \mathcal{M}(\phi[R := R + (i,j)]) + 1 \\
& = & \mathcal{M}(\phi) + 2 \\
\mathcal{M}(a_{1,16}) & = & max(\mathcal{M}(b_{1,16}), \mathcal{M}(b_{2,16}), \mathcal{M}(b_{3,16}), \mathcal{M}(b_{4,16}) \\
& = & \mathcal{M}(\phi) + 2 \\
\mathcal{M}(a_{2,16}) & = & max(\mathcal{M}(b_{11,16}), \mathcal{M}(b_{12,16}), \mathcal{M}(b_{13,16}), \mathcal{M}(b_{14,16}) \\
& = & \mathcal{M}(\phi) + 2 \\
\mathcal{M}(RHR_{16}) & = & max(\mathcal{M}(a_{1,16}), \mathcal{M}(a_{2,16})) \\
& = & \mathcal{M}(\phi) + 2
\end{array}
$$

$$
\mathcal{M}'((\bowtie\ n\ R\ \phi)[R := R + (i,j)]) \quad = \quad \mathcal{M}'(\phi[R := R + (i,j)]) + 1
$$

$$
\begin{array}{rcl}
\mathcal{M}'(c_{1,16}) & = & max(\mathcal{M}'(\{j\}), \mathcal{M}'(\phi[R := R + (i,j)])) \\
& = & \mathcal{M}'(\phi[R := R + (i,j)]) \\
\mathcal{M}'(c_{2,16}) & = & max(\mathcal{M}'(\neg\{j\}), \mathcal{M}'(\neg(\phi[R := R + (i,j)]))) \\
& = & \mathcal{M}'(\phi[R := R + (i,j)]) \\
\mathcal{M}'(b_{1,16}) & = & \mathcal{M}(\{i\}) \\
& = & 0 \\
\mathcal{M}'(b_{2,16}) & = & \mathcal{M}(\exists U.c_{1,16}) \\
& = & \mathcal{M}'(c_{1,16}) \\
& = & \mathcal{M}'(\phi[R := R + (i,j)])
\end{array}
$$

$$
\begin{array}{rcc}
\mathcal{M}'(b_{3,16}) & = & \mathcal{M}(\forall R.\neg\{j\}) \\
 & = & \mathcal{M}'(\neg\{j\}) \\
 & = & 0 \\
\mathcal{M}'(b_{4,16}) & = & \mathcal{M}((\bowtie\ (n-1)\ R\ \phi[R := R + (i,j)]) \\
 & = & \mathcal{M}'(\phi[R := R + (i,j)]) \\
\mathcal{M}'(b_{11,16}) & = & \mathcal{M}'(\neg\{i\}) \\
 & = & 0 \\
\mathcal{M}'(b_{12,16}) & = & \mathcal{M}'(\forall U.c_{2,16}) \\
 & = & \mathcal{M}'(c_{2,16}) \\
 & = & \mathcal{M}(\phi[R := R + (i,j]) \\
\mathcal{M}'(b_{13,16}) & = & \mathcal{M}'(\exists R.\{j\}) \\
 & = & \mathcal{M}'(\{j\}) \\
 & = & 0 \\
\mathcal{M}'(b_{14,16}) & = & \mathcal{M}'((\bowtie\ n\ R\ \phi[R := R + (i,j)])) \\
 & = & \mathcal{M}'(\phi[R := R + (i,j)]) \\
\mathcal{M}'(a_{1,16}) & = & max(\mathcal{M}'(b_{1,16}), \mathcal{M}'(b_{2,16}), \mathcal{M}'(b_{3,16}), \mathcal{M}'(b_{4,16}) \\
 & = & \mathcal{M}'(\phi[R := R + (i,j)]) \\
\mathcal{M}'(a_{2,16}) & = & max(\mathcal{M}'(b_{11,16}), \mathcal{M}'(b_{12,16}), \mathcal{M}'(b_{13,16}), \mathcal{M}'(b_{14,16}) \\
 & = & \mathcal{M}'(\phi[R := R + (i,j)]) \\
\mathcal{M}'(RHR_{16}) & = & max(\mathcal{M}'(a_{1,16}), \mathcal{M}'(a_{2,16})) \\
 & = & \mathcal{M}(\phi[R := R + (i,j)])
\end{array}
$$

46 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

47

$$
\begin{array}{rcc}
\mathcal{M}((\bowtie\ n\ R\ \phi)[R := R - (i,j)]) & = & \mathcal{M}((\bowtie\ n\ R\ \phi) + 1 \\
 & = & \mathcal{M}(\phi) + 2 \\
\mathcal{M}(c_{1,17}) & = & max(\mathcal{M}(\{j\}), \mathcal{M}(\phi[R := R - (i,j)])) \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(c_{2,17}) & = & max(\mathcal{M}(\neg\{j\}), \mathcal{M}(\neg(\phi[R := R - (i,j)]))) \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(b_{1,17}) & = & \mathcal{M}(\{i\}) \\
 & = & 0 \\
\mathcal{M}(b_{2,17}) & = & \mathcal{M}(\exists U.c_{1,17}) \\
 & = & \mathcal{M}(c_{1,17}) + 1 \\
 & = & \mathcal{M}(\phi) + 2 \\
\mathcal{M}(b_{3,17}) & = & \mathcal{M}(\exists R.\{j\}) \\
 & = & \mathcal{M}(\neg\{j\}) + 1 \\
 & = & 1 \\
\mathcal{M}(b_{4,17}) & = & \mathcal{M}((\bowtie\ (n+1)\ R\ \phi[R := R - (i,j)])) \\
 & = & \mathcal{M}(\phi[R := R - (i,j)]) + 1 \\
 & = & \mathcal{M}(\phi) + 2 \\
\mathcal{M}(b_{11,17}) & = & \mathcal{M}(\neg\{i\}) \\
 & = & 0
\end{array}
$$

$$
\begin{array}{rcc}
\mathcal{M}(b_{12,17}) & = & \mathcal{M}(\forall U.c_{2,17}) \\
& = & \mathcal{M}(c_{2,17}) + 1 \\
& = & \mathcal{M}(\phi) + 2 \\
\mathcal{M}(b_{13,17}) & = & \mathcal{M}(\forall R.\neg\{j\}) \\
& = & \mathcal{M}(\{j\}) + 1 \\
& = & 1 \\
\mathcal{M}(b_{14,17}) & = & \mathcal{M}((\bowtie\ n\ R\ \phi[R := R + (i,j)])) \\
& = & \mathcal{M}(\phi[R := R - (i,j)]) + 1 \\
& = & \mathcal{M}(\phi) + 2 \\
\mathcal{M}(a_{1,17}) & = & max(\mathcal{M}(b_{1,17}), \mathcal{M}(b_{2,17}), \mathcal{M}(b_{3,17}), \mathcal{M}(b_{4,17}) \\
& = & \mathcal{M}(\phi) + 2 \\
\mathcal{M}(a_{2,17}) & = & max(\mathcal{M}(b_{11,17}), \mathcal{M}(b_{12,17}), \mathcal{M}(b_{13,17}), \mathcal{M}(b_{14,17}) \\
& = & \mathcal{M}(\phi) + 2 \\
\mathcal{M}(RHR_{17}) & = & max(\mathcal{M}(a_{1,17}), \mathcal{M}(a_{2,17})) \\
& = & \mathcal{M}(\phi) + 2 \\
\mathcal{M}'((\bowtie\ n\ R\ \phi)[R := R - (i,j)]) & = & \mathcal{M}'(\phi[R := R - (i,j)]) + 1 \\
\mathcal{M}'(c_{1,17}) & = & max(\mathcal{M}'(\{j\}), \mathcal{M}'(\phi[R := R - (i,j)])) \\
& = & \mathcal{M}'(\phi[R := R - (i,j)]) \\
\mathcal{M}'(c_{2,17}) & = & max(\mathcal{M}'(\neg\{j\}), \mathcal{M}'(\neg(\phi[R := R - (i,j)]))) \\
& = & \mathcal{M}'(\phi[R := R - (i,j)]) \\
\mathcal{M}'(b_{1,17}) & = & \mathcal{M}(\{i\}) \\
& = & 0 \\
\mathcal{M}'(b_{2,17}) & = & \mathcal{M}(\exists U.c_{1,17}) \\
& = & \mathcal{M}'(c_{1,17}) \\
& = & \mathcal{M}'(\phi[R := R - (i,j)]) \\
\mathcal{M}'(b_{3,17}) & = & \mathcal{M}(\forall R.\neg\{j\}) \\
& = & \mathcal{M}'(\neg\{j\}) \\
& = & 0 \\
\mathcal{M}'(b_{4,17}) & = & \mathcal{M}((\bowtie\ (n+1)\ R\ \phi[R := R - (i,j)]) \\
& = & \mathcal{M}'(\phi[R := R - (i,j)]) \\
\mathcal{M}'(b_{11,17}) & = & \mathcal{M}'(\neg\{i\}) \\
& = & 0 \\
\mathcal{M}'(b_{12,17}) & = & \mathcal{M}'(\forall U.c_{2,17}) \\
& = & \mathcal{M}'(c_{2,17}) \\
& = & \mathcal{M}(\phi[R := R - (i,j]) \\
\mathcal{M}'(b_{13,17}) & = & \mathcal{M}'(\forall R.\neg\{j\}) \\
& = & \mathcal{M}'(\{j\}) \\
& = & 0 \\
\mathcal{M}'(b_{14,17}) & = & \mathcal{M}'((\bowtie\ n\ R\ \phi[R := R - (i,j)])) \\
& = & \mathcal{M}'(\phi[R := R - (i,j)]) \\
\mathcal{M}'(a_{1,17}) & = & max(\mathcal{M}'(b_{1,17}), \mathcal{M}'(b_{2,17}), \mathcal{M}'(b_{3,17}), \mathcal{M}'(b_{4,17}) \\
& = & \mathcal{M}'(\phi[R := R - (i,j)])
\end{array}
$$

$$\begin{array}{rcl}
\mathcal{M}'(a_{2,17}) & = & max(\mathcal{M}'(b_{11,17}), \mathcal{M}'(b_{12,17}), \mathcal{M}'(b_{13,17}), \mathcal{M}'(b_{14,17}) \\
& = & \mathcal{M}'(\phi[R := R - (i,j)]) \\
\mathcal{M}'(RHR_{17}) & = & max(\mathcal{M}'(a_{1,17}), \mathcal{M}'(a_{2,17})) \\
& = & \mathcal{M}(\phi[R := R - (i,j)])
\end{array}$$

48 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

49

$$\begin{array}{rcl}
\mathcal{M}((\bowtie\ n\ R\ \phi)[R' := Q]) & = & \mathcal{M}((\bowtie\ n\ R\ \phi)) + 1 \\
& = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}((\bowtie\ n\ R\ \phi[R' := Q])) & = & \mathcal{M}(\phi[R' := Q]) \\
& = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}'((\bowtie\ n\ R\ \phi)[R' := Q]) & = & \mathcal{M}'(\phi[R' := Q]) + 1 \\
\mathcal{M}'((\bowtie\ n\ R\ \phi[R' := Q])) & = & \mathcal{M}'(\phi[R' := Q])
\end{array}$$

50 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

51

$$\begin{array}{rcl}
\mathcal{M}((\bowtie\ n\ R\ \phi)[R := Q]) & = & \mathcal{M}((\bowtie\ n\ R\ \phi)) + 1 \\
& = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}((\bowtie\ n\ Q\ \phi[R := Q])) & = & \mathcal{M}(\phi[R := Q]) \\
& = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}'((\bowtie\ n\ R\ \phi)[R := Q]) & = & \mathcal{M}'(\phi[R := Q]) + 1 \\
\mathcal{M}'((\bowtie\ n\ Q\ \phi[R := Q])) & = & \mathcal{M}'(\phi[R := Q])
\end{array}$$

52 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

53

$$\begin{array}{rcl}
\mathcal{M}((\bowtie\ n\ R\ \phi)[R' := (i,j)]) & = & \mathcal{M}((\bowtie\ n\ R\ \phi)) + 1 \\
& = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}((\bowtie\ n\ R\ \phi[R' := (i,j)])) & = & \mathcal{M}(\phi[R' := (i,j)]) \\
& = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}'((\bowtie\ n\ R\ \phi)[R' := (i,j)]) & = & \mathcal{M}'(\phi[R' := (i,j)]) + 1 \\
\mathcal{M}'((\bowtie\ n\ R\ \phi[R' := (i,j)])) & = & \mathcal{M}'(\phi[R' := (i,j)])
\end{array}$$

54

$$
\begin{array}{rcc}
\mathcal{M}((<\ n\ R\ \phi)[R := (i,j)]) & = & \mathcal{M}((<\ n\ R\ \phi)) + 1 \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(RHS_{54}) & = & max(\mathcal{M}(\neg\{i\}), \mathcal{M}(a_{1,54})) \\
 & = & max(\mathcal{M}(\{i\}), max(\mathcal{M}(\neg\{j\}), \\
 & & \mathcal{M}(\neg\phi[R\ :=\ (i,j)]))) \\
 & = & max(0, max(0, \mathcal{M}(\phi) + 1)) \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}'((<\ n\ R\ \phi)[R := (i,j)]) & = & \mathcal{M}'((<\ n\ R\ \phi[R := (i,j)])) + 1 \\
 & = & \mathcal{M}'(\phi[R := (i,j)]) + 1 \\
\mathcal{M}'(RHS_{54}) & = & max(\mathcal{M}'(\neg\{i\}), \mathcal{M}'(a_{1,54})) \\
 & = & max(\mathcal{M}'(\{i\}), max(\mathcal{M}'(\neg\{j\}), \\
 & & \mathcal{M}'(\neg\phi[R\ :=\ (i,j)]))) \\
 & = & max(0, max(0, \mathcal{M}'(\phi[R := (i,j)]))) \\
 & = & \mathcal{M}'(\phi[R := (i,j)])
\end{array}
$$

55

$$
\begin{array}{rcc}
\mathcal{M}((<\ n\ R\ \phi)[R := (i,j)]) & = & \mathcal{M}((<\ n\ R\ \phi)) + 1 \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(\top) & = & 0
\end{array}
$$

56

$$
\begin{array}{rcc}
\mathcal{M}((\geq\ n\ R\ \phi)[R := (i,j)]) & = & \mathcal{M}((\geq\ n\ R\ \phi)) + 1 \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(RHS_{56}) & = & max(\mathcal{M}(\{i\}), \mathcal{M}(a_{1,56})) \\
 & = & max(0, max(\mathcal{M}(\{j\}), \mathcal{M}(\phi[R\ :=\ (i,j)]))) \\
 & = & max(0, max(0, \mathcal{M}(\phi) + 1)) \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}'((\geq\ n\ R\ \phi)[R := (i,j)]) & = & \mathcal{M}'((\geq\ n\ R\ \phi[R := (i,j)])) + 1 \\
 & = & \mathcal{M}'(\phi[R := (i,j)]) + 1 \\
\mathcal{M}'(RHS_{56}) & = & max(\mathcal{M}'(\{i\}), \mathcal{M}'(a_{1,56})) \\
 & = & max(0, max(\mathcal{M}'(\{j\}), \mathcal{M}'(\phi[R\ :=\ (i,j)]))) \\
 & = & max(0, max(0, \mathcal{M}'(\phi[R := (i,j)]))) \\
 & = & \mathcal{M}'(\phi[R := (i,j)])
\end{array}
$$

57

$$
\begin{array}{rcc}
\mathcal{M}((\geq\ n\ R\ \phi)[R := (i,j)]) & = & \mathcal{M}((\geq\ n\ R\ \phi)) + 1 \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(\bot) & = & 0
\end{array}
$$

58

$$
\begin{array}{rcc}
\mathcal{M}((<\ n\ R^-\ \phi)[R := (i,j)]) & = & \mathcal{M}((<\ n\ R^-\ \phi)) + 1 \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(RHS_{58}) & = & max(\mathcal{M}(\neg\{j\}), \mathcal{M}(a_{1,58})) \\
 & = & max(\mathcal{M}(\{j\}), max(\mathcal{M}(\neg\{i\}), \\
 & & \mathcal{M}(\neg\phi[R\ :=\ (i,j)]))) \\
 & = & max(0, max(0, \mathcal{M}(\phi) + 1)) \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}'((<\ n\ R^-\ \phi)[R := (i,j)]) & = & \mathcal{M}'((<\ n\ R^-\ \phi[R := (i,j)])) + 1 \\
 & = & \mathcal{M}'(\phi[R := (i,j)]) + 1 \\
\mathcal{M}'(RHS_{58}) & = & max(\mathcal{M}'(\neg\{j\}), \mathcal{M}'(a_{1,58})) \\
 & = & max(\mathcal{M}'(\{j\}), max(\mathcal{M}'(\neg\{i\}), \\
 & & \mathcal{M}'(\neg\phi[R\ :=\ (i,j)]))) \\
 & = & max(0, max(0, \mathcal{M}'(\phi[R := (i,j)]))) \\
 & = & \mathcal{M}'(\phi[R := (i,j)])
\end{array}
$$

59

$$
\begin{array}{rcc}
\mathcal{M}((<\ n\ R^-\ \phi)[R := (i,j)]) & = & \mathcal{M}((<\ n\ R^-\ \phi)) + 1 \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(\top) & = & 0
\end{array}
$$

60

$$
\begin{array}{rcc}
\mathcal{M}((\geq\ n\ R^-\ \phi)[R := (i,j)]) & = & \mathcal{M}((\geq\ n\ R^-\ \phi)) + 1 \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(RHS_{60}) & = & max(\mathcal{M}(\{j\}), \mathcal{M}(a_{1,60})) \\
 & = & max(0, max(\mathcal{M}(\{i\}), \mathcal{M}(\phi[R\ :=\ (i,j)]))) \\
 & = & max(0, max(0, \mathcal{M}(\phi) + 1)) \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}'((\geq\ n\ R^-\ \phi)[R := (i,j)]) & = & \mathcal{M}'((\geq\ n\ R^-\ \phi[R := (i,j)])) + 1 \\
 & = & \mathcal{M}'(\phi[R := (i,j)]) + 1 \\
\mathcal{M}'(RHS_{60}) & = & max(\mathcal{M}'(\{j\}), \mathcal{M}'(a_{1,60})) \\
 & = & max(0, max(\mathcal{M}'(\{i\}), \mathcal{M}'(\phi[R\ :=\ (i,j)]))) \\
 & = & max(0, max(0, \mathcal{M}'(\phi[R := (i,j)]))) \\
 & = & \mathcal{M}'(\phi[R := (i,j)])
\end{array}
$$

61

$$
\begin{array}{rcc}
\mathcal{M}((\geq\ n\ R^-\ \phi)[R := (i,j)]) & = & \mathcal{M}((\geq\ n\ R^-\ \phi)) + 1 \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(\bot) & = & 0
\end{array}
$$

62

$$
\begin{array}{rcl}
\mathcal{M}((\bowtie\ n\ R\ \phi)[i \gg^{in} j]) & = & \mathcal{M}((\bowtie\ n\ R\ \phi)) + 1 \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(RHS_{62}) & = & max(\mathcal{M}(a_{1,62}), (\bowtie\ n\ R\, \phi[i \gg^{in} j]), \mathcal{M}(a_{2,62}), \\
 & & \mathcal{M}(a_{3,62}), (\bowtie\ (n+1)\ R\, \phi[i \gg^{in} j]), \\
 & & \mathcal{M}(a_{4,62}), (\bowtie\ (n-1)\ R\, \phi[i \gg^{in} j]), \\
 & & \mathcal{M}(a_{5,62}), (\bowtie\ (n+1)\ R\, \phi[i \gg^{in} j]), \\
 & & \mathcal{M}(a_{6,62}), \mathcal{M}(a_{7,62}), \mathcal{M}(a_{8,62}), \\
 & & \mathcal{M}(a_{9,62}), (\bowtie\ n\ R\, \phi[i \gg^{in} j]) \\
 & = & max(0, max(0, \mathcal{M}(\phi) + 1)) \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}'((\geq\ n\ R\ \phi)[i \gg^{in} j]) & = & \mathcal{M}'(\phi[i \gg^{in} j]) + 1 \\
\mathcal{M}'(RHS_{62}) & = & max(\mathcal{M}'(a_{1,62}), (\bowtie\ n\ R\, \phi[i \gg^{in} j]), \mathcal{M}'(a_{2,62}), \\
 & & \mathcal{M}'(a_{3,62}), (\bowtie\ (n+1)\ R\, \phi[i \gg^{in} j]), \\
 & & \mathcal{M}'(a_{4,62}), (\bowtie\ (n-1)\ R\, \phi[i \gg^{in} j]), \\
 & & \mathcal{M}'(a_{5,62}), (\bowtie\ (n+1)\ R\, \phi[i \gg^{in} j]), \\
 & & \mathcal{M}'(a_{6,62}), \mathcal{M}'(a_{7,62}), \mathcal{M}'(a_{8,62}), \\
 & & \mathcal{M}'(a_{9,62}), (\bowtie\ n\ R\, \phi[i \gg^{in} j]) \\
 & = & max(0, max(0, \mathcal{M}'(\phi[R := (i,j)]))) \\
 & = & \mathcal{M}'(\phi[R := (i,j)])
\end{array}
$$

63

$$
\begin{array}{rcl}
\mathcal{M}((<\ n\ R^{-}\ \phi)[i \gg^{in} j]) & = & \mathcal{M}((<\ n\ R^{-}\ \phi)) + 1 \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(RHS_{63}) & = & max(\mathcal{M}(a_{1,63}), \mathcal{M}(a_{2,63}), \mathcal{M}(a_{3,k,63}), \mathcal{M}(a_{4,k,63}), \\
 & & \mathcal{M}(a_{5,63}), (<\ n\ R^{-}\, \phi[i \gg^{in} j])) \\
 & = & max(0, max(0, \mathcal{M}(\phi) + 1)) \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}'((<\ n\ R^{-}\ \phi)[i \gg^{in} j]) & = & \mathcal{M}'(\phi[i \gg^{in} j]) + 1 \\
\mathcal{M}'(RHS_{63}) & = & max(\mathcal{M}'(a_{1,63}), \mathcal{M}'(a_{2,63}), \mathcal{M}'(a_{3,k,63}), \mathcal{M}'(a_{4,k,63}), \\
 & & \mathcal{M}'(a_{5,63}), (<\ n\ R^{-}\, \phi[i \gg^{in} j])) \\
 & = & max(0, max(0, \mathcal{M}'(\phi[i \gg^{in} j]))) \\
 & = & \mathcal{M}'(\phi[i \gg^{in} j])
\end{array}
$$

64

$$
\begin{array}{rcl}
\mathcal{M}((\geq\ n\ R^{-}\ \phi)[i \gg^{in} j]) & = & \mathcal{M}((\geq\ n\ R^{-}\ \phi)) + 1 \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(RHS_{64}) & = & max(\mathcal{M}(a_{1,64}), \mathcal{M}(a_{2,k,64}), \mathcal{M}(a_{3,k,64}), \\
 & & \mathcal{M}(a_{4,64}), (\geq\ n\ R\, \phi[i \gg^{in} j])) \\
 & = & max(0, max(0, \mathcal{M}(\phi) + 1)) \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}'((\geq\ n\ R^{-}\ \phi)[i \gg^{in} j]) & = & \mathcal{M}'(\phi[i \gg^{in} j]) + 1 \\
\mathcal{M}'(RHS_{64}) & = & max(\mathcal{M}'(a_{1,64}), \mathcal{M}'(a_{2,k,64}), \mathcal{M}'(a_{3,k,64}), \\
 & & \mathcal{M}'(a_{4,64}), (\geq\ n\ R\, \phi[i \gg^{in} j])) \\
 & = & max(0, max(0, \mathcal{M}'(\phi[i \gg^{in} j]))) \\
 & = & \mathcal{M}'(\phi[i \gg^{in} j])
\end{array}
$$

65 – 67 As $i \gg^{out} j$ can be seen as $i \gg^{in} j$ applied to the inverse of the edges, rules 65 to 67 can be obtained from rules 62 to 64.

68

$$\begin{array}{lcl}
\mathcal{M}((\exists R.\phi)[C := C \pm i]) & = & \mathcal{M}(\exists R.\phi) + 1 \\
 & = & \mathcal{M}(\phi) + 2 \\
\mathcal{M}(\exists R.(\phi[C := C \pm i])) & = & \mathcal{M}(\phi[C := C \pm i]) + 1 \\
 & = & \mathcal{M}(\phi) + 2 \\
\mathcal{M}'((\exists R.\phi)[C := C \pm i]) & = & \mathcal{M}'(\phi[C := C \pm i]) + 1 \\
\mathcal{M}'(\exists R.(\phi[C := C \pm i])) & = & \mathcal{M}(\phi[C := C \pm i])
\end{array}$$

69 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

70

$$\begin{array}{lcl}
\mathcal{M}((\exists R.\phi)[C := \psi]) & = & \mathcal{M}(\exists R.\phi) + 1 \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(\exists R.(\phi[C := \psi])) & = & \mathcal{M}(\phi[C := \psi]) \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}'((\exists R.\phi)[C := \psi]) & = & \mathcal{M}'(\phi[C := \psi]) + 1 \\
\mathcal{M}'(\exists R.(\phi[C := \psi])) & = & \mathcal{M}(\phi[C := \psi])
\end{array}$$

71 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

72

$$\begin{array}{lcl}
\mathcal{M}((\exists R.\phi)[R' := R' \pm (i,j)]) & = & \mathcal{M}(\exists R.\phi) + 1 \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(\exists R.(\phi[R' := R' \pm (i,j)])) & = & \mathcal{M}(\phi[R' := R' \pm (i,j)]) \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}'((\exists R.\phi)[R' := R' \pm (i,j)]) & = & \mathcal{M}'(\phi[R' := R' \pm (i,j)]) + 1 \\
\mathcal{M}'(\exists R.(\phi[R' := R' \pm (i,j)])) & = & \mathcal{M}(\phi[R' := R' \pm (i,j)])
\end{array}$$

73 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

74

$$\begin{array}{lcl}
\mathcal{M}((\exists R.\phi)[R := R + (i,j)]) & = & \mathcal{M}(\exists R.\phi) + 1 \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(d_1) & = & max(\mathcal{M}(\neg\{i\}), \mathcal{M}(d_{1,1}), \mathcal{M}(d_{1,2}), \mathcal{M}(\{i\}), \mathcal{M}(d_{1,3})) \\
 & = & max(\mathcal{M}(\{i\}), \mathcal{M}(d_{1,1,1}) + 1, max(\mathcal{M}(\{j\}), \mathcal{M}(d_{1,2,1})) + 1, \\
 & & 0, \mathcal{M}(d_{1,3,1}) + 1) \\
 & = & max(0, \mathcal{M}(\phi) + 1, max(0, \mathcal{M}(\phi)) + 1, \mathcal{M}(\phi) + 1) \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}'((\exists R.\phi)[R := R + (i,j)]) & = & \mathcal{M}'(\phi[R := R + (i,j)]) + 1 \\
\mathcal{M}'(d_1) & = & max(\mathcal{M}'(\neg\{i\}), \mathcal{M}'(d_{1,1}), \mathcal{M}'(d_{1,2}), \mathcal{M}'(\{i\}), \mathcal{M}'(d_{1,3})) \\
 & = & max(\mathcal{M}'(\{i\}), \mathcal{M}'(\phi[R := R + (i,j)]), max(\mathcal{M}'(\{j\}), \\
 & & \mathcal{M}'(\phi[R := R + (i,j)])), 0, \mathcal{M}'(\phi[R := R + (i,j)])) \\
 & = & \mathcal{M}'(\phi[R := R + (i,j)])
\end{array}$$

75 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

76

$$\begin{array}{rcl}
\mathcal{M}((\exists R.\phi)[R := R - (i,j)]) & = & \mathcal{M}(\exists R.\phi) + 1 \\
& = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(d_2) & = & max(\mathcal{M}(\neg\{i\}), \mathcal{M}(d_{2,1}), \mathcal{M}(\{i\}), \mathcal{M}(d_{2,2})) \\
& = & max(\mathcal{M}(\{i\}), max(\mathcal{M}(d_{2,1,1}), \mathcal{M}(\neg\{j\})) + 1, 0, \mathcal{M}(d_{2,2,1}) + 1) \\
& = & max(0, max(\mathcal{M}(\phi) + 1, \mathcal{M}(\{i\})) + 1, \mathcal{M}(\phi) + 1) \\
& = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}'((\exists R.\phi)[R := R - (i,j)]) & = & \mathcal{M}'(\phi[R := R - (i,j)]) + 1 \\
\mathcal{M}'(d_2) & = & max(\mathcal{M}'(\neg\{i\}), \mathcal{M}'(d_{2,1}), \mathcal{M}'(\{i\}), \mathcal{M}'(d_{2,2})) \\
& = & max(\mathcal{M}'(\{i\}), max(\mathcal{M}'(\phi[R := R - (i,j)]), \mathcal{M}'(\neg\{j\})), \\
& & 0, \mathcal{M}'(\phi[R := R - (i,j)])) \\
& = & max(0, max(\mathcal{M}'(\phi[R := R - (i,j)]), \mathcal{M}'(\{j\})), \\
& & \mathcal{M}'(\phi[R := R - (i,j)])) \\
& = & \mathcal{M}'(\phi[R := R - (i,j)])
\end{array}$$

77 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

78

$$\begin{array}{rcl}
\mathcal{M}((\exists R.\phi)[R' := Q]) & = & \mathcal{M}(\exists R.\phi) + 1 \\
& = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(\exists R.(\phi[R' := Q])) & = & \mathcal{M}(\phi[R' := Q]) \\
& = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}'((\exists R.\phi)[R' := Q]) & = & \mathcal{M}'(\phi[R' := Q]) + 1 \\
\mathcal{M}'(\exists R.(\phi[R' := Q])) & = & \mathcal{M}(\phi[R' := Q])
\end{array}$$

79 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

80

$$\begin{array}{rcl}
\mathcal{M}((\exists R.\phi)[R := Q]) & = & \mathcal{M}(\exists R.\phi) + 1 \\
& = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(\exists Q.(\phi[R := Q])) & = & \mathcal{M}(\phi[R := Q]) \\
& = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}'((\exists R.\phi)[R := Q]) & = & \mathcal{M}'(\phi[R := Q]) + 1 \\
\mathcal{M}'(\exists Q.(\phi[R := Q])) & = & \mathcal{M}(\phi[R := Q])
\end{array}$$

81 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

82

$$\begin{array}{rcl}
\mathcal{M}((\exists R.\phi)[R' := (i,j)]) & = & \mathcal{M}(\exists R.\phi) + 1 \\
& = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(\exists R.(\phi[R' := (i,j)])) & = & \mathcal{M}(\phi[R' := (i,j)]) \\
& = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}'((\exists R.\phi)[R' := (i,j)]) & = & \mathcal{M}'(\phi[R' := (i,j)]) + 1 \\
\mathcal{M}'(\exists R.(\phi[R' := (i,j)])) & = & \mathcal{M}(\phi[R' := (i,j)])
\end{array}$$

83 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

84

$$\begin{array}{rcc}
\mathcal{M}((\exists R.\phi)[R := (i,j)]) & = & \mathcal{M}(\exists R.\phi) + 1 \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(RHS_{84}) & = & max(\mathcal{M}(\{i\}, max(\mathcal{M}(\{j\}), \mathcal{M}(\phi[R := (i,j)])))) \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}'((\exists R.\phi)[R := (i,j)]) & = & \mathcal{M}'(\phi[R := (i,j)]) + 1 \\
\mathcal{M}'(RHS_{84}) & = & max(\mathcal{M}(\{i\}, max(\mathcal{M}(\{j\}), \mathcal{M}(\phi[R := (i,j)])))) \\
 & = & \mathcal{M}(\phi[R := (i,j)])
\end{array}$$

85 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

86 – 89 Rules 86 to 89 are translations of rules 62 to 67 with $\bowtie$ replaced by $\geq$ and $n = 1$

90

$$\begin{array}{rcc}
\mathcal{M}((\forall R.\phi)[C := C \pm i]) & = & \mathcal{M}(\forall R.\phi) + 1 \\
 & = & \mathcal{M}(\phi) + 2 \\
\mathcal{M}(\forall R.(\phi[C := C \pm i])) & = & \mathcal{M}(\phi[C := C \pm i]) + 1 \\
 & = & \mathcal{M}(\phi) + 2 \\
\mathcal{M}'((\forall R.\phi)[C := C \pm i]) & = & \mathcal{M}'(\phi[C := C \pm i]) + 1 \\
\mathcal{M}'(\forall R.(\phi[C := C \pm i])) & = & \mathcal{M}'(\phi[C := C \pm i])
\end{array}$$

91 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

92

$$\begin{array}{rcc}
\mathcal{M}((\forall R.\phi)[R' := R' \pm (i,j)]]) & = & \mathcal{M}(\forall R.\phi) + 1 \\
 & = & \mathcal{M}(\phi) + 2 \\
\mathcal{M}(\forall R.(\phi[R' := R' \pm (i,j)])) & = & \mathcal{M}(\phi[R' := R' \pm (i,j)]) + 1 \\
 & = & \mathcal{M}(\phi) + 2 \\
\mathcal{M}'((\forall R.\phi)[R' := R' \pm (i,j)]) & = & \mathcal{M}'(\phi[R' := R' \pm (i,j)]) + 1 \\
\mathcal{M}'(\forall R.(\phi[R' := R' \pm (i,j)])) & = & \mathcal{M}'(\phi[R' := R' \pm (i,j)])
\end{array}$$

93 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

94

$$\begin{array}{rcc}
\mathcal{M}((\forall R.\phi)[C := \psi]) & = & \mathcal{M}(\forall R.\phi) + 1 \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(\forall R.(\phi[C := \psi])) & = & \mathcal{M}(\phi[C := \psi]) \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}'((\forall R.\phi)[C := \psi]) & = & \mathcal{M}'(\phi[C := \psi]) + 1 \\
\mathcal{M}'(\forall R.(\phi[C := \psi])) & = & \mathcal{M}(\phi[C := \psi])
\end{array}$$

95 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

96

$$\begin{array}{rcl}
\mathcal{M}((\forall R.\phi)[R := R + (i,j)]) & = & \mathcal{M}(\forall R.\phi) + 1 \\
 & = & \mathcal{M}(\phi) + 2 \\
\mathcal{M}(d_3) & = & max(\mathcal{M}(\neg\{i\}), \mathcal{M}(d_{3,1}), \mathcal{M}(d_{3,2}), \mathcal{M}(\{i\}), \mathcal{M}(d_{3,3})) \\
 & = & max(\mathcal{M}(\{i\}), \mathcal{M}(d_{3,1,1}), max(\mathcal{M}(\{j\}), \mathcal{M}(d_{3,2,1})) + 1, \\
 & & 0, \mathcal{M}(d_{3,3,1}) + 1) \\
 & = & max(0, \mathcal{M}(\phi) + 2, \mathcal{M}(\phi) + 2) \\
 & = & \mathcal{M}(\phi) + 2 \\
\mathcal{M}'((\forall R.\phi)[R := R + (i,j)]) & = & \mathcal{M}(\forall R.\phi[R := R + (i,j)]) + 1 \\
\mathcal{M}'(d_3) & = & max(\mathcal{M}'(\neg\{i\}), \mathcal{M}'(d_{3,1}), \mathcal{M}'(d_{3,2}), \mathcal{M}'(\{i\}), \mathcal{M}'(d_{3,3})) \\
 & = & max(\mathcal{M}'(\{i\}), \mathcal{M}'(\phi[R := R + (i,j)]), max(\mathcal{M}'(\{j\}), \\
 & & \mathcal{M}(\phi[R := R + (i,j)])), 0, \mathcal{M}(\phi[R := R + (i,j)])) \\
 & = & max(0, \mathcal{M}(\phi[R := R + (i,j)]), \mathcal{M}(\phi[R := R + (i,j)])) \\
 & = & \mathcal{M}(\phi)
\end{array}$$

97 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

98

$$\begin{array}{rcl}
\mathcal{M}((\forall R.\phi)[R := R - (i,j)]) & = & \mathcal{M}(\forall R.\phi) + 1 \\
 & = & \mathcal{M}(\phi) + 2 \\
\mathcal{M}(d_4) & = & max(\mathcal{M}(\neg\{i\}), \mathcal{M}(d_{4,1}), \mathcal{M}(\{i\}), \mathcal{M}(d_{4,2})) \\
 & = & max(\mathcal{M}(\{i\}), max(\mathcal{M}(d_{4,1,1}), \mathcal{M}(\{j\})) + 1, 0, \\
 & & \mathcal{M}(d_{4,2,1}) + 1) \\
 & = & max(0, max(\mathcal{M}(\phi) + 1, 0) + 1, \mathcal{M}(\phi) + 2) \\
 & = & \mathcal{M}(\phi) + 2 \\
\mathcal{M}'((\forall R.\phi)[R := R - (i,j)]) & = & \mathcal{M}(\phi[R := R - (i,j)]) + 1 \\
\mathcal{M}'(d_4) & = & max(\mathcal{M}'(\neg\{i\}), \mathcal{M}'(d_{4,1}), \mathcal{M}'(\{i\}), \mathcal{M}'(d_{4,2})) \\
 & = & max(\mathcal{M}'(\{i\}), max(\mathcal{M}'(\phi[R := R - (i,j)]), \mathcal{M}'(\{j\})), \\
 & & 0, \mathcal{M}'(\phi[R := R - (i,j)])) \\
 & = & max(0, max(\mathcal{M}'(\phi[R := R - (i,j)]), 0), \\
 & & \mathcal{M}'(\phi[R := R - (i,j)])) \\
 & = & \mathcal{M}'(\phi[R := R - (i,j)])
\end{array}$$

99 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

100

$$\begin{array}{rcl}
\mathcal{M}((\forall R.\phi)[R' := Q]) & = & \mathcal{M}(\forall R.\phi) + 1 \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(\forall R.(\phi[R' := Q])) & = & \mathcal{M}(\phi[R' := Q]) \\
 & = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}'((\forall R.\phi)[R' := Q]) & = & \mathcal{M}'(\phi[R' := Q]) + 1 \\
\mathcal{M}'(\forall R.(\phi[R' := Q])) & = & \mathcal{M}(\phi[R' := Q])
\end{array}$$

101 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

102

$$\begin{array}{rcc}
\mathcal{M}((\forall R.\phi)[R := Q]) & = & \mathcal{M}(\forall R.\phi) + 1 \\
& = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(\forall Q.(\phi[R := Q])) & = & \mathcal{M}(\phi[R := Q]) \\
& = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}'((\forall R.\phi)[R := Q]) & = & \mathcal{M}'(\phi[R := Q]) + 1 \\
\mathcal{M}'(\forall Q.(\phi[R := Q])) & = & \mathcal{M}(\phi[R := Q])
\end{array}$$

103 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

104

$$\begin{array}{rcc}
\mathcal{M}((\forall R.\phi)[R' := (i,j)]) & = & \mathcal{M}(\forall R.\phi) + 1 \\
& = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(\forall R.(\phi[R' := (i,j)])) & = & \mathcal{M}(\phi[R' := (i,j)]) \\
& = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}'((\forall R.\phi)[R' := (i,j)]) & = & \mathcal{M}'(\phi[R' := (i,j)]) + 1 \\
\mathcal{M}'(\forall R.(\phi[R' := (i,j)])) & = & \mathcal{M}(\phi[R' := (i,j)])
\end{array}$$

105 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

106

$$\begin{array}{rcc}
\mathcal{M}((\forall R.\phi)[R := (i,j)]) & = & \mathcal{M}(\forall R.\phi) + 1 \\
& = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}(RHS_{106}) & = & max(\mathcal{M}(\{i\}, max(\mathcal{M}(\{j\}), \mathcal{M}(\phi[R := (i,j)])))) \\
& = & \mathcal{M}(\phi) + 1 \\
\mathcal{M}'((\forall R.\phi)[R := (i,j)]) & = & \mathcal{M}'(\phi[R := (i,j)]) + 1 \\
\mathcal{M}'(RHS_{106}) & = & max(\mathcal{M}(\{i\}, max(\mathcal{M}(\{j\}), \mathcal{M}(\phi[R := (i,j)])))) \\
& = & \mathcal{M}(\phi[R := (i,j)])
\end{array}$$

107 As the definitions of $\mathcal{M}$ and $\mathcal{M}'$ do not discriminate R and R^-, this rule is identic to the previous one.

108 − 111 Rules 108 to 111 are translations of rules 62 to 67 with $\bowtie = <$ and $n = 1$

where :

$$\begin{array}{rcll}
RHS_{35} & = & (\{i\} \Leftrightarrow \{j\}) & \Rightarrow \ \exists R.Self) \\
& \sqcup & (\neg\{i\} \sqcap \{j\} & \Rightarrow \ \exists R.Self \sqcup \exists R.\{i\}) \\
RHS_{37} & = & (\{i\} \Leftrightarrow \{j\}) & \Rightarrow \ \exists R.Self) \\
& \sqcup & (\neg\{i\} \sqcap \{j\} & \Rightarrow \ \exists R.Self \sqcup \exists R^-.\{i\}) \\
RHS_{45} & = & (\{i\} \sqcap \exists U.(\{j\} \ \sqcap \phi[R := R + (i,j)]) \sqcap \forall R.\neg\{j\}) & \\
& \Rightarrow & (\bowtie \ (n-1) \ R \ \phi[R := R + (i,j)])) & \\
& \sqcap & ((\neg\{i\} \sqcup \forall U.(\neg\{j\} \sqcup \neg\phi[R := R + (i,j)]) \sqcup \exists R.\{j\}) & \\
& \Rightarrow & (\bowtie \ n \ R \ \phi[R := R + (i,j)])) & \\
RHS_{47} & = & (\{i\} \sqcap \exists U.(\{j\} \ \sqcap \phi[R := R - (i,j)]) \sqcap \exists R.\{j\}) & \\
& \Rightarrow & (\bowtie \ (n+1) \ R \ \phi[R := R - (i,j)])) & \\
& \sqcap & ((\neg\{i\} \sqcup \forall U.(\neg\{j\} \sqcup \neg\phi[R := R - (i,j)]) \sqcup \forall R.\neg\{j\}) & \\
& \Rightarrow & (\bowtie \ n \ R \ \phi[R := R - (i,j)])) &
\end{array}$$

$$
\begin{array}{lll}
RHS_{54} & = & \neg\{i\} \sqcup \forall U.(\neg\{j\} \sqcup \neg\phi[R := (i,j)]) \\
RHS_{56} & = & \{i\} \sqcap \exists U.(\{j\} \sqcap \phi[R := (i,j)]) \\
RHS_{58} & = & \neg\{j\} \sqcup \forall U.(\neg\{i\} \sqcup \neg\phi[R := (i,j)]) \\
RHS_{60} & = & \{j\} \sqcap \exists U.(\{i\} \sqcap \phi[R := (i,j)]) \\
RHS_{62} & = & (\exists U.(\{i\} \sqcap \{j\}) \\
 & \Rightarrow & (\bowtie\ n\ R\ \phi[i \gg^{in} j])) \\
 & \sqcap & (\exists U.(\{i\} \sqcap \neg\{j\}) \Rightarrow \\
 & & (\exists R.(\{i\} \sqcap \phi[i \gg^{in} j]) \sqcap \forall R.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \neg\phi[i \gg^{in} j]) \\
 & \Rightarrow & (\bowtie\ (n+1)\ R\ \phi[i \gg^{in} j])) \\
 & \sqcap & (\exists R.(\{i\} \sqcap \neg\phi[i \gg^{in} j]) \sqcap \forall R.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \phi[i \gg^{in} j]) \\
 & \Rightarrow & (\bowtie\ (n-1)\ R\ \phi[i \gg^{in} j])) \\
 & \sqcap & (\exists R.(\{i\} \sqcap \phi[i \gg^{in} j]) \sqcap \exists R.\{j\} \\
 & \Rightarrow & (\bowtie\ (n+1)\ R\ \phi[i \gg^{in} j])) \\
 & \sqcap & ((\forall R.\neg\{i\}) \\
 & \sqcup & (\exists R.(\{i\} \sqcap \neg\phi[i \gg^{in} j]) \sqcap \exists R.\{j\}) \\
 & \sqcup & (\exists R.(\{i\} \sqcap \phi[i \gg^{in} j]) \sqcap \forall R.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \phi[i \gg^{in} j])) \\
 & \sqcup & (\exists R.(\{i\} \sqcap \neg\phi[i \gg^{in} j]) \sqcap \forall R.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \neg\phi[i \gg^{in} j])) \\
 & \Rightarrow & (\bowtie\ n\ R\ \phi[i \gg^{in} j]))) \\
RHS_{63} & = & (\{i\} \sqcap \neg\{j\}) \\
 & \sqcup & (\neg\{i\} \sqcap \{j\} \\
 & \Rightarrow & \bigsqcup_{k\in[0,n]}(<\ k\ R^{-}\ \phi[i \gg^{in} j]) \\
 & \sqcap & \exists U.(\{i\} \sqcap (<\ (n-k)\ R^{-}\ (\phi[i \gg^{in} j] \sqcap \neg\exists R^{-}.\{j\})))) \\
 & \sqcup & ((\{i\} \Leftrightarrow \{j\}) \\
 & \Rightarrow & (<\ n\ R^{-}\ \phi[i \gg^{in} j])) \\
RHS_{64} & = & (\neg\{i\} \sqcap \{j\} \\
 & \Rightarrow & \bigsqcup_{k\in[0,n]}(\geq\ k\ R^{-}\ \phi[i \gg^{in} j]) \\
 & \sqcap & \exists U.(\{i\} \sqcap (\geq\ (n-k)\ R^{-}\ (\phi[i \gg^{in} j] \sqcap \exists R^{-}.\{j\})))) \\
 & \sqcap & ((\{i\} \Leftrightarrow \{j\}) \\
 & \Rightarrow & (\geq\ n\ R^{-}\ \phi[i \gg^{in} j])) \\
RHS_{84} & = & \{i\} \sqcap \exists U.(\{j\} \sqcap \phi[R := (i,j)]) \\
RHS_{106} & = & (\{i\} \sqcap \exists U.(\{j\} \sqcap \phi[R := (i,j)])) \sqcup \neg\{i\} \\
a_{1,45} & = & (\{i\} \sqcap \exists U.(\{j\}\ \sqcap \phi[R := R + (i,j)]) \sqcap \forall R.\neg\{j\}) \\
 & \Rightarrow & (\bowtie\ (n-1)\ R\ \phi[R := R + (i,j)]) \\
a_{2,45} & = & (\neg\{i\} \sqcup \forall U.(\neg\{j\} \sqcup \neg\phi[R := R + (i,j)]) \sqcup \exists R.\{j\}) \\
 & \Rightarrow & (\bowtie\ n\ R\ \phi[R := R + (i,j)]) \\
a_{1,47} & = & (\{i\} \sqcap \exists U.(\{j\}\ \sqcap \phi[R := R - (i,j)]) \sqcap \exists R.\{j\}) \\
 & \Rightarrow & (\bowtie\ (n+1)\ R\ \phi[R := R - (i,j)]) \\
a_{2,47} & = & (\neg\{i\} \sqcup \forall U.(\neg\{j\} \sqcup \neg\phi[R := R - (i,j)]) \sqcup \forall R.\neg\{j\}) \\
 & \Rightarrow & (\bowtie\ n\ R\ \phi[R := R - (i,j)]) \\
a_{1,54} & = & \forall U.(\neg\{j\} \sqcup \neg\phi[R := (i,j)]) \\
a_{1,56} & = & \exists U.(\{j\} \sqcap \phi[R := (i,j)]) \\
a_{1,58} & = & \forall U.(\neg\{i\} \sqcup \neg\phi[R := (i,j)]) \\
a_{1,60} & = & \exists U.(\{i\} \sqcap \phi[R := (i,j)]) \\
a_{1,62} & = & (\exists U.(\{i\} \sqcap \{j\}) \\
a_{2,62} & = & (\exists U.(\{i\} \sqcap \neg\{j\}) \\
a_{3,62} & = & (\exists R.(\{i\} \sqcap \phi[i \gg^{in} j]) \sqcap \forall R.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \neg\phi[i \gg^{in} j])
\end{array}
$$

$$
\begin{array}{lcl}
a_{4,62} & = & (\exists R.(\{i\} \sqcap \neg\phi[i \gg^{in} j]) \sqcap \forall R.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \phi[i \gg^{in} j]) \\
a_{5,62} & = & (\exists R.(\{i\} \sqcap \phi[i \gg^{in} j]) \sqcap \exists R.\{j\} \\
a_{6,62} & = & ((\forall R.\neg\{i\}) \\
a_{7,62} & = & (\exists R.(\{i\} \sqcap \neg\phi[i \gg^{in} j]) \sqcap \exists R.\{j\}) \\
a_{8,62} & = & (\exists R.(\{i\} \sqcap \phi[i \gg^{in} j]) \sqcap \forall R.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \phi[i \gg^{in} j])) \\
a_{9,62} & = & (\exists R.(\{i\} \sqcap \neg\phi[i \gg^{in} j]) \sqcap \forall R.\neg\{j\} \sqcap \exists U.(\{j\} \sqcap \neg\phi[i \gg^{in} j])) \\
 & \Rightarrow & (\bowtie\ n\ R\ \phi[i \gg^{in} j])))) \\
a_{1,63} & = & (\{i\} \sqcap \neg\{j\}) \\
a_{2,63} & = & (\neg\{i\} \sqcap \{j\} \\
a_{3,k,63} & = & (<\ k\ R^{-}\ \phi[i \gg^{in} j]) \\
a_{4,k,63} & = & \exists U.(\{i\} \sqcap (<\ (n-k)\ R^{-}\ (\phi[i \gg^{in} j] \sqcap \neg\exists R^{-}.\{j\})))) \\
a_{5,63} & = & ((\{i\} \Leftrightarrow \{j\}) \\
a_{1,64} & = & \neg\{i\} \sqcap \{j\} \\
a_{2,k,64} & = & (\geq\ k\ R^{-}\ \phi[i \gg^{in} j]) \\
a_{3,k,64} & = & \exists U.(\{i\} \sqcap (\geq\ (n-k)\ R^{-}\ (\phi[i \gg^{in} j] \sqcap \exists R^{-}.\{j\})))) \\
a_{4,64} & = & \{i\} \Leftrightarrow \{j\} \\
b_{1,45} & = & \{i\} \\
b_{2,45} & = & \exists U.(\{j\}\ \sqcap \phi[R := R + (i,j)]) \\
b_{3,45} & = & \forall R.\neg\{j\} \\
b_{4,45} & = & (\bowtie\ (n-1)\ R\ \phi[R := R + (i,j)])) \\
b_{11,45} & = & \neg\{i\} \\
b_{12,45} & = & \forall U.(\neg\{j\}\ \sqcup \neg(\phi[R := R + (i,j)])) \\
b_{13,45} & = & \exists R.\{j\} \\
b_{14,45} & = & (\bowtie\ n\ R\ \phi[R := R + (i,j)])) \\
b_{1,47} & = & \{i\} \\
1b_{2,47} & = & \exists U.(\{j\}\ \sqcap \phi[R := R - (i,j)]) \\
b_{3,47} & = & \exists R.\{j\} \\
b_{4,47} & = & (\bowtie\ (n+1)\ R\ \phi[R := R - (i,j)])) \\
b_{11,47} & = & \neg\{i\} \\
b_{12,47} & = & \forall U.(\neg\{j\}\ \sqcup \neg(\phi[R := R - (i,j)])) \\
b_{13,47} & = & \forall R.\neg\{j\} \\
b_{14,47} & = & (\bowtie\ n\ R\ \phi[R := R - (i,j)])) \\
c_{1,45} & = & (\{j\}\ \sqcap \phi[R := R + (i,j)]) \\
c_{2,45} & = & (\neg\{j\}\ \sqcup \neg(\phi[R := R + (i,j)])) \\
c_{1,47} & = & (\{j\}\ \sqcap \phi[R := R - (i,j)]) \\
c_{2,47} & = & (\neg\{j\}\ \sqcup \neg(\phi[R := R - (i,j)])) \\
d_1 & = & (\{i\} \Rightarrow \exists R.(\phi[R := R + (i,j)]) \sqcup \exists U.(\{j\} \sqcap \phi[R := R + (i,j)])) \\
 & \sqcap & (\neg\{i\} \Rightarrow \exists R.(\phi[R := R + (i,j)])) \\
 & = & (\neg\{i\} \sqcup \exists R.(\phi[R := R + (i,j)]) \sqcup \exists U.(\{j\} \sqcap \phi[R := R + (i,j)])) \\
 & \sqcap & (\{i\} \sqcup \exists R.(\phi[R := R + (i,j)])) \\
d_{1,1} & = & \exists R.(\phi[R := R + (i,j)]) \\
d_{1,2} & = & \exists U.(\{j\} \sqcap \phi[R := R + (i,j)]) \\
d_{1,3} & = & \exists R.(\phi[R := R + (i,j)]) \\
d_{1,2,1} & = & \{j\} \\
d_{1,2,2} & = & \phi[R := R + (i,j)] \\
d_{1,3,1} & = & \phi[R := R + (i,j)]
\end{array}
$$

$$
\begin{array}{lcl}
d_2 & = & (\{i\} \Rightarrow (\exists R.(\phi[R := R - (i,j)] \sqcap \neg\{j\}))) \\
& \sqcap & (\neg\{i\} \Rightarrow \exists R.(\phi[R := R - (i,j)])) \\
& = & (\neg\{i\} \sqcup (\exists R.(\phi[R := R - (i,j)] \sqcap \neg\{j\}))) \\
& \sqcap & (\{i\} \sqcup \exists R.(\phi[R := R - (i,j)])) \\
d_{2,1} & = & \exists R.(\phi[R := R - (i,j)] \sqcap \neg\{j\}) \\
d_{2,2} & = & \exists R.(\phi[R := R - (i,j)]) \\
d_{2,1,1} & = & \phi[R := R - (i,j)] \\
d_{2,1,2} & = & \neg\{j\} \\
d_{2,2,1} & = & \phi[R := R - (i,j)] \\
d_3 & = & (\{i\} \Rightarrow (\forall R.(\phi[R := R + (i,j)]) \sqcap \exists U.(\{j\} \sqcap \phi[R := R + (i,j)]))) \\
& \sqcap & (\neg\{i\} \Rightarrow \forall R.(\phi[R := R + (i,j)])) \\
& = & (\neg\{i\} \sqcup (\forall R.(\phi[R := R + (i,j)]) \sqcap \exists U.(\{j\} \sqcap \phi[R := R + (i,j)]))) \\
& \sqcap & (\{i\} \sqcup \forall R.(\phi[R := R + (i,j)])) \\
d_{3,1} & = & \forall R.(\phi[R := R + (i,j)]) \\
d_{3,2} & = & \exists U.(\{j\} \sqcap \phi[R := R + (i,j)]) \\
d_{3,3} & = & \forall R.(\phi[R := R + (i,j)]) \\
d_{3,2,1} & = & \phi[R := R + (i,j)] \\
d_{3,3,1} & = & \phi[R := R + (i,j)] \\
d_4 & = & (\{i\} \Rightarrow \forall R.(\phi[R := R - (i,j)] \sqcup \{j\})) \\
& \sqcap & (\neg\{i\} \Rightarrow \forall R.(\phi[R := R - (i,j)])) \\
& = & (\neg\{i\} \sqcup \forall R.(\phi[R := R - (i,j)] \sqcup \{j\})) \\
& \sqcap & (\{i\} \sqcup \forall R.(\phi[R := R - (i,j)])) \\
d_{4,1} & = & \forall R.(\phi[R := R - (i,j)] \sqcup \{j\}) \\
d_{4,2} & = & \forall R.(\phi[R := R - (i,j)]) \\
d_{4,1,1} & = & \phi[R := R - (i,j)] \\
d_{4,2,1} & = & \phi[R := R - (i,j)]
\end{array}
$$

□

Now we show that, given an $\mathcal{ALCQUIOS}elf\sigma$ concept, applying the translation rules yields an $\mathcal{ALCQUIOS}elf$ concept.

Lemma 3. *Let Σ be a signature, ϕ an $\mathcal{ALCQUIOS}elf\sigma$ concept, ψ a normal form obtained from ϕ by using the system $\mathcal{T}$, then ψ is an $\mathcal{ALCQUIOS}elf$ concept.*

The proof of this lemma is done by contradiction. If we assume that ψ contains a substitution, then ψ can be rewritten by means of a rule in $\mathcal{T}$. This contradicts the fact that ψ is in normal form. Therefore ψ is substitution-free. ψ is a $\mathcal{ALCQUIOS}elf$ concept since concepts constructors used in $\mathcal{T}$ are $\mathcal{ALCQUIOS}elf\sigma$ concepts.

From the above three lemmas 1, 2 and 3, the proof of Theorem 6.2.1 is straightforward.

The same proof may be used to show that other DLs are closed under substitutions. This is summarized in the following corollary.

Corollary 6.2.1. *The logics $\mathcal{ALCUO}$, $\mathcal{ALCUOS}elf$, $\mathcal{ALCUIO}$, $\mathcal{ALCQUO}$, $\mathcal{ALCUIOS}elf$, $\mathcal{ALCQUOS}elf$, $\mathcal{ALCQUIO}$, $\mathcal{ALCUO}@$, $\mathcal{ALCUO}@\mathcal{S}elf$,*

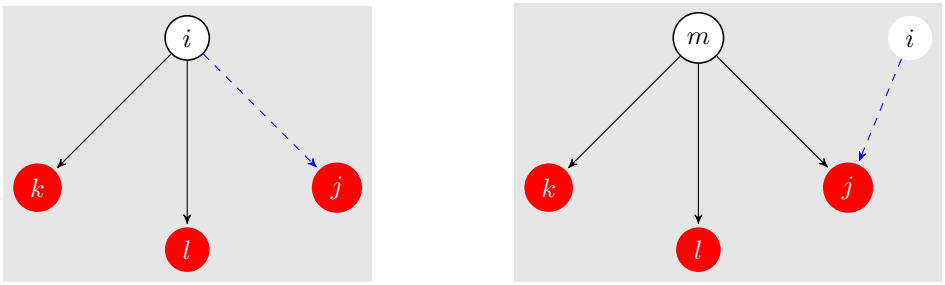


Figure 6.4: Example illustrating rule 38. The nodes satisfying $\phi[R := R + (i, j)]$ are drawn in red, the current node is circled in black. The leftmost graph shows the case in which $\{i\}$ is true, and then $(\forall R.\phi)[R := R + (i, j)]$ will be true if j satisfies $\phi[R := R + (i, j)]$. The rightmost graph shows that if $\{i\}$ is false, then the modification does not affect the property.

$\mathcal{ALCUIO}@$, $\mathcal{ALCQUO}@$, $\mathcal{ALCUIO}@\mathcal{Self}$, $\mathcal{ALCQUO}@\mathcal{Self}$, $\mathcal{ALCQUIO}@$ are closed under substitutions.

To prove this corollary, one just has to reuse the proof of Theorem 6.2.1. Indeed, with the exception of nominals and the universal role that are present in all logics of the corollary, all constructors that appear on the right-hand side of a rule also appear on the left-hand side. The restriction of $\mathcal{M}$ and $\mathcal{M}'$ to the remaining rules ensures the termination of the system.

Those are not the only logics that are closed under substitutions. Indeed, it is possible to prove that in presence of $@$, $\mathcal{U}$ is no longer needed.

Corollary 6.2.2. *The logics $\mathcal{ALCO}@$, $\mathcal{ALCO}@\mathcal{Self}$, $\mathcal{ALCIO}@$, $\mathcal{ALCQO}@$, $\mathcal{ALCIO}@\mathcal{Self}$, $\mathcal{ALCQO}@\mathcal{Self}$, $\mathcal{ALCQIO}@$ are closed under substitutions.*

Once more, to prove this corollary, $\mathcal{T}$ is used. Only the rules that actually use the universal role are changed and thus all others are not repeated. One can easily see that all occurrences of U are of the form $\exists U.(\{i\} \sqcap C)$ or $\forall U.(\neg\{i\} \sqcup C)$. Those can be replaced with $@_i C$. That both have the same interpretation and that it doesn't not impede the algorithm to terminate is easily checked.

23 $(\bowtie\ n\ R\ \phi)[R := R + (i, j)] \rightsquigarrow$
$((\{i\} \sqcap @_j(\phi[R := R + (i, j)]) \sqcap \forall R.\neg\{j\}) \Rightarrow$
$(\bowtie\ (n-1)\ R\ \phi[R := R + (i, j)]))$
$\sqcap\ ((\neg\{i\} \sqcup @_j\neg(\phi[R := R + (i, j)]) \sqcup \exists R.\{j\}) \Rightarrow$
$(\bowtie\ n\ R\ \phi[R := R + (i, j)]))$

24 $(\bowtie\ n\ R^-\ \phi)[R := R + (i, j)] \rightsquigarrow$
$((\{j\} \sqcap @_i(\phi[R := R + (i, j)]) \sqcap \forall R^-.\neg\{i\}) \Rightarrow$
$(\bowtie\ (n-1)\ R^-\ \phi[R := R + (i, j)]))$
$\sqcap\ ((\neg\{j\} \sqcup @_i\neg(\phi[R := R + (i, j)]) \sqcup \exists R^-.\{i\}) \Rightarrow$
$(\bowtie\ n\ R^-\ \phi[R := R + (i, j)]))$

25 $(\bowtie\ n\ R\ \phi)[R := R - (i,j)] \rightsquigarrow$

$$\begin{array}{l}((\{i\} \sqcap @_j(\phi[R := R - (i,j)]) \sqcap \exists R.\{j\}) \quad \Rightarrow \\ (\bowtie\ (n+1)\ R\ \phi[R := R - (i,j)])) \\ \sqcap\ ((\neg\{i\} \sqcup @_j\neg(\phi[R := R - (i,j)]) \sqcup \forall R.\neg\{j\}) \ \Rightarrow \\ (\bowtie\ n\ R\ \phi[R := R - (i,j)]))\end{array}$$

26 $(\bowtie\ n\ R^-\ \phi)[R := R - (i,j)] \rightsquigarrow$

$$\begin{array}{l}((\{j\} \sqcap @_i(\phi[R := R - (i,j)]) \sqcap \exists R^-.\{i\}) \quad \Rightarrow \\ (\bowtie\ (n+1)\ R^-\ \phi[R := R - (i,j)])) \\ \sqcap\ ((\neg\{j\} \sqcup @_i\neg(\phi[R := R - (i,j)]) \sqcup \forall R^-.\neg\{i\}) \Rightarrow \\ (\bowtie\ n\ R^-\ \phi[R := R - (i,j)]))\end{array}$$

31 $(\exists R.\phi)[R := R + (i,j)] \rightsquigarrow (\{i\} \Rightarrow$
$@_j(\phi[R := R + (i,j)]) \sqcup \exists R.\phi[R := R + (i,j)])$
$\sqcap(\neg\{i\} \Rightarrow \exists R.(\phi[R := R + (i,j)]))$

32 $(\exists R^-.\phi)[R := R + (i,j)] \rightsquigarrow (\{j\} \Rightarrow$
$@_i(\phi[R := R + (i,j)]) \sqcup \exists R^-.\phi[R := R + (i,j)])$
$\sqcap(\neg\{j\} \Rightarrow \exists R^-.(\phi[R := R + (i,j)]))$

39 $(\forall R.\phi)[R := R + (i,j)] \rightsquigarrow (\{i\} \Rightarrow$
$\forall R.(\phi[R := R + (i,j)]) \sqcap @_j(\phi[R := R + (i,j)]))$
$\sqcap(\neg\{i\} \Rightarrow \forall R.(\phi[R := R + (i,j)]))$

40 $(\forall R^-.\phi)[R := R + (i,j)] \rightsquigarrow (\{j\} \Rightarrow$
$\forall R^-.(\phi[R := R + (i,j)]) \sqcap @_i(\phi[R := R + (i,j)]))$
$\sqcap(\neg\{j\} \Rightarrow \forall R^-.(\phi[R := R + (i,j)]))$

We have proven that the most expressive logics that we considered, namely those containing at least $\mathcal{O}$ and one of $\mathcal{U}$ or @, are closed under substitutions even though they may incur a blow-up in the size of the formulae during the translation phase.

6.3 Increased DLs

In this section, a family of Description Logics which are strictly less expressive than their extension with substitutions are introduced. In addition to the purely technical interest of sorting logics depending on whether or not they have the property of being closed under substitutions, studying less expressive logics is meaningful as the more expressive logics are known to have a greater complexity[65]. Moreover, the translation removing substitutions is itself exponential which makes the problem clearly intractable.

The first Description Logic that is considered is $\mathcal{ALC}$. $\mathcal{ALC}$ does not contain $\mathcal{O}$ nor $\mathcal{U}$ or @, thus ABoxes and TBoxes need to be used. Yet, we still require that no substitution appears in them.

Theorem 6.3.1. *$\mathcal{ALC}$ is not closed under substitutions.*

Intuitively, $\mathcal{ALC}$ is not strong enough to express whether a node has several successors or only one, and consequently to predict whether the deletion of an arc will leave this node with a successor or not.

More formally, to prove that the addition of substitutions to $\mathcal{ALC}$ strictly increases its expressiveness, we use a result from [24]. They prove that $\mathcal{ALC}$ concepts, as well as TBoxes and ABoxes, are stable under bisimilar interpretations

Theorem 6.3.2. *For every concept C of the logic $\mathcal{ALC}$, for every two interpretations $\mathcal{I}_1$ and $\mathcal{I}_2$, for every bisimulation relation Z between $\mathcal{I}_1$ and $\mathcal{I}_2$, if an element x_1 of $\mathcal{I}_1$ satisfies C and there is an element x_2 of $\mathcal{I}_2$ such that $x_1 Z x_2$, then x_2 satisfies C. The same can be said for the satisfiability of ABox and TBox assertions.*

Definition 6.3.1 (Bisimulation)**.** *Given a signature ($\boldsymbol{C}$, $\boldsymbol{R}$, $\boldsymbol{I}$) and two interpretations $\mathcal{I}$ and $\mathcal{J}$, a non-empty binary relation $\mathcal{Z} \subseteq (\Delta^{\mathcal{I}} \times \Delta^{\mathcal{J}})$ is a bisimulation if it satisfies:*

$(\mathcal{ALC}_1)$ $d_1 Z d_2 \implies \forall A \in \boldsymbol{C}, (d_1 \in A^{\mathcal{I}} \Leftrightarrow d_2 \in A^{\mathcal{J}})$

$(\mathcal{ALC}_2)$ $\forall R \in \boldsymbol{R}, (d_1 Z d_2 \wedge d_1 R^{\mathcal{I}} e_1 \implies \exists e_2 . d_2 R^{\mathcal{J}} e_2 \wedge e_1 Z e_2)$

$(\mathcal{ALC}_3)$ $\forall R \in \boldsymbol{R}, (d_1 Z d_2 \wedge d_2 R^{\mathcal{J}} e_2 \implies \exists e_1 . d_1 R^{\mathcal{I}} e_1 \wedge e_1 Z e_2)$

$(\mathcal{ALC}_4)$ $\forall i \in \boldsymbol{I}, i^{\mathcal{I}} Z i^{\mathcal{J}}$

The idea of the proof of Theorem 6.3.1 consists in building an interpretation $\mathcal{I}_1$ such that an $\mathcal{ALC}$-concept ϕ containing substitutions holds and $\mathcal{I}_2$ bisimilar to $\mathcal{M}_1$ where ϕ does not hold, thus proving that $\mathcal{ALC}\sigma$ is not as expressive as $\mathcal{ALC}$. Figure 6.5 depicts such interpretations for the concept name $C[C := C - i]$.

Let's check that $\mathcal{M}_1$ and $\mathcal{M}_2$ are bisimilar. As the only concept in $\mathbf{C}$ is A and $\mathbf{R}$ is empty, we verify the four axioms given in Definition 6.3.1:

$(\mathcal{ALC}_1)$ $d_1 Z d_3 \rightsquigarrow (d_1 \in C^{\mathcal{I}_1} \Leftrightarrow d_3 \in C^{\mathcal{I}_2})\checkmark$
$d_2 Z d_3 \rightsquigarrow (d_2 \in C^{\mathcal{I}_1} \Leftrightarrow d_3 \in C^{\mathcal{I}_2})\checkmark$

$(\mathcal{ALC}_2)$ $\mathbf{R} = \emptyset$ ✓.

$(\mathcal{ALC}_3)$ $\mathbf{R} = \emptyset$ ✓.

$(\mathcal{ALC}_4)$ $i^{\mathcal{I}_1} Z i^{\mathcal{I}_2}\checkmark$

So $\mathcal{I}_1$ and $\mathcal{I}_2$ are bisimilar.

However, after the removal of the individual i from the concept C, the concept C still holds in d_2 and does not hold in d_3. This shows that there is no equivalent concept to $C[C := C - i]$ in $\mathcal{ALC}$, which is enough to show that $\mathcal{ALC}$ is not closed under substitution.

One could wonder whether it is possible to find a way, using a TBox and an ABox to find something that would be equivalent to $C[C := C - i]$ in $\mathcal{ALC}$. As the signature is $(\{C\}, \emptyset, \{i\})$, there are only few choices available as individual assertions or concept inclusions.

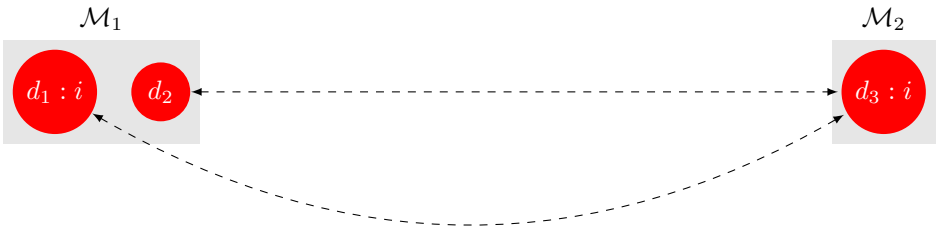


Figure 6.5: On the left, $\mathcal{M}_1$ is a model of $C[C := C - i]$. On the right, $\mathcal{M}_2$ is not. Nodes satisfying C are drawn in red. Plain arrows represent R, dashed arrows represent Z. The signature is $(\{C\}, \{R\}, \{i\})$.

What one would like to express is whether or not i is the only node satisfying C. The only possible individual assertions deal with the individual i. $\mathcal{ALC}$ does not allow to say that i is the only node satisfying C that way. On the other hand, the only way to express that the only node that satisfies C is i would be to use nominals. Thus, there is no way to build an ABox nor a TBox that would provide an equivalent for $C[C := C - i]$.

In [24], one may find other definitions of bisimulations for various Description Logics used to show that DL-concepts are stable. Below, we recall briefly these definitions. The names of the axioms indicate which logic they apply to. For instance, if one wants to deal with $\mathcal{ALCUI}$, one has to use axioms $(\mathcal{ALC}_1)$, $(\mathcal{ALC}_2)$, $(\mathcal{ALC}_3)$, $(\mathcal{ALC}_4)$, $(\mathcal{U}_1)$, $(\mathcal{U}_2)$, $(\mathcal{I}_1)$ and $(\mathcal{I}_2)$ to define bisimulations.

Definition 6.3.2. *The definition of bisimulation can be extended for more expressive logics by adding the following axioms:*

$(\mathcal{U}_1)$ $\forall d \in \Delta^{\mathcal{I}}, \exists d' \in \Delta^{\mathcal{J}}.dZd'$

$(\mathcal{U}_2)$ $\forall d' \in \Delta^{\mathcal{J}}, \exists d \in \Delta^{\mathcal{J}}.dZd'$

$(\mathcal{I}_1)$ $\forall R \in \boldsymbol{R}, (d_1 Z d_2 \wedge d_1 R^{-\mathcal{I}} e_1 \implies \exists e_2.d_2 R^{-\mathcal{J}} e_2 \wedge e_1 Z e_2)$

$(\mathcal{I}_2)$ $\forall R \in \boldsymbol{R}, (d_1 Z d_2 \wedge d_2 R^{-\mathcal{J}} e_2 \implies \exists e_1.d_1 R^{-\mathcal{J}} e_1 \wedge e_1 Z e_2)$

$(\mathcal{O})$ $\forall i \in \boldsymbol{I}, d_1 Z d_2 \implies (d_1 = i^{\mathcal{I}} \Leftrightarrow d_2 = i^{\mathcal{J}})$

$(\mathcal{Q})$ $\forall R \in \boldsymbol{R}, (d_1 Z d_2 \implies$ *Z is a bijection between the R-successors of d_1 and those of d_2).*

$(\mathcal{IQ})$ $\forall R \in \boldsymbol{R}, (d_1 Z d_2 \implies$ *Z is a bijection between the R^--successors of d_1 and those of d_2).*

$(\mathcal{S}elf)$ $\forall R \in \boldsymbol{R}, d_1 Z d_2 \implies (d_1 R^{\mathcal{I}} d_1 \Leftrightarrow d_2 R^{\mathcal{J}} d_2)$.

One could see that there is no rule for @. Actually, it is easy to prove that $(\mathcal{ALC}_4)$ is enough for @.

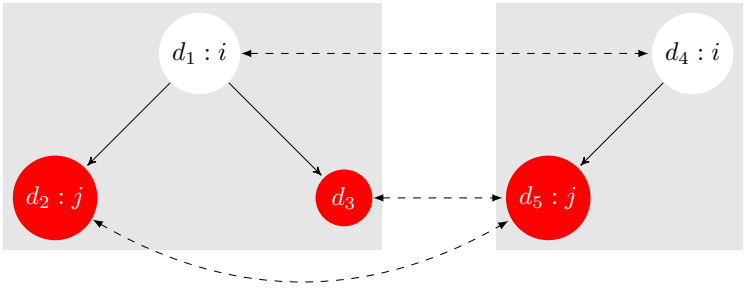


Figure 6.6: On the left, $\mathcal{I}_1$ is a model of $(\exists R.\phi)[R := R - (i,j)]$. On the right, $\mathcal{I}_2$ is not. Nodes satisfying ϕ are drawn in red.

Let us provide another example in Figure 6.6 for the concept $(\exists R.\phi)[R := R - (i,j)]$. Let us prove that $\mathcal{M}_1$ and $\mathcal{M}_2$ are bisimilar. As the only concept in $\mathbf{C}$ is C and the only role in $\mathbf{R}$ is R, we check:

- $(\mathcal{ALCC}_1)$
  - $d_1 Z d_4 \rightsquigarrow (d_1 \in C^{\mathcal{I}} \Leftrightarrow d_4 \in C^{\mathcal{J}})$✓
  - $d_2 Z d_5 \rightsquigarrow (d_2 \in C^{\mathcal{I}} \Leftrightarrow d_5 \in C^{\mathcal{J}})$✓
  - $d_3 Z d_5 \rightsquigarrow (d_3 \in C^{\mathcal{I}} \Leftrightarrow d_5 \in C^{\mathcal{J}})$✓
- $(\mathcal{ALCC}_2)$
  - $d_1 Z d_4 \wedge d_1 R^{\mathcal{I}} d_2 \rightsquigarrow d_4 R^{\mathcal{J}} d_5 \wedge d_2 Z d_5$✓
  - $d_1 Z d_4 \wedge d_1 R^{\mathcal{I}} d_3 \rightsquigarrow d_4 R^{\mathcal{J}} d_5 \wedge d_3 Z d_5$✓
- $(\mathcal{ALCC}_3)$
  - $d_1 Z d_4 \wedge d_4 R^{\mathcal{J}} d_5 \rightsquigarrow d_1 R^{\mathcal{I}} d_2 \wedge d_2 Z d_5$✓
- $(\mathcal{ALCC}_4)$
  - $i^{\mathcal{I}} Z i^{\mathcal{J}}$✓
  - $j^{\mathcal{I}} Z j^{\mathcal{J}}$✓

This proves that there is no equivalent concept to $(\exists R.C)[R := R - (i,j)]$ is $\mathcal{ALCC}$. Yet, it doesn't prove that it is not possible to find a combination of ABoxes and TBoxes that would be equivalent to that concept. Intuitively, one has to test whether i has j as only neighbour labeled C or not. Once more, individual assertions can only add facts about i and j and not other elements and it is not possible to say that j is the only neighbor of i labeled with C without nominals or counting quantifiers. The same way, concept inclusions do not provide a way to express this property.

With the same example, we can prove that $\mathcal{ALCU}$, $\mathcal{ALCS}elf$, $\mathcal{ALCI}$, $\mathcal{ALCUI}$, $\mathcal{ALCUS}elf$, $\mathcal{ALCIS}elf$ and $\mathcal{ALCUIS}elf$ are extended when substitutions are allowed.

- $(\mathcal{U}_1)$
  - $d_1 Z d_4$✓
  - $d_2 Z d_5$✓
  - $d_3 Z d_5$✓
- $(\mathcal{U}_2)$
  - $d_1 Z d_4$✓
  - $d_2 Z d_5$✓
- $(\mathcal{I}_1)$
  - $d_2 Z d_5 \wedge d_2 R^{-\mathcal{I}} d_1 \rightsquigarrow d_5 R^{-\mathcal{J}} d_4 \wedge d_1 Z d_4$✓
  - $d_3 Z d_5 \wedge d_3 R^{-\mathcal{I}} d_1 \rightsquigarrow d_5 R^{-\mathcal{J}} d_4 \wedge d_1 Z d_4$✓
- $(\mathcal{I}_2)$
  - $d_2 Z d_5 \wedge d_5 R^{-\mathcal{J}} d_4 \rightsquigarrow d_2 R^{-\mathcal{I}} d_1 \wedge d_1 Z d_4$✓
- $(\mathcal{S}elf)$
  - $d_1 Z d_4 \rightsquigarrow (d_1 R^{\mathcal{I}} d_1 \Leftrightarrow d_4 R^{\mathcal{J}} d_4)$✓
  - $d_2 Z d_5 \rightsquigarrow (d_2 R^{\mathcal{I}} d_2 \Leftrightarrow d_5 R^{\mathcal{J}} d_5)$✓
  - $d_3 Z d_5 \rightsquigarrow (d_3 R^{\mathcal{I}} d_3 \Leftrightarrow d_5 R^{\mathcal{J}} d_5)$✓

On the other hand, if $\mathcal{Q}$ was part of the logic, this would not be a bisimulation. Indeed, $d_1 Z d_4$ but Z is not a bijection between the R-neighbours of d_1 and those of d_4.

The example of Figure 6.5 allows to prove that the DLs given in the following corollary are not closed under substitutions. The reader may verify that the axioms of bisimulations corresponding to these logics are satisfed by Z.

- $(\mathcal{U}_1)$
  - $d_1 Z d_3$✓
  - $d_2 Z d_3$✓
- $(\mathcal{U}_2)$
  - $d_1 Z d_3$✓
- $(\mathcal{I}_1)$
  - $\mathbf{R} = \emptyset$ thus $\forall R \in \mathbf{R}.(dZd' \wedge dR^{-\mathcal{I}} e \Rightarrow \exists e'.d' R^{-\mathcal{J}} e' \wedge eZe'$✓

- $(\mathcal{I}_2)$
  - $\mathbf{R} = \emptyset$ thus $\forall R \in \mathbf{R}.(dZd' \wedge d'R^{-\mathcal{J}}e' \Rightarrow \exists e.dR^{-\mathcal{I}}e \wedge eZe'$✓
- $(\mathcal{Q})$
  - $\mathbf{R} = \emptyset$ thus $\forall R \in \mathbf{R}.(dZd' \Rightarrow Z$ is a bijection between the R-successors of d and those of d'✓
- $(\mathcal{IQ})$
  - $\mathbf{R} = \emptyset$ thus $\forall R \in \mathbf{R}.(dZd' \Rightarrow Z$ is a bijection between the R^--successors of d and those of d'✓
- $(\mathcal{S}elf)$
  - $\mathbf{R} = \emptyset$ thus $\forall R \in \mathbf{R}.dZd' \rightsquigarrow (dR^{\mathcal{I}}d \Leftrightarrow d'R^{\mathcal{J}}d')$✓

Corollary 6.3.1. *The logics $\mathcal{ALCU}$, $\mathcal{ALCS}elf$, $\mathcal{ALCI}$, $\mathcal{ALCQ}$, $\mathcal{ALCUI}$, $\mathcal{ALCUS}elf$, $\mathcal{ALCQU}$, $\mathcal{ALCIS}elf$, $\mathcal{ALCQS}elf$, $\mathcal{ALCQUI}$, $\mathcal{ALCQIS}elf$, $\mathcal{ALCQUS}elf$, $\mathcal{ALCUIS}elf$ and $\mathcal{ALCQUIS}elf$ are not closed under substitutions.*

Another question is whether a limiting a logic to ABoxes and TBoxes make it closed under substitution. The answer is no. Let us consider the concept inclusion $C \subseteq D$ where both C and D are concept names. Let us now consider $(C \subseteq D)[C := C - i]$. Intuitively, it is satisfied if $C \subseteq D$ or i is the only node $\in C \backslash D$. The only way to speak about i without using nominals is through individual assertions but they do not allow to express that it is the only one in a concept. Let us now consider $(i : C)[C := C + j]$. It is satisfied if $i : C$ or $i = j$. But it is not possible to express it without using a disjunction of ABoxes and TBoxes which is not possible in standard $\mathcal{ALC}$ or its extensions lacking $\mathcal{O}$.

If a DL contains $\mathcal{O}$, the example of Figure 6.6 and Figure 6.5 no longer are counter-examples because Z is not a bisimulation anymore. Indeed, for instance in Figure 6.5, d_2Zd_3 but $d_3 = i^{\mathcal{J}}$ whereas $d_2 \neq i^{\mathcal{I}}$ contradicting $(\mathcal{O})$.

6.4 The grey area

The technique that was used in the previous section cannot be applied directly in the case of the logic $\mathcal{ALCO}$. Indeed, we can show the following result.

Theorem 6.4.1. *$\mathcal{ALCO}\sigma$ concepts are stable under $\mathcal{ALCO}$-bisimulations.*

Proof. The idea is to work by induction on the concept constructors. We actually work with the more expressive $\mathcal{ALCOIQS}elf$.

- The concepts of $\mathcal{ALCOIQS}elf$ are know to be invariant for $\mathcal{ALCOIQS}elf$-bisimulations.
- If the concept is either $\bot\sigma$, $\{i\}\sigma$, $c\sigma$ or $(\exists R.Self)\sigma$, using the translations of Section 6.2, one can obtain an equivalent concept in $\mathcal{ALCOIQS}elf$.

- If the concept is $(\neg\phi)\sigma$, $(\phi \sqcup \psi)\sigma$ and $(\phi \sqcap \psi)\sigma$, this is proved by induction

- If $y \in ((\bowtie nR\phi)[C := C \pm i])^{\mathcal{I}}$ then, as the valuation of C does not affect the valuation of R, $y^{\mathcal{I}} \in \{z|\{w|(z,w) \in R^{\mathcal{I}} \sqcap w \in \phi[C := C \pm i]^{\mathcal{I}}\} \bowtie n\}$. By induction, for each w such that $w \in \phi[C := C \pm i]^{\mathcal{I}}$ are there exists w' such that $w' \in \phi[C := C \pm i]^{\mathcal{J}}$ and wZw'. Due to $(\mathcal{Q})$, there is a bijection between the w and the w' and the w' are R-neighbors of y'. Thus $y' \in ((\bowtie nR\phi)[C := C \pm i])^{\mathcal{J}}$.

- The same can be done the other way and for $(\bowtie nR^{-}\phi)[C := C \pm i]$, $(\bowtie nR^{-}\phi)[C := \psi]$, $(\bowtie nR\phi)[R' := R' \pm (i,j)]$, $(\bowtie nR^{-}\phi)[R' := R' \pm (i,j)]$, $(\bowtie nR\phi)[R' := Q]$, $(\bowtie nR^{-}\phi)[R' := Q]$, $(\bowtie nR\phi)[R' := (i,j)]$ and $(\bowtie nR^{-}\phi)[R' := (i,j)]$.

- If $y \in ((\bowtie nR\phi)[R := R + (i,j)])^{\mathcal{I}}$ then $y^{\mathcal{I}} \in (\{z|\{w|(z,w) \in R^{\mathcal{I}} \sqcap w \in \phi[R := R + (i,j)]^{\mathcal{I}}\} \cup \{i|j^{\mathcal{I}} \in \phi[R := R + (i,j)]^{\mathcal{I}}\}) \bowtie n\}$. There are two cases:

  - If $y \neq i^{\mathcal{I}}$ or $j^{\mathcal{I}} \notin (\phi[R := R + (i,j)])^{\mathcal{I}}$ or $(i^{\mathcal{I}}, j^{\mathcal{I}}) \in R^{\mathcal{I}}$ then $\{i|j^{\mathcal{I}} \in \phi[R := R+(i,j)]^{\mathcal{I}}\}$ is either empty or a subset of $\{w|(z,w) \in R^{\mathcal{I}} \sqcap w \in \phi[R := R + (i,j)]^{\mathcal{I}}\}$. In that case, due to rule $(\mathcal{Q})$ and the induction hypothesis, there is a bijection between the R-neighbors in $\mathcal{I}$ and $\mathcal{J}$ and the one satisfying $\phi[R := R + (i,j)]$ are thus the same number. Hence $y' \in ((\bowtie nR\phi)[R := R + (i,j)])^{\mathcal{J}}$
  - If not, the reasoning can be applied to all the R-neighbors satisfying $\phi[R := R + (i,j)]$ of y and, due to rule $(\mathcal{O})$ there is exactly one new neighbor added by the substitution on each side. Thus $y' \in ((\bowtie nR\phi)[R := R + (i,j)])^{\mathcal{J}}$.

- The same can be done the other way and for $(\bowtie nR^{-}\phi)[R := R + (i,j)]$.

- If $y \in ((\bowtie nR\phi)[R := R - (i,j)])^{\mathcal{I}}$ then $y \in (\{z|\{w|(z,w) \in R^{\mathcal{I}} \sqcap w \in \phi[R := R - (i,j)]^{\mathcal{I}}\} \backslash \{i|j^{\mathcal{I}} \in \phi[R := R - (i,j)]^{\mathcal{I}}\}) \bowtie n\}$. There are two cases:

  - If $y \neq i^{\mathcal{I}}$ or $j^{\mathcal{I}} \notin (\phi[R := R - (i,j)])^{\mathcal{I}}$ or $(i^{\mathcal{I}}, j^{\rangle}) \notin R^{\mathcal{I}}$ then $\{i|j^{\mathcal{I}} \in \phi[R := R - (i,j)]^{\mathcal{I}}\}$ is either empty or disjoint from $\{w|(z,w) \in R^{\mathcal{I}} \sqcap w \in \phi[R := R - (i,j)]^{\mathcal{I}}\}$. In that case, due to rule $(\mathcal{Q})$ and the induction hypothesis, there is a bijection between the R-neighbors in $\mathcal{I}$ and $\mathcal{J}$ and the one satisfying $\phi[R := R - (i,j)]$ are thus the same number. Hence $y' \in ((\bowtie nR\phi)[R := R - (i,j)])^{\mathcal{J}}$
  - If not, the reasoning can be applied to all the R-neighbors satisfying $\phi[R := R - (i,j)]$ of y and, due to rule $(\mathcal{O})$ there is exactly one new neighbor deleted by the substitution on each side. Thus $y' \in ((\bowtie nR\phi)[R := R - (i,j)])^{\mathcal{J}}$.

- The same can be done the other way and for $(\bowtie nR^{-}\phi)[R := R - (i,j)]$.

- If $y \in ((\bowtie nR\phi)[R := Q])^{\mathcal{I}}$ then $y \in (\bowtie nQ\phi[R := Q])^{\mathcal{I}}$. Let w be such that yQw and $w \in \phi[R := Q]^{\mathcal{I}}$ then there exists w' such that wZw' and $w' \in \phi[R := Q]^{\mathcal{J}}$. As there is a bijection between the w's and the w''s, $y' \in ((\bowtie nR\phi)[R := Q])^{\mathcal{I}}$.
- The same can be done the other way round and for $((\bowtie nR^{-}\phi)[R := Q])^{\mathcal{I}}$
- If $y \in ((\bowtie nR\phi)[R := (i,j)])^{\mathcal{I}}$ then either:
  - $y = i^{\mathcal{I}}$ and $j^{\mathcal{I}} \in \phi[R := (i,j)]^{\mathcal{I}}$ and then $y' = i^{\mathcal{J}}$ and $j^{\mathcal{J}} \in \phi[R := (i,j)]^{\mathcal{J}}$ and thus $y' \in ((\bowtie nR\phi)[R := (i,j)])^{\mathcal{J}}$
  - or, $y \neq i^{\mathcal{I}}$ or $j^{\mathcal{I}} \notin \phi[R := (i,j)]^{\mathcal{I}}$ and then $y' \neq i^{\mathcal{J}}$ or $j^{\mathcal{J}} \notin \phi[R := (i,j)]^{\mathcal{J}}$ and thus $y' \in ((\bowtie nR\phi)[R := (i,j)])^{\mathcal{J}}$
- The same can be done the other way round and for $((\bowtie nR^{-}\phi)[R := (i,j)])^{\mathcal{I}}$
- If $y \in ((\bowtie nR\phi)[i \gg^{in} j])^{\mathcal{I}}$ then if $y \in \{z|\{w|(z,w) \in (R^{\mathcal{I}} \backslash \{(z,i^{\mathcal{I}})\} \cup \{(z,j^{\mathcal{I}})|(z,i^{\mathcal{I}}) \in R^{\mathcal{I}}) \sqcap w \in \phi[i \gg^{in} j]^{\mathcal{I}}\} \bowtie n\}$. As there is a bijection between the R-neighbors of y satisfying $\phi[i \gg^{in} j]$ and those of y', if $i^{\mathcal{I}} \in \phi[i \gg^{in} j]^{\mathcal{I}}$ then $i^{\mathcal{J}} \in \phi[i \gg^{in} j]^{\mathcal{J}}$, and, if $j^{\mathcal{I}} \in \phi[i \gg^{in} j]^{\mathcal{I}}$ then $j^{\mathcal{J}} \in \phi[i \gg^{in} j]^{\mathcal{J}}$, there is a bijection between the elements of $\{w|(y,w) \in (R^{\mathcal{I}} \backslash \{(y,i^{\mathcal{I}})\}) \sqcap w \in \phi[i \gg^{in} j]^{\mathcal{I}}\}$ and those of $\{w|(y,w) \in (R^{\mathcal{I}} \backslash \{(y,i^{\mathcal{I}})\}) \sqcap w \in \phi[i \gg^{in} j]^{\mathcal{I}}\}$. Thus $y' \in ((\bowtie nR\phi)[i \gg^{in} j])^{\mathcal{J}}$.
- The same can be done the other way round and for $(\bowtie nR\phi)[i \gg^{out} j]$, $(\bowtie nR^{-}\phi)[i \gg^{in} j]$ and $(\bowtie nR^{-}\phi)[i \gg^{out} j]$
- $\exists R.\phi$ and $\forall R.\phi$ being special cases of $(\bowtie nR\phi)$, they can be treated as previously.

□

Theorem 6.4.1 could lead us to think that the addition of substitutions to $\mathcal{ALCO}$ does not increase the expressiveness of the logic. This is not true. We propose below another kind of binary relations that are able to distinguish between $\mathcal{ALCO}$ and $\mathcal{ALCO}\sigma$.

Definition 6.4.1. *Given two interpretations $\mathcal{I}$ and $\mathcal{J}$, a* rooted-bisimulation *between $\mathcal{I}$ and $\mathcal{J}$ rooted in (x,x') is a binary relation $\mathcal{Z} \subseteq (\Delta^{\mathcal{I}} \times \Delta^{\mathcal{J}})$ such that:*

$(\mathcal{ALC}_1)$ $d_1 Z d_2 \implies \forall C \in \boldsymbol{C}, (d_1 \in C^{\mathcal{I}} \Leftrightarrow d_2 \in C^{\mathcal{J}})$

$(\mathcal{ALC}_2)$ $\forall R \in \boldsymbol{R}, (d_1 Z d_2 \wedge d_1 R^{\mathcal{I}} e_1 \implies \exists e_2 . d_2 R^{\mathcal{J}} e_2 \wedge e_1 Z e_2)$

$(\mathcal{ALC}_3)$ $\forall R \in \boldsymbol{R}, (d_1 Z d_2 \wedge d_2 R^{\mathcal{J}} e_2 \implies \exists e_1 . d_1 R^{\mathcal{I}} e_1 \wedge e_1 Z e_2)$

$(r\mathcal{ALC}_4)$ xZx'

This definition is extended for logics containing $\mathcal{ALC}$ the same way as the definition of bisimulations.

The only difference with the definition of bisimulation is rule ($\mathcal{ALC}_4$) that is replaced with rule ($r\mathcal{ALC}_4$). It is easy to see that rooted-bisimulation are more general than bisimulations. Indeed, any bisimulation is equivalent to the union of rooted-bisimulations rooted in the nominals.

Definition 6.4.2. *A concept C is* stable under rooted-bisimulations *if, for any interpretations $\mathcal{I}$, $\mathcal{J}$, any rooted-bisimulation Z between $\mathcal{I}$, $\mathcal{J}$ rooted in (x,x') and any y reachable from x, there exists y' such that yZy' and $y \in C^{\mathcal{I}} \Leftrightarrow y' \in C^{\mathcal{J}}$.*

Theorem 6.4.2. *$\mathcal{ALCO}$-concepts are stable under rooted-bisimulations*

Before tackling Theorem 6.4.2, we show an useful property.

Theorem 6.4.3. *Given two interpretations $\mathcal{I}$ and $\mathcal{J}$ and a* rooted-bisimulation *between $\mathcal{I}$ and $\mathcal{J}$ rooted in (x,x') Z, if y is reachable from x then there exists y' reachable from x' such that yZy' and if z' is reachable from x' then there exists z reachable from x such that zZz'.*

Proof. This is shown by induction on the length of the path from x to y (resp. from x' to z').

From rule ($r\mathcal{ALC}_4$), xZx' thus the property is valid for paths of length 0.

Let's assume the property is valid for paths of length N. Let's pick y (resp. z') reachable such that the shortest path between x and y (resp. x' and z') if of length N + 1, that is there exists w (resp. w') such that there is a path of length N between x and w (resp. x' and w'). Due to the induction hypothesis, there exists v' (resp. v) reachable from x' (resp. x) such that wZv' (resp. vZw'). But, thanks to ($\mathcal{ALC}_2$) (resp. ($\mathcal{ALC}_3$)), that means that there exists y' (resp. z) such that $v'R^{\mathcal{J}}y'$ (resp. $vR^{\mathcal{I}}w$) and yZy' (resp. zZz'). Hence the property is valid for paths of length N + 1. □

Let us now prove Theorem 6.4.2.

Proof. Theorem 6.4.3 proves the existence, for each y reachable from r of y' reachable from r' such that yZy'.

We will do an induction on the constructors of $\mathcal{ALCO}$-concepts.

$\bot$ As no element belongs to $\bot^{\mathcal{I}}$ or $\bot^{\mathcal{J}}$, $y \in \bot^{\mathcal{I}} \Leftrightarrow y' \in \bot^{\mathcal{J}}$

c This is given by the rule ($\mathcal{ALC}_1$)

$\neg\, C$ By induction, if $y \in C^{\mathcal{I}} \Leftrightarrow y' \in C^{\mathcal{J}}$ then $y \in (\neg C)^{\mathcal{I}} \Leftrightarrow y' \in (\neg C)^{\mathcal{J}}$

$C \sqcap D$ By induction, if $y \in C^{\mathcal{I}} \Leftrightarrow y' \in C^{\mathcal{J}}$ and $y \in D^{\mathcal{I}} \Leftrightarrow y' \in D^{\mathcal{J}}$ then $y \in (C \sqcap D)^{\mathcal{I}} \Leftrightarrow y' \in (C \sqcap D)^{\mathcal{J}}$

$C \sqcup D$ By induction, if $y \in C^{\mathcal{I}} \Leftrightarrow y' \in C^{\mathcal{J}}$ and $y \in D^{\mathcal{I}} \Leftrightarrow y' \in D^{\mathcal{J}}$ then $y \in (C \sqcup D)^{\mathcal{I}} \Leftrightarrow y' \in (C \sqcup D)^{\mathcal{J}}$

$\exists R.C$ If $y \in (\exists R.C)^{\mathcal{I}}$ then there exists z such that $y\ R^{\mathcal{I}}\ z$ and $z \in C^{\mathcal{I}}$. From $(\mathcal{ALC}_2)$, there exists z' such that $y'\ R^{\mathcal{J}}\ z'$ and zZz'. Then, by induction, $z' \in C^{\mathcal{J}}$ and thus $y' \in (\exists R.C)^{\mathcal{J}}$. The same can be done the other way using $(\mathcal{ALC}_3)$.

$\forall R.C$ If $y \in (\forall R.C)^{\mathcal{I}}$ then for all z such that $y\ R^{\mathcal{I}}\ z$, $z \in C^{\mathcal{I}}$. From $(\mathcal{ALC}_2)$, there exists z' such that $y'\ R^{\mathcal{J}}\ z'$ and zZz'. Then, by induction, $z' \in C^{\mathcal{J}}$ and thus $y' \in (\forall R.C)^{\mathcal{J}}$. The same can be done the other way using $(\mathcal{ALC}_3)$.

o This is given by the rule $\mathcal{O}$

$\exists R.Self$ If $y \in (\exists R.Self)^{\mathcal{I}}$ then $yR^{\mathcal{I}}y$. From rule $(\mathcal{S}elf)$, $y'R^{\mathcal{J}}y'$ thus $y' \in (\exists R.Self)^{\mathcal{J}}$

$\exists R^-.C$ If $y \in (\exists R.C)^{\mathcal{I}}$ then there exists z such that $z\ R^{\mathcal{I}}\ y$ and $z \in C^{\mathcal{I}}$. From rule $(\mathcal{I}_1)$, there exists z' such that $y'\ R^{\mathcal{J}}\ z'$ and zZz'. Then, by induction, $z' \in C^{\mathcal{J}}$ and thus $y' \in (\exists R.C)^{\mathcal{J}}$. The same can be done the other way using $(\mathcal{I}_2)$.

$\forall R^-.C$ If $y \in (\forall R.C)^{\mathcal{I}}$ then for all z such that $z\ R^{\mathcal{I}}\ y$, $z \in C^{\mathcal{I}}$. From $(\mathcal{I}_1)$, there exists z' such that $y'\ R^{\mathcal{J}}\ z'$ and zZz'. Then, by induction, $z' \in C^{\mathcal{J}}$ and thus $y' \in (\forall R.C)^{\mathcal{J}}$. The same can be done the other way using $(\mathcal{I}_2)$.

$(\bowtie nRC)$ If $y \in (\bowtie nRC)^{\mathcal{I}}$ then the cardinality of $\{z | yR^{\mathcal{I}}z \sqcap z \in C^{\mathcal{I}}\} \bowtie n$. From $(\mathcal{Q})$, there exist exactly as many z' such that $y'\ R^{\mathcal{J}}\ z'$ and zZz'. Then, by induction, $z' \in C^{\mathcal{J}}$ and thus $y' \in (\bowtie nRC)^{\mathcal{J}}$. The same can be done the other way.

$(\bowtie nR^-C)$ If $y \in (\bowtie nR^-C)^{\mathcal{I}}$ then the cardinality of $\{z | zR^{\mathcal{I}}y \sqcap z \in C^{\mathcal{I}}\} \bowtie n$. From $(\mathcal{IQ})$, there exist exactly as many z' such that $z'\ R^{\mathcal{J}}\ y'$ and zZz'. Then, by induction, $z' \in C^{\mathcal{J}}$ and thus $y' \in (\bowtie nR^-C)^{\mathcal{J}}$. The same can be done the other way.

□

Theorem 6.4.4. *$\mathcal{ALCO}\sigma$-concepts are not stable under rooted-bisimulations.*

Figure 6.7 provides a counter-example illustrating Theorem 6.4.4.

Let us prove that Z in this counter-example is indeed a rooted-bismulation rooted in (d_1, d_3).

- $(\mathcal{ALC}_1)$
  - $d_1 Z d_3 \implies (d_1 \in C^{\mathcal{I}_1} \Leftrightarrow d_3 \in C^{\mathcal{I}_2})$✓ as $\mathbf{C} = \{C\}$.
- $(\mathcal{ALC}_2)$
  - As $\mathbf{R} = \{R\}$ and $R^{\mathcal{I}_1} = \emptyset$, $d_1 Z d_3 \wedge d_1 R^{\mathcal{I}_1} e \implies \exists e'.d_3 R^{\mathcal{I}_2} e' \wedge eZe'$✓
- $(\mathcal{ALC}_3)$

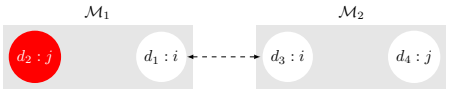


Figure 6.7: On the left, $\mathcal{I}_1$ is a model of $(\forall R.C)[R := R + (i, j)]$. On the right, $\mathcal{I}_2$ is not. Nodes satisfying C are drawn in red. Plain arrows represent R. Dashed arrows represent Z. The signature is $\{\{C\}, \{R\}, \{i, j\}\}$. $\mathcal{M}_1$ and $\mathcal{M}_2$ are (d_1, d_3)-rooted-bisimilar.

  - As $\mathbf{R} = \{R\}$ and $R^{\mathcal{I}_2} = \emptyset$, $d_1 Z d_3 \wedge d_3 R^{\mathcal{I}_2} e' \implies \exists e. d_1 R^{\mathcal{I}_1} e \wedge e Z e'$✓

- $(r\mathcal{ALC}_4)$

  - $d_1 Z d_3$✓

This result can be extended to the logics $\mathcal{ALCOI}$, $\mathcal{ALCOQ}$, $\mathcal{ALCOSelf}$, $\mathcal{ALCOIQ}$, $\mathcal{ALCOISelf}$ and $\mathcal{ALCOIQSelf}$ by adding the exact same conditions from Definition 6.3.2. The exact same example of Figure 6.7 can be used to prove these logics version of Theorem 6.4.4

- $(\mathcal{I}_1)$

  - As $\mathbf{R} = \{R\}$ and $R^{\mathcal{I}_1} = \emptyset$, $d_1 Z d_3 \wedge d_1 R^{-\mathcal{I}_1} e \implies \exists e'. d_3 R^{-\mathcal{I}_2} e' \wedge e Z e'$✓

- $(\mathcal{I}_2)$

  - As $\mathbf{R} = \{R\}$ and $R^{\mathcal{I}_2} = \emptyset$, $d_1 Z d_3 \wedge d_3 R^{-\mathcal{I}_2} e' \implies \exists e. d_1 R^{\mathcal{I}_1} e \wedge e Z e'$✓

- $(\mathcal{Q})$

  - As $\mathbf{R} = \{R\}$ and $R^{\mathcal{I}_1} = R^{\mathcal{I}_2} = \emptyset$, $d_1 Z d_3 \implies Z$ is a bijection between the R-successors of d_1 and those of d_3✓

- $(\mathcal{IQ})$

  - As $\mathbf{R} = \{R\}$ and $R^{\mathcal{I}_1} = R^{\mathcal{I}_2} = \emptyset$, $d_1 Z d_3 \implies Z$ is a bijection between the R^--successors of d_1 and those of d_3✓

- $(\mathcal{Self})$

  - As $\mathbf{R} = \{R\}$ and $R^{\mathcal{I}_1} = R^{\mathcal{I}_2} = \emptyset$, $d_1 Z d_3 \Rightarrow (d_1 R^{\mathcal{I}_1} d_1 \Leftrightarrow d_3 R^{\mathcal{I}_2} d_3)$✓

6.5 Application to the hospital

Let us now illustrate how a program is proven correct in one of the Description Logics presented so far. The logic $\mathcal{ALCQUIO}$ is chosen as it is one of the smallest of those that we have proven closed under substitutions that allows us to express the properties we need.

Let us first give a look at the properties that were stated in Section 3.3 and check that they can be expressed. The first property is that "each member of the medical staff is either a nurse or a physician but not both". This can be translated to $\forall U.(\text{MS} \Rightarrow ((\text{NU} \sqcap \neg\ \text{PH}) \sqcup (\neg\ \text{NU} \sqcap \text{PH})))$. The second property is that "all patients and all medical staffers are persons". This is equivalent to $\forall U.(\text{MS} \sqcup \text{PA} \Rightarrow \text{PE})$. The third property states that each person that can write in a folder can also read it. This property cannot be expressed in $\mathcal{ALCQUIO}$ or any other description logic without role inclusion. It is thus dropped for now. The fourth property has the same problem. It states that "each person that can read a folder about a patient treats that patient". The fifth property says that "Only medical staffers can treat persons and only patients can be treated". It can be translated as $\forall U.(\exists \mathit{treats}.\top \Rightarrow \text{MS}) \sqcap \forall U.(\exists \mathit{treats}^{-}.\top \Rightarrow \text{PA})$. Finally, the sixth property is "Every patient has exactly one referent physician" that can be written as $\forall U.(\text{PA} \Rightarrow (= 1 \mathit{ref\_phys} \top))$.

In this example, illustrated in Figure 6.8, the fourth transformation that deletes a physician and re-affects all his patients to another physician in the same department is used. For the sake of simplicity none of the precondition, postcondition and invariant feature all the properties that are required in Section 3.3. The statement is the same as the one in Figure 4.3 but the precondition and postcondition are different. The chosen precondition is $\exists U.(\{\text{PH}_1\} \sqcap PH \sqcap \neg\{\text{PH}_2\}) \sqcap \exists U.(\{\text{PH}_2\} \sqcap \text{MS}) \sqcap \forall U.(\exists\ \mathit{treats}.\ \top \Rightarrow \text{MS})$ that can be read as PH_1 is an existing physician, PH_2 is a member of the medical staff and only members of the medical staff can treat people. The chosen postcondition is $\exists U.(\{\text{PH}_1\} \sqcap \neg PH) \sqcap \forall U.(\exists\ \mathit{treats}.\top \Rightarrow MS)$ that means that PH_1 no longer is a physician and that only members of the medical staff can treat people. The invariants for the loops are inv_1 and inv_2. inv_2 is set to be equal to the postcondition while inv_1 is $\exists U.(\{\text{PH}_1\} \sqcap \neg \text{PH} \sqcap \neg\{\text{PH}_2\}) \sqcap \exists U.(\{\text{PH}_2\} \sqcap \text{MS}) \sqcap \forall U.(\exists\ \mathit{treats}.\top \Rightarrow MS)$.

One also has to translate the conditions into $\mathcal{ALCQUIO}$. The condition of the "if" statement is $\exists d.((\text{PH}_1, d) : \mathit{works\_in} \wedge (\text{PH}_2, d) : \mathit{works\_in})$. $\mathcal{ALCQUIO}$ does not allow variables. Using a quantifier, one can still express this condition as $c_0 \equiv \exists U.(\exists \mathit{works\_in}^{-}.(\text{PH}_1) \sqcap \exists \mathit{works\_in}^{-}.(\text{PH}_2))$. Using the same inversion of the edge, the condition of the first loop $\exists p.(\text{PH}_1, p) : \mathit{treats}$ can be written as $c_1 \equiv \exists U.(\exists \mathit{treats}^{-}.\text{PH}_1)$. Finally, the last condition $\exists p.(p, \text{PH}_1) : \mathit{ref\_phys}$ becomes $c_2 \equiv \exists U.(\exists \mathit{ref\_phys}.\text{PH}_1)$.

Let us now check that the specification $(Pre, s, Post)$ is correct. The correctness formula is $(Pre \Rightarrow wp(s, Post)) \sqcap vc(s, Post)$. As s is of the form `if` c_0 `then` s_0, $wp(s, Post) = c_0 \Rightarrow wp(s_0, Post)$ and $vc(s, Post) = vc(s_0, Post)$. Thus the correctness formula is $(Pre \sqcap c_0 \Rightarrow wp(s_0, Post)) \sqcap vc(s_0, Post)$. In turn, s_0 is of the form $PH := PH\text{-}\ \text{PH}_1; s_1$ and thus $wp(s_0, Post) = wp(s_1, Post)[PH := PH\text{-}\ \text{PH}_1]$ and $vc(s_0, Post) = vc(s_1, Post)$ thus the correctness formula is $(Pre \sqcap c_0 \Rightarrow$

Pre: $\exists U.(\{\text{PH}_1\} \sqcap \text{PH} \sqcap \neg\{\text{PH}_2\}) \sqcap \exists U.(\{\text{PH}_2\} \sqcap \text{MS}) \sqcap \forall U.(\exists treats.\top \Rightarrow MS)$
if $\exists U.(\exists works\_in^{-}.\{\text{PH}_1\}) \sqcap \exists U.(\exists works\_in^{-}.\{\text{PH}_2\})$ **then**
$PH := PH$ - PH_1;
while $\exists U.(\exists treats^{-}.\{\text{PH}_1\})$ **do**
select p **with** $\exists U.(\{p\} \sqcap treats^{-}.\{\text{PH}_1\})$;
$treats := treats$ - (PH_1,p);
$treats := treats$ + (PH_2,p);
while $\exists U.(\exists ref\_phys.\{\text{PH}_1\})$ **do**
select p **with** $\exists U.(\{p\} \sqcap \exists ref\_phys.\{\text{PH}_1\})$;
$ref\_phys := ref\_phys$ - (p,PH_1);
$ref\_phys := ref\_phys$ + (p,PH_2);
$MS := MS$- PH_1;
Post: $\exists U.(\{\text{PH}_1\} \sqcap \neg\text{PH}) \sqcap \forall U.(\exists treats.\top \Rightarrow \text{MS})$

Figure 6.8: An example using the fourth transformation

$wp(s_1, Post)[PH := PH$- $\text{PH}_1]) \sqcap vc(s_1, Post)$. Then $s_1 = s_1'; s_2$ where s_1' is of the form **while** c_1 **do** s_{10}. Thus $wp(s_1, Post) = inv_1$ and the correctness formula becomes $(Pre \sqcap c_0 \Rightarrow inv_1[PH := PH$- $\text{PH}_1]) \sqcap vc(s_1, Post)$.

Let us now focus on $Pre \sqcap c_0 \Rightarrow inv_1[PH := PH$- $\text{PH}_1]$. From the rule 15 introduced in Section 6.2, $inv_1[PH := PH$- $\text{PH}_1] = (\exists U.(\{\text{PH}_1\} \sqcap \neg\text{PH} \sqcap \neg\{\text{PH}_2\}))$ $[PH := PH$- $\text{PH}_1] \sqcap (\exists U.(\{\text{PH}_2\} \sqcap MS))[PH := PH$- $\text{PH}_1] \sqcap (\forall U.(\exists\ treats.\top \Rightarrow MS))$ $[PH := PH$- $\text{PH}_1]$.

Let us now work with $(\forall U.(\exists\ treats.\top \Rightarrow \text{MS}))[PH := PH$- $\text{PH}_1]$. From rule 90, its is equal to $\forall U.(\exists\ treats.\top \Rightarrow \text{MS})[PH := PH$- $\text{PH}_1]$. One can now apply rule 13 and 14 to get $\forall U.((\exists\ treats.\top)[PH := PH$- $\text{PH}_1] \Rightarrow \text{MS}[PH := PH$- $\text{PH}_1]$. Applying the rules 68 and then 1 to the first part and 4 to the second part yields $\forall U.(\exists\ treats.\top \Rightarrow \text{MS})$.

Meanwhile, by applying rule 68, $(\exists U.(\{\text{PH}_1\} \sqcap \neg\text{PH} \neg\{\text{PH}_2\}))[PH := PH$- $\text{PH}_1] = \exists U.((\{\text{PH}_1\} \sqcap \neg\text{PH} \sqcap \neg\{\text{PH}_2\}))[PH := PH$- $\text{PH}_1]$. Let us now apply rule 15 to get $\exists U.((\{\text{PH}_1\}[PH := PH$- $\text{PH}_1] \sqcap (\neg\ PH)[PH := PH$- $\text{PH}_1] \sqcap (\neg\{\text{PH}_2\})[PH :=$ PH- $\text{PH}_1]))$. Then, by applying the rules 2, 10 and 13, one obtains $\exists U.(\{\text{PH}_1\} \sqcap$ $(\neg\ PH \sqcup \{\text{PH}_1\}) \sqcap \neg\{\text{PH}_2\})$. One can now observe that this is equivalent to $\exists U.(\{$ $\text{PH}_1\} \sqcap \neg\{\text{PH}_2\})$. The same can be done with $\exists U.(\{\text{PH}_2\} \sqcap \text{MS})$ to prove that it is left unchanged by the substitution. Thus $inv_1[PH := PH$- $\text{PH}_1] = \exists U.(\{\text{PH}_1\} \sqcap$ $\neg\{\text{PH}_2\})\ \sqcap \exists U.(\{\text{PH}_2\} \sqcap \text{MS}) \sqcap \forall U.(\exists treats.\top \Rightarrow \text{MS})$. This is trivially implied by $Pre = \exists U.(\{\text{PH}_1\} \sqcap \text{PH} \sqcap \{\text{PH}_2\}) \sqcap \exists U.(\{\text{PH}_2\} \sqcap \text{MS}) \sqcap \forall U.(\exists treats.\top \Rightarrow \text{MS})$.

One thus has to prove that $vc(s_1, Post)$ is valid. As $vc(s_1, Post) = (inv_1 \sqcap c_1 \Rightarrow wp(s_{10}, inv_1)) \sqcap (inv_1 \sqcap \neg c_1 \Rightarrow wp(s_2, Post)) \sqcap vc(s_{10}, inv_1) \sqcap vc(s_2, Post)$, let us split the study of the validity of $vc(s_1, Post)$ in four parts.

Let us start with $vc(s_{10}, inv_1)$. As $s_{10} = s_{11}; s_{12}; s_{13}$, $vc(s_{10}, inv_1) = vc(s_{11}, wp(s_{12}; s_{13}, inv_1)) \sqcap vc(s_{12}, wp(s_{13}, inv_1)) \sqcap vc(s_{13}, inv_1)$. But all of these

vcs are equal to $\top$ and thus $vc(s_{10}, inv_1)$ is trivially valid.

Let us now study $inv_1 \sqcap c_1 \Rightarrow wp(s_{10}, inv_1)$. As $s_{10} = s_{11}; s_{12}; s_{13}$, $wp(s_{10}, inv_1) = wp(s_{11}, wp(s_{12}; s_{13}, inv_1)) = \forall p.(\exists U.(\{p\} \sqcap \exists treats^{-}.\{\text{PH}_1\}) \Rightarrow wp(s_{12}; s_{13}, inv_1)$. Then, as both s_{12} and s_{13} are atomic transformations, $wp(s_{12}; s_{13}, inv_1) = inv_1[treats := treats + (\text{PH}_2, p)][treats := treats - (\text{PH}_1, p)]$. Thus, one has to prove that $\forall p.(inv_1 \wedge c_1 \wedge \exists U.(\{p\} \sqcap \exists treats^{-}.\{\text{PH}_1\}) \Rightarrow inv_1[treats := treats + (\text{PH}_2, p)][treats := treats - (\text{PH}_1, p)]$.

Let us compute $inv_1[treats := treats + (\text{PH}_2, p)][treats := treats - (\text{PH}_1, p)]$. Once more, it is straightforward that $\exists U.(\{\text{PH}_1\} \sqcap \neg\text{PH} \sqcap \neg\{\text{PH}_2\}) \sqcap \exists U.(\{\text{PH}_2\} \sqcap \text{MS})$ is left unmodified by the substitutions. It is alos obviously implied by inv_1.

One thus focuses on $\forall U.(\exists TR.\top \Rightarrow \text{MS})[treats := treats + (\text{PH}_2, p)][treats := treats - (\text{PH}_1, p)]$. It is easily proved that it is equivalent to $\forall U.((\exists TR.\top)[treats := treats + (\text{PH}_2, p)][treats := treats - (\text{PH}_1, p)] \Rightarrow \text{MS})$. By applying rule 31, and then rule 1, one gets that $(\exists TR.\top)[treats := treats + (\text{PH}_2, p)][treats := treats - (\text{PH}_1, p)]$ is equivalent to $((\{\text{PH}_2\} \Rightarrow \exists U.\{p\} \sqcup \exists treats.\top) \sqcap (\neg\{\text{PH}_2\} \Rightarrow \exists treats.\top)[treats := treats - (\text{PH}_1, p)]$. Then, by applying rule 33, one can prove that it is equivalent to $(\{\text{PH}_2\} \Rightarrow \exists U.\{p\} \sqcup ((\{\text{PH}_1\} \Rightarrow \exists treats.\neg\{p\}) \wedge (\neg\{\text{PH}_1\} \Rightarrow \exists treats.\top))) \sqcap (\neg\{\text{PH}_2\} \Rightarrow ((\{\text{PH}_1\} \Rightarrow \exists treats.\neg\{p\}) \wedge (\neg\{\text{PH}_1\} \Rightarrow \exists treats.\top)))$.

In order to prove that this is implied by $inv_1 \sqcap c_1 \sqcap \exists U.(\{p\} \sqcap \exists treats^{-}.\{\text{PH}_1\}$, one has to distinguish four cases:

- $\{\text{PH}_1\} \sqcap \{\text{PH}_2\}$ is satisfied: This is made impossible by $\exists U.(\{\text{PH}_1\} \sqcap \neg\{\text{PH}_2\})$.
- $\{\text{PH}_1\} \sqcap \neg\{\text{PH}_2\}$ is satisfied: Then $(\forall U.\exists treats.\top \Rightarrow \text{MS}) \Rightarrow (\forall U.\exists treats.\neg\{p\} \Rightarrow \text{MS})$ is what has to be proved valid and it obviously is.
- $\{\text{PH}_2\} \sqcap \neg\{\text{PH}_1\}$ is satisfied: Then $\exists U.(\{\text{PH}_2\} \sqcap \text{MS}) \Rightarrow (\{\text{PH}_2\} \sqcap \neg\{\text{PH}_1\} \Rightarrow \text{MS})$ is what has to be proved valid and it obviously is.
- $\neg\{\text{PH}_1\} \sqcap \neg\{\text{PH}_2\}$ is satisfied: Then $(\forall U.\exists treats.\top \Rightarrow \text{MS}) \Rightarrow (\forall U.\exists treats.\top \Rightarrow \text{MS})$ is what has to be proved valid and it obviously is.

Thus $inv_1 \sqcap c_1 \Rightarrow wp(s_{10}, inv_1)$ is valid.

Let us now study $inv_1 \sqcap \neg c_1 \Rightarrow wp(s_2, Post)$. As s_2 is of the form `while` c_2 `do`, $wp(s_2, Post) = inv_2$. As $inv_1 \Rightarrow inv_2$, this is valid.

One now focuses on $vc(s_2, Post)$. It is equal to $(inv_2 \sqcap c_2 \Rightarrow wp(s_{20}; s_{21}; s_{22}, inv_2)) \sqcap (inv_2 \sqcap \neg c_2 \Rightarrow wp(s_3, Post)) \sqcap vc(s_{20}; s_{21}; s_{22}, Post)$. As previously, as $s_{20} =$ `select` p `with` $\exists U.(\{p\} \sqcap \exists ref\_phys.\{\text{PH}_1\})$, $s_{21} = ref\_phys := ref\_phys - (p, \text{PH}_1)$ and $s_{22} = ref\_phys := ref\_phys + (p, \text{PH}_2)$, $vc(s_{20}; s_{21}; s_{22}, Post) = \top$.

Let us focus first on the validity of $inv_2 \sqcap c_2 \Rightarrow wp(s_{20}; s_{21}; s_{22}, inv_2)$. As $s_{20} =$ `select` p `with` $\exists U.(\{p\} \sqcap \exists ref\_phys.\{\text{PH}_1\})$, $s_{21} = ref\_phys := ref\_phys - (p, \text{PH}_1)$ and $s_{22} = ref\_phys := ref\_phys + (p, \text{PH}_2)$, $wp(s_{20}; s_{21}; s_{22}, inv_2) = \forall p.\exists U.(\{p\} \sqcap \exists ref\_phys.\{\text{PH}_1\}) \Rightarrow inv_2[ref\_phys := ref\_phys + (p, \text{PH}_1)][ref\_phys := ref\_phys - (p, \text{PH}_2)]$. As ref_phys never occurs in inv_2, it is left unmodified by the substitutions. Thus, one has to prove that $\forall p.(inv_2 \sqcap c_2 \sqcap \exists U.(\{p\} \sqcap \exists ref\_phys.\{\text{PH}_1\} \Rightarrow inv_2)$ is valid. It is obviously the case as $inv_2 \Rightarrow inv_2$.

Finally, let us study $inv_2 \sqcap \neg c_2 \Rightarrow wp(s_3, Post)$. As $s_3 = \text{MS} := \text{MS} - \text{PH}_1$, $wp(s_3, Post) = Post[\text{MS} := \text{MS} - \text{PH}_1]$. As does not occur in $Post$, $wp(s_3, Post) = Post$. Then, as $inv_2 = Post$, $inv_2 \sqcap \neg c_2 \Rightarrow wp(s_3, Post)$ is obviously valid.

6.6 Conclusion

This chapter dealt with Description Logics and how one could use them in the graph tranformation framework that we have heretofore introduced. As Description Logics have several key arguments justifying their use, they were a prime candidate as a logic used to prove the correctness of graph transformations. Nevertheless, several of their shortcomings were pointed out.

The least expressive logics considered, $\mathcal{ALC}$ and its extensions without nominals and some way to express properties of remote nodes, despite having very low complexity are not closed under substitutions and thus require the definition of alternate algorithm to prove the correctness of graph transformation.

The most expressive logics, $\mathcal{ALCOU}$, $\mathcal{ALCO}@$ and their extensions are closed under substitutions but the rewriting system generates an exponential blow-up in the size of the formulae that is highly detrimental.

Nonetheless, there exists several Description Logics that we didn't consider. For instance, $\mathcal{ALCOIQ}br$, defined in [1], allows for more complex roles. Due to these, it is possible to get rid of substitutions without the exponential blow-up on the size of the formulae which makes it a better fit.

Chapter 7

Using a logic with reachability: $\mathcal{C}2\mathcal{PDL}$

We previously introduced the description logics as a possible choice as logics used in order to transform graphs. Nevertheless, these logics have some shortcomings. In particular, one may want to use reachability conditions that are impossible to express in Description Logics or, for that matter, first-order logic. In order to tackle that kind of properties, we use a logic that is much closer to dynamic logics. This logic was used in [14].

Using dynamic logics makes sense. Indeed, the models of dynamic logics are graphs and thinking about programs is usual in dynamic logics. To be more precise, in dynamic logics, models often represent executions where nodes are the various states of the data and edges represent the execution of programs. Nonetheless, we do not use dynamic logics that way but as a formal system to describe graphs. That is our edges are actual edges and programs are used to modify the models.

After formally introducing the syntax of the logic, we prove that it is closed under substitutions. We also prove that it is decidable. Finally, we introduce restrictions on the rules so that it is possible to express the applicability condition.

7.1 Syntax

In this chapter, we introduce $\mathcal{C}2\mathcal{PDL}$, a logic that we introduce to define pre- and postconditions and to compute the weakest precondition of a specification. It is a mix of Converse Propositional Dynamic Logic [27] and Combinatory Propositional Dynamic Logic [54], both commonly known as $\mathcal{CPDL}$. $\mathcal{C}2\mathcal{PDL}$ contains elements of Propositional Dynamic Logic, that allows one to define complex role constructors, and Hybrid Logic, which allows one to use the power of nominals. $\mathcal{C}2\mathcal{PDL}$ further extends the $\mathcal{CPDL}$s in two ways: it splits the universe into elements that are part of the model notwithstanding

the substitutions and elements that may be created by an action or that have been deleted; it also extends the notion of universal role to "total" roles over subsets of the universe in order to be able to deal with these modifications of the universe.

Definition 7.1.1 (Syntax of $\mathcal{C}2\mathcal{PDL}$)**.** *Given three countably infinite and pairwise disjoint alphabets* Σ*, the set of* names*,* Φ_0*, the set of* atomic propositions*,* Π_0*, the set of* atomic programs*, the language of* $\mathcal{C}2\mathcal{PDL}$ *is composed of* formulas *and* programs*. We partition the set of names* Σ *into two countably infinite alphabets* Σ_1 *and* Σ_2 *such that* $\Sigma_1 \cup \Sigma_2 = \Sigma$ *and* $\Sigma_1 \cap \Sigma_2 = \emptyset$*. Formulas* ϕ *and programs* α *are defined as:*

$\phi \quad := \quad i \mid \phi_0 \mid \neg\phi \mid \phi \vee \phi \mid \langle\alpha\rangle\phi$

$\alpha \quad := \quad \alpha_0 \mid \nu_S \mid \alpha;\alpha \mid \alpha \cup \alpha \mid \alpha^* \mid \alpha^- \mid \phi?$

where $i \in \Sigma$*,* $\phi_0 \in \Phi_0$*,* $\alpha_0 \in \Pi_0$ *and* $S \subseteq \Sigma$*.*

We denote by Π *the set of programs and by* Φ *the set of formulas. As usual,* $\phi \wedge \psi$ *stands for* $\neg(\neg\phi \vee \neg\psi)$ *and* $[\alpha]\phi$ *stands for* $\neg(\langle\alpha\rangle\neg\phi)$*. We also write simply* ν *instead of* ν_Σ*.*

For now, the splitting of Σ seems artificial. It is actually grounded in the use we want to make of the logic. Roughly speaking, Σ_1 stands for the names that are used in "the" current model whereas Σ_2 stands for the names that may be used in the future (or have been used in the past but do not participate to the current model).

Definition 7.1.2 (Model)**.** *A* model *is a tuple* $\mathcal{M} = (M, R, \chi, V)$ *where* M *is a set called the* universe*,* $\chi : \Sigma \to M$ *is a surjective mapping such that* $\chi(\Sigma_E) \cap \chi(\Sigma_O) = \emptyset$*,* $R : \Pi \to \mathcal{P}(M^2)$ *and* $V : \Phi \to \mathcal{P}(M)$ *are mappings such that:*

- *For each* $\alpha_0 \in \Pi_0$*,* $R(\alpha_0) \in \mathcal{P}(\chi(\Sigma_E)^2)$
- *For each* $i \in \Sigma$*,* $V(i) = \{\chi(i)\}$
- $R(\nu_S) = \chi(S)^2$ *for* $S \subseteq \Sigma$
- *For each* $\phi_0 \in \Phi_0$*,* $V(\phi_0) \in \mathcal{P}(\chi(\Sigma_E))$
- $R(\alpha \cup \beta) = R(\alpha) \cup R(\beta)$
- $V(\neg A) = M \backslash V(A)$
- $R(A?) = \{(s, s) | s \in V(A)\}$
- $V(A \vee B) = V(A) \cup V(B)$
- $R(\alpha^-) = \{(s, t) | (t, s) \in R(\alpha)\}$
- $V(\langle\alpha\rangle A) = \{s | \exists t \in M.((s, t) \in R(\alpha) \wedge t \in V(A))\}$

- $R(\alpha^*) = \bigcup_{k<\omega} R(\alpha^k)$ *where* α^k *stands for the sequence* $\alpha; \ldots; \alpha$ *of length* k
- $R(\alpha;\beta) = \{(s,t)|\exists v.((s,v) \in R(\alpha) \wedge (v,t) \in R(\beta))\}$

In the following, we write $sR_\alpha t$ *for* $(s,t) \in R(\alpha)$

Fig.7.1 gives an example of models. $\mathcal{C}2\mathcal{PDL}$ will be used in this work to label nodes and to express properties of attributed graphs.

From now on, we will only consider graphs over the alphabet (Φ,Π). Given a graph $G = (N, E, \phi_N, \phi_E, s, t)$ and a formula ϕ, we say that $G \models \phi$ if there exists $n \in N$ such that $n \models \phi$ where the universe M is the set of nodes N. R and V are defined as usual: for $\phi_0 \in \Phi_0$ (resp. $\pi_0 \in \Pi_0$), $V(\phi_0) = \{x \in N | \phi_0 \in L_N(x)\}$ (resp. $R(\pi_0) = \{(x,y) \in N^2 | \exists e \in E.s(e) = x \wedge t(e) = y \wedge L_E(e) = \pi_0\}$). R and V are then extended to non-atomic propositions and programs following the same rules defined in the models. As usual, a formula ϕ is satisfiable if there exists a graph G such that $G \models \phi$ and unsatisfiable otherwise and it is valid if for all models G, $G \models \phi$ and invalid otherwise. We denote by S a subset of $\mathcal{C}$ which consists of names such that for each name $s \in S$ there is at most one node $n \in N$ such that $n \models s$. One may remark that all models can be considered as graphs. The converse is false.

Often, we will write $i : C$ instead of $i : \{C\}$ to say that node i is labelled with the formula C.

Attributed graphs where all nodes are named will be called *named graphs*. This notion of graphs will be used in the proofs.

Definition 7.1.3 (Named Graph)**.** *A* named graph G *is an attributed graph such that the set of names* $S \subseteq C$ *satisfies:*

$(a)\forall s \in S.\exists n \in N.s \in L_N(n)$ $(b)\forall n \in N.L_N(n) \cap S \neq \emptyset$
$(c)\forall n, n' \in N, n \neq n', L_N(n) \cap L_N(n') \cap S = \emptyset$

Notation: From (a), (b) and (c), it is obvious that, as each name labels at least one node, each node is labeled by at least one name and each name labels at most one node, it is possible to define two functions θ and μ such that $\forall s \in S, \theta(s)$ is the node named s and $\forall n \in N, \mu(n)$ is a name of n. This allows to define χ and thus named graphs and models are equivalent structures. We will thus consider from now on that formulae are interpreted over named graphs. Figure 7.1 shows a model and a counter-model of the formula $[\nu](1 \Rightarrow [R^-][R^{-*}]\neg 1)$.

Example 7.1.1. *Let us now come back to the running example of the Hospital, presented in Section 3, by illustrating some of the properties.*

- *"Each member of the medical staff is either a nurse or a physician but not both"* $\rightsquigarrow$ $[\nu](MS \Rightarrow (NU \wedge \neg PH) \wedge (\neg MS \wedge PH)$.
- *"All patients and all medical staffers are persons"* $\rightsquigarrow$ $[\nu]((PA \vee MS) \Rightarrow PE)$

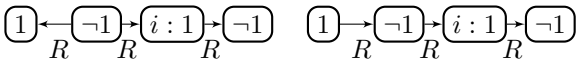


Figure 7.1: Model and counter-model. All nodes of the left graph satisfy the formula $[\nu](1 \Rightarrow [R^-][R^{-*}]\neg 1)$. This is not the case for the graph given on the right since $i \not\models (1 \Rightarrow [R^-][R^{-*}]\neg 1)$.

- *"Each person that can write in a folder can also read it" cannot be expressed as it needs to keep track of the person and the file*
- *"Each person that can read a folder about a patient treats that patien" cannot be expressed as it needs to keep track of the person and the patient*
- *"Only medical staffers can treat persons and only patients can be treated"* $\rightsquigarrow [\nu]((\langle \text{treats} \rangle\ PE \Rightarrow MS) \wedge (\langle \text{treats}^- \rangle\ \top \Rightarrow PA))$
- *"Every patient has exactly one referent physician" cannot be expressed as it is not possible to count neighbours*
- *"Every file references, albeit maybe indirectly, the Hospital Policy file"* $\rightsquigarrow [\nu](FI \Rightarrow \langle \text{reference}^* \rangle \text{HP})$

7.2 Closure under substitution

In this section, we start looking at whether $\mathcal{C}2\mathcal{PDL}$ satisfies the requirements that have been outlined in Chapter 5. We first prove that $\mathcal{C}2\mathcal{PDL}$ is closed under substitution.

First, though, we define *well-formed* formulae. Indeed, some sequences of substitutions do not make sense. For instance, $\phi[new\_node(i)][C := C+i]$ would create a node i after modifying it and thus either the node i didn't exist when its label changed, which shouldn't be possible, or it existed and it doesn't make sense to create it anew.

Definition 7.2.1 (Well-formed formula)**.** *A formula is said to be well-formed if it is possible to find a set* $E \subseteq \Sigma$ *such that the following inference rules are respected:*

$$\frac{\phi^{E \cup \{i\}} \quad i \notin E}{\phi[add(i)]^E} \qquad \frac{\phi^{E-\{i\}} \quad i \in E}{\phi[del(i)]^E} \qquad \frac{\phi^E \quad \theta \neq add(i), \theta \neq del(i) \quad i,j \in E}{\phi[\theta]^E}$$

$$\frac{}{i^{\Sigma_1}} \qquad \frac{}{\phi_0^{\Sigma_1}} \qquad \frac{\phi^E}{(\neg\phi)^E} \qquad \frac{\phi_1^E \quad \phi_2^E}{(\phi_1 \vee \phi_2)^E}$$

We now work to prove that well-formed formulae are closed under substitutions.

Theorem 7.2.1. *$\mathcal{C}2\mathcal{PDL}$ is closed under substitutions.*

To prove the theorem, we introduce a rewriting system $\mathcal{RS}$. Its goal is to transform any formula where substitutions occur into a substitution-free formula. It is not always possible to do that in one step. The rewriting system thus contains rules that remove substitutions completely and other rules that moves the substitution inward. Each rule is such that the left-hand side and the right-hand side are equivalent.

Let $\sigma, \sigma' \in \Theta$, $\sigma \in \{[\alpha_0 := \alpha], [\alpha_0 := (i,j)], [\alpha_0 := \alpha_0 + (i,j)], [\alpha_0 := \alpha_0 - (i,j)], [new\_node(i)], [new\_edge(i,j)], [del\_edge(i,j)], [i \gg^{in} j], [i \gg^{out} j]\}$, ϕ_0 and $\phi_1 \in \Phi_0$, $\phi_0 \neq \phi_1$, ϕ and $\psi \in \Phi$, $i \in \Sigma$ and $\alpha \in \Pi$ then $\mathcal{RS}_\phi$ is:

Rule ϕ_1 : $\top\sigma \rightsquigarrow \top$

Rule ϕ_2 : $i\sigma \rightsquigarrow i$

Rule ϕ_3 : $\phi_0\sigma' \rightsquigarrow \phi_0$

Rule ϕ_4 : $\phi_0[\phi_1 := \phi_0 \pm i] \rightsquigarrow \phi_0$

Rule ϕ_5 : $\phi_0[\phi_0 := \phi_0 + i] \rightsquigarrow \phi_0 \vee i$

Rule ϕ_6 : $\phi_0[\phi_0 := \phi_0 - i] \rightsquigarrow \phi_0 \wedge \neg i$

Rule ϕ_7 : $\phi_0[\phi_1 := \phi] \rightsquigarrow \phi_0$

Rule ϕ_8 : $\phi_0[\phi_0 := \phi] \rightsquigarrow \phi$

Rule ϕ_9 : $\phi_0[\phi_1 := i] \rightsquigarrow \phi_0$

Rule ϕ_{10} : $\phi_0[\phi_0 := i] \rightsquigarrow i$

Rule ϕ_{11} : $\phi_0[del\_node(i)] \rightsquigarrow \phi_0 \wedge \neg i$

Rule ϕ_{12} : $(\neg\phi)\sigma \rightsquigarrow \neg(\phi\sigma)$

Rule ϕ_{13} : $(\phi \vee \psi)\sigma \rightsquigarrow (\phi\sigma) \vee (\psi\sigma)$

Rule ϕ_{14}: $(\langle\alpha\rangle\phi)\sigma \rightsquigarrow \langle\alpha\sigma\rangle(\phi\sigma)$

We now introduce rewriting rules allowing to get rid of the substitutions occurring in programs. Let $\sigma, \sigma', \sigma'' \in \Theta$, $\sigma' \in \{[\phi_0 := \phi], [\phi_0 := i], [\phi_0 := \phi_0 \pm i], [new\_node(i)]\}$, $\sigma'' \notin \{[new\_node(i)], [del\_node(i)]\}$, $S \subseteq \Sigma$, ϕ_0 and $\phi_1 \in \Phi_0$ such that $\phi_0 \neq \phi_1$, ϕ and $\psi \in \Phi$, $\alpha_0, \alpha_1 \in \Pi_0$, $\alpha_0 \neq \alpha_1$ and α, and $\beta \in \Pi$ and $i, j \in \Sigma$, then $\mathcal{RS}_\alpha$ is:

Rule α_1 : $\alpha_0\sigma' \rightsquigarrow \alpha_0$

Rule α_2 : $\alpha_0[\alpha_1 := \alpha_1 \pm (i,j)] \rightsquigarrow \alpha_0$

Rule α_3 : $\alpha_0[\alpha_0 := \alpha_0 + (i,j)] \rightsquigarrow \alpha_0 \cup (i?; \nu_{\sigma_1}; j?)$

Rule α_4 : $\alpha_0[\alpha_0 := \alpha_0 - (i,j)] \rightsquigarrow (\neg i)?; \alpha_0 \cup \alpha_0; (\neg j)?$

Rule α_5 : $\alpha_0[\alpha_1 := \alpha] \rightsquigarrow \alpha_0$

Rule α_6 : $\alpha_0[\alpha_0 := \alpha] \rightsquigarrow \alpha$

Rule α_7 : $\alpha_0[\alpha_1 := (i,j)] \rightsquigarrow \alpha_0$

Rule α_8 : $\alpha_0[\alpha_0 := (i,j)] \rightsquigarrow (i?; \nu_{\sigma_1}; j?)$

Rule α_9 : $\alpha_0[del\_node(i)] \rightsquigarrow (\neg i)?; \alpha_0; (\neg i)?$

Rule α_{10} : $\alpha_0[new\_edge(i,j)] \rightsquigarrow \alpha_0$

Rule α_{11} : $\alpha_0[del\_edge(i,j)] \rightsquigarrow (\neg i)?; \alpha_0 \cup \alpha_0; (\neg j)?$

Rule α_{12} : $\alpha_0[i \gg^{in} j] \rightsquigarrow \alpha_0; ((\neg i)? \cup i?; \nu; j?)$

Rule α_{13} : $\alpha_0[i \gg^{out} j] \rightsquigarrow (\neg i?; (\neg j)? \cup j?; \nu_{\Sigma_E}; i?); \alpha_0$

Rule α_{14} : $\nu_S \sigma'' \rightsquigarrow \nu_S$

Rule α_{15} : $\nu_S[new\_node(i)] \rightsquigarrow \nu_{S[new\_node(i)]}$

Rule α_{16} : $\nu_S[del\_node(i)] \rightsquigarrow \nu_{S[del\_node(i)]}$

Rule α_{17} : $(\alpha;\beta)\sigma \rightsquigarrow (\alpha\sigma);(\beta\sigma)$

Rule α_{18} : $(\alpha \cup \beta)\sigma \rightsquigarrow (\alpha\sigma) \cup (\beta\sigma)$

Rule α_{19} : $(\alpha^-)\sigma \rightsquigarrow (\alpha\sigma)^-$

Rule α_{20} : $(\alpha^*)\sigma \rightsquigarrow (\alpha\sigma)^*$

Rule α_{21} : $(\phi?)\sigma \rightsquigarrow (\phi\sigma)?$

The rules α_{15} and α_{16} introduce substitutions that affect sets and thus forces the introduction of new rules. Let $i \in \Sigma$, $S_1, S_2 \subseteq \Sigma$:

Rule S_1 : $\Sigma_1[add(i)] \rightsquigarrow \Sigma_1 \cup \{i\}$

Rule S_2 : $\Sigma_1[del(i)] \rightsquigarrow \Sigma_1 \cap \overline{\{i\}}$

Rule S_3 : $\Sigma_2[add(i)] \rightsquigarrow \Sigma_2 \cap \overline{\{i\}}$

Rule S_4 : $\Sigma_2[del(i)] \rightsquigarrow \Sigma_2 \cup \{i\}$

Rule S_5 : $(S_1 \cup S_2)\sigma \rightsquigarrow S_1\sigma \cup S_2\sigma$

Rule S_6 : $(S_1 \cap S_2)\sigma \rightsquigarrow S_1\sigma \cap S_2\sigma$

Rule S_7 : $\overline{S_1}\sigma \rightsquigarrow \overline{S_1\sigma}$

Rule S_8 : $\{i\}\sigma \rightsquigarrow \{i\}$

We now prove that these rules are correct. We first prove that they are correct on atomic roles and concepts.

Lemma 4. *Let $\mathcal{R}$ be one of rules ϕ_1 - ϕ_{11}, A_r be the right-hand side and A_l be the left-hand side of $\mathcal{R}$. Given any model $\mathcal{M} = (M, R, \chi, V)$, $V(A_l) = V(A_r)$.*

Proof. The proof uses the model $\mathcal{M}' = (M, R', \chi, V')$ obtained form $\mathcal{M}$ by using the definitions in Section 3.1.

Rule ϕ_1 : As $V(\top)$ is independent of the definition of V, R, Σ_1 and Σ_2, $V(\top\sigma) = V(\top)$.

Rule ϕ_2 : As nodes are never renamed, $V(i\sigma) = V(i)$.

Rule ϕ_3 : As σ' does not modify modifies $V(\phi_0)$, $V(\phi_0\sigma') = V(\phi)$.

Rule ϕ_4 : As only $V(\phi_1)$ is modified, $V(\phi_0[\phi_1 := \phi_1 \pm i]) = V(\phi_0)$.

Rule ϕ_5 : As $V(\phi_0[\phi_0 := \phi_0 + i]) = V'(\phi_0) = \{\chi(i)\} \cup V(\phi_0)$, $V(\phi_0[\phi_0 := \phi_0 + i]) = V(\phi_0 \vee i_1)$.

Rule ϕ_6 : As $V(\phi_0[\phi_0 := \phi_0 - i]) = V'(\phi_0) = V(\phi_0)\backslash\{\chi(i)\}$, $V(\phi_0[\phi_0 := \phi_0 - i]) = V(\phi_0 \wedge \neg i)$.

Rule ϕ_7 : As only $V(\phi_1)$ is modified, $V(\phi_0[\phi_1 := \phi]) = V(\phi_0)$.

Rule ϕ_8 : As $V(\phi_0[\phi_0 := \phi]) = V'(\phi_0)$ and $V'(\phi_0) = V(\phi)$, $V(\phi_0[\phi_0 := \phi]) = V(\phi)$.

Rule ϕ_9 : As only $V(\phi_1)$ is modified, $V(\phi_0[\phi_1 := i]) = V(\phi_0)$.

Rule ϕ_{10} : As $V(\phi_0[\phi_0 := i]) = V'(\phi_0) = \{\chi(i)\}$, $V(\phi_0[\phi_0 := i]) = V(i)$.

Rule ϕ_{11} : As $V(\phi_0[del\_node(i)]) = V'(\phi) = V(\phi)\backslash\{\chi(i)\}$, $V(\phi_0[del\_node(i)]) = V(\phi_0 \wedge \neg i_1)$.

□

We now do the same thing with the atomic programs.

Lemma 5. *Let $\mathcal{R}$ be one of rules α_1 - α_{13}, A_r be the right-hand side and A_l be the left-hand side of $\mathcal{R}$. Given any model $\mathcal{M} = (M, R, \chi, V)$, $R(A_l) = R(A_r)$.*

Proof. The proof uses the model $\mathcal{M}' = (M, R', \chi, V')$ obtained form $\mathcal{M}$ by using the definitions in Section 3.1.

Rule α_1 : As σ does not modify $R(\alpha_0)$, $R(\alpha_0\sigma') = R(\alpha_0)$.

Rule α_2 : As only $R(\alpha_1)$ is modified, $R(\alpha_0[\alpha_1 := \alpha_1 \pm (i, j)]) = R(\alpha_0)$.

Rule α_3 : As $R(\alpha_0[\alpha_0 := \alpha_0 + (i, j)]) = R'(\alpha_0) = R(\alpha_0) \cup \{\chi(i), \chi(j)\}$, $R(\alpha_0[\alpha_0 := \alpha_0 + (i, j)]) = R(\alpha_0 \cup (i?; \nu_{\Sigma_1}; j?))$.

Rule α_4 : As $R(\alpha_0[\alpha_0 := \alpha_0 - (i,j)]) = R'(\alpha_0) = R(\alpha_0)\backslash\{\chi(i), \chi(j)\}$, $R(\alpha_0[\alpha_0 := \alpha_0 - (i,j)]) = R((\neg i)?; \alpha_0 \cup \alpha_0; (\neg j)?)$.

Rule α_5 : As only $R(\alpha_1)$ is modified, $R(\alpha_0[\alpha_1 := \alpha]) = R(\alpha_0)$.

Rule α_6 : As $R(\alpha_0[\alpha_0 := \alpha]) = R'(\alpha_0) = R(\alpha)$, $R(\alpha_0[\alpha_0 := \alpha]) = R(\alpha)$.

Rule α_7 : As only $R(\alpha_1)$ is modified, $R(\alpha_0[\alpha_1 := (i,j)]) = R(\alpha_0)$.

Rule α_8 : As $R(\alpha_0[\alpha_0 := (i,j)]) = R'(\alpha_0) = \{\chi(i), \chi(j)\}$, $R(\alpha_0[\alpha_0 := (i,j)]) = R(i?; \nu_{\Sigma_1}; j?)$.

Rule α_9 As $R(\alpha_0[del\_node(i)]) = R'(\alpha_0) = R(\alpha_0)\backslash(\{(\chi(i), m')\} \cup \{(m', \chi(i))\})$, $R(\alpha_0[del\_node(i)]) = R((\neg i?); \alpha_0; (\neg i)?)$.

Rule α_{10} As $[new\_edge(i,j)$ does not modify $R(\alpha_0)$, $R(\alpha_0[new\_edge(i,j)) = R(\alpha_0)$.

Rule α_{11} : As $R(\alpha_0[del\_edge(i,j)]) = R'(\alpha_0) = R(\alpha_0)\backslash\{\chi(i), \chi(j)\}$, $R(\alpha_0[del\_edge(i,j)]) = R((\neg i)?; \alpha_0 \cup \alpha_0; (\neg j)?)$.

Rule α_{12} : As $R(\alpha_0[i \gg^{in} j]) = R'(\alpha_0) = R(\alpha_0)\backslash\{(m', i) \in R(\alpha_0)\} \cup \{(m', j) | (m', i) \in R(\alpha_0)\}$, $R(\alpha_0[i \gg^{in} j]) = R(\alpha_0; ((\neg i)? \cup (i?; \nu_{\Sigma_1}; j?)))$.

Rule α_{13} : As $R(\alpha_0[i \gg^{out} j]) = R'(\alpha_0) = R(\alpha_0)\backslash\{(i, m') \in R(\alpha_0)\} \cup \{(j, m') | (i, m') \in R(\alpha_0)\}$, $R(\alpha_0[i \gg^{out} j]) = R(\neg i?; ((\neg j)? \cup (j?; \nu_{\Sigma_1}; i?)); \alpha_0)$.

□

We now prove that the rewriting of the sets is correct.

Proof. Rule S_1 : As i_2 is added to Σ_1, $\chi(\Sigma_1[add(i_2)]) = \chi(\Sigma_1) \cup \{i_2\}$

Rule S_2 : As i_2 is deleted from Σ_1, $\chi(\Sigma_1[del(i_1)]) = \chi(\Sigma_1) \cap \overline{\{i_1\}}$

Rule S_3 : As i_1 is deleted from Σ_2, $\chi(\Sigma_2[add(i_2)]) = \chi(\Sigma_2) \cap \overline{\{i_2\}}$

Rule S_4 : As i_1 is added to Σ_2, $\chi(\Sigma_2[del(i_1)]) = \chi(\Sigma_2) \cup \{i_i\}$

Rule S_5 : As only Σ_1 and Σ_2 are modified by σ, $\chi((S_1 \cup S_2)\sigma) = \chi(S_1\sigma) \cup \chi(S_2\sigma)$

Rule S_6 : As only Σ_1 and Σ_2 are modified by σ, $\chi((S_1 \cap S_2)\sigma) = \chi(S_1\sigma) \cap \chi(S_2\sigma)$

Rule S_7 : As only Σ_1 and Σ_2 are modified by σ, $\chi(\overline{S_2}\sigma) = \chi(\overline{S_1\sigma})$

Rule S_8 : As only Σ_1 and Σ_2 are modified by σ, $\chi(\{i\}\sigma) \rightsquigarrow \{i\}$

□

We now do the same with the other constructors:

Lemma 6. *Let $\mathcal{R}$ be one of rules ϕ_{12} - ϕ_{14}, A_r be the righthand side and A_l be the lefthand side of $\mathcal{R}$. Given any model $\mathcal{M} = (M, R, \chi, V)$, $V(A_l) = V(A_r)$.*

Proof. The proof uses the model $\mathcal{M}' = (M, R', \chi, V')$ obtained form $\mathcal{M}$ by using the definitions in Section 3.1.

Rule ϕ_{12} : $V((\neg\phi)\sigma) = V'(\neg\phi) = M \cap \overline{\{V'(\phi)\}} = V(\neg(\phi\sigma))$.

Rule ϕ_{13} : $V((\phi \vee \psi)\sigma) = V'(\phi \vee \psi) = V'(\phi) \cup V'(\psi) = V((\phi\sigma) \vee (\psi\sigma))$.

Rule ϕ_{14} : $V((\langle\alpha\rangle\phi)\sigma) = V'(\langle\alpha\rangle\phi) = \{s|\exists t \in M.((s,t) \in R'(\alpha) \wedge t \in V'(\phi)\} = V(\langle\alpha\sigma\rangle(\phi\sigma))$.

□

Lemma 7. *Let $\mathcal{R}$ be one of rules α_{14} - α_{21}, A_r be the righthand side and A_l be the lefthand side of $\mathcal{R}$. Given any model $\mathcal{M} = (M, R, \chi, V)$, $R(A_l) = R(A_r)$.*

Proof. The proof uses the model $\mathcal{M}' = (M, R', \chi, V')$ obtained form $\mathcal{M}$ by using the definitions in Section 3.1.

Rule α_{14} : As σ" does not modify S, $R(\nu_S\sigma") = R(\nu_S)$.

Rule α_{15} : $R(\nu_S[new\_node(i)) = R'(\nu_S) = \{(s,t)|(s,t) \in S[new\_node(i)]\} = R(\nu_{S[new\_node(i)]})$.

Rule α_{16} : $R(\nu_S[del\_node(i)) = R'(\nu_S) = \{(s,t)|(s,t) \in S[del\_node(i)]\} = R(\nu_{S[del\_node(i)]})$.

Rule α_{17} : $R((\alpha;\beta)\sigma) = R'(\alpha;\beta) = \{(s,t)|\exists v.((s,v) \in R'(\alpha) \wedge (v,t) \in R'(\beta)\} = R((\alpha\sigma);(\beta\sigma))$.

Rule α_{18} : $R((\alpha \cup \beta)\sigma) = R'(\alpha \cup \beta) = R'(\alpha) \cup R'(\beta) = R((\alpha\sigma) \cup (\beta\sigma))$.

Rule α_{19} : $R((\alpha^-)\sigma) = R'(\alpha^-) = \{(s,t)|(t,s) \in R'(\alpha)\} = R((\alpha\sigma)^-)$.

Rule α_{20} : $R((\alpha^*)\sigma) = R'(\alpha^*) = \bigcup_{k\leq\omega} R'(\alpha^k) = R((\alpha\sigma)^*)$.

Rule α_{21} : $R((\phi?)\sigma) = R'(\phi?) = \{(s,s)|s \in V'(\phi)\} = R((\phi\sigma)?)$.

□

7.3 Decidability

Theorem 7.3.1. *Given a formula ϕ of $\mathcal{C}2\mathcal{PDL}$, the satisfiability and the validity of ϕ are decidable.*

In order to take into account the new constructors, we propose the following axiom schemes and rules derived from the ones for $\mathcal{CPDL}$ that we prove sound and complete. We will also prove the decidability of the satisfiability (resp. validity) problem for $\mathcal{C}2\mathcal{PDL}$.

- PDL axioms:

  (Bool) All boolean tautologies

(□) $[\alpha](A \to B) \to ([\alpha]A \to [\alpha]B)$

(;) $< \alpha;\beta > A \leftrightarrow < \alpha >< \beta > A$

(∪) $< \alpha \cup \beta > A \leftrightarrow < \alpha > A \vee < \beta > A$

(?) $< A? > B \leftrightarrow A \wedge B$

(*) $< \alpha^* > A \leftrightarrow A \vee < \alpha >< \alpha^* > A$

(-) $< \alpha > [\alpha^-]A \leftrightarrow A$

- Names

  (Σ1) $< \nu_\Sigma > c$

  (Σ2) $< \nu_\Sigma > (c \wedge A) \to [\nu_\Sigma](c \to A)$

- Universal programs

  (ν_S1) $\forall c, d \in S.c \to < \nu_S > d$

  (ν_S2) $\forall \{c, d\} \not\subseteq S.c \to [\nu_S]\neg d$

  (ν_S3) $< \nu_S >< \nu_S > A \to < \nu_S > A$

  (ν_S4) $A \to [\nu_S] < \nu_S > A$

  (ν_Σ1) $A \to < \nu_\Sigma > A$

  (ν_Σ2) $< \alpha > A \to < \nu_\Sigma > A$

- Optional nodes

  (Σ_O1) $\forall c \in \Sigma_O, \forall \phi \in \Phi_0.c \to \neg\phi$

  (Σ_O2) $\forall c \in \Sigma_O, \forall \alpha \in \Pi_0.c \to [\alpha]\bot \wedge [\alpha^-]\bot$

- Rules:
  We give 5 deductive rules:

  (Ax) If A is an axiom, $\vdash A$.

  (MP) If $\vdash A$ and $\vdash A \to B$, then $\vdash B$

  (Ind) If $\vdash [\gamma][\alpha^k]A$, for all $k < \omega$, then $\vdash [\gamma][\alpha^*]A$

  (Cov) If $\vdash [\gamma]\neg c$, for all $c \in \Sigma$, then $\vdash [\gamma]\bot$

  (Nec) If $\vdash A$, then $\vdash [\nu_\Sigma]A$

Theorem 7.3.2 (Soundness)**.** *If* $\vdash A$ *then* $\vDash A$.

Proof. This is a straightforward induction on $\vdash$:

- Every node of the model $\mathcal{M}$ satisfies the boolean tautology A. Hence, $\mathcal{M} \vDash A$.

- Let m be a node of the model $\mathcal{M}$ then:

  - Either $\exists m'$ such that $(m, m') \in R(\alpha)$ and $m' \in V(A \wedge \neg B)$ and thus $m \in V(< \alpha > (A \wedge \neg B))$,

– or $\forall m'.(m,m') \notin R(\alpha)$ or $m' \in V(\neg A \vee B)$. Then either $\exists m''$ such that $(m,m'') \in R(\alpha)$ and $m'' \in V(\neg A)$ and thus $m \in V(< \alpha > \neg A)$,

– or $\forall m'.(m,m') \notin R(\alpha)$ or $m' \in V(B)$ and then $m \in V([\alpha]B)$.

In all possible cases, $m \in V(< \alpha > (A \wedge \neg B) \vee < \alpha > \neg A \vee [\alpha]B))$. Thus $m \in V([\alpha](A \to B) \to ([\alpha]A \to [\alpha]B))$.

- Let m be a node of the model $\mathcal{M}$ then:

  – Either $\exists m', m''$ such that $(m,m') \in R(\alpha)$ and $(m',m'') \in R(\beta)$ and $m'' \in V(A)$, that is $m \in V(< \alpha >< \beta > A)$, and thus $\exists m''$ such that $(m,m'') \in R(\alpha;\beta)$ and $m'' \in V(A)$ that is $m \in V(< \alpha;\beta > A)$,

  – or $\forall m', m''$, $(m,m') \notin R(\alpha)$ or $(m',m'') \notin R(\beta)$ or $m'' \notin V(A)$, that is $m \in V([\alpha][\beta]\neg A)$, and thus $\forall m''.(m,m'') \notin R(\alpha;\beta)$ or $m'' \notin V(A)$ that is $m \in V([\alpha;\beta]\neg A)$.

  In all possible cases, $m \in V((< \alpha >< \beta > A \wedge < \alpha;\beta > A) \vee ([\alpha][\beta]\neg A \wedge [\alpha;\beta]\neg A))$. Thus $m \in V(< \alpha >< \beta > A \leftrightarrow < \alpha;\beta > A)$.

- Let m be a node of the model $\mathcal{M}$ then:

  – Either $\exists m'$ such that $(m,m') \in R(\alpha)$ and $m' \in V(A)$, that is $m \in V(< \alpha > A)$ and $m \in V(< \alpha > A \vee < \beta > A)$, and thus $\exists m'$ such that $(m,m') \in R(\alpha \cup \beta)$ and $m' \in V(A)$, that is $m \in V(< \alpha \cup \beta > A)$,

  – or $\exists m'$ such that $(m,m') \in R(\beta)$ and $m' \in V(A)$, that is $m \in V(< \beta > A)$ and $m \in V(< \alpha > A \vee < \beta > A)$, and thus $\exists m'$ such that $(m,m') \in R(\alpha \cup \beta)$ and $m' \in V(A)$, that is $m \in V(< \alpha \cup \beta > A)$,

  – or $\forall m'$, either $(m,m') \notin R(\alpha)$ and $(m,m') \notin R(\beta)$ or $m' \notin V(A)$, that is $m \in V([\alpha]\neg A \wedge [\beta]\neg A)$, and thus $\forall m'$, $(m,m') \notin R(\alpha \cup \beta)$ or $m' \notin V(A)$ that is $m \in V([\alpha \cup \beta]\neg A)$.

  In all possible cases, $m \in V(((< \alpha > A \vee < \beta > A) \wedge < \alpha \cup \beta > A) \vee ([\alpha]\neg A \wedge [\beta]\neg A \wedge [\alpha \cup \beta]\neg A))$. Thus $m \in V(< \alpha \cup \beta > A \leftrightarrow < \alpha > A \vee < \beta > A)$

- Let m be a node of the model $\mathcal{M}$ then:

  – Either $m \in V(A)$ and $m \in V(B)$, that is $\exists m' = m$ such that $(m,m') \in R(A?)$ and $m' \in V(B)$ and thus $m \in V(< A? > B)$, and then $m \in V(A \wedge B)$,

  – or $m \notin V(A)$ or $m \notin V(B)$, that is $\forall m'.(m,m') \notin R(A?)$ or $m' \notin V(B)$ and thus $m \in [A?]\neg B$, and then $m \in V(\neg A \vee \neg B)$

  In all possible cases, $m \in V((< A? > B \wedge A \wedge B) \vee ([A?]\neg B \wedge (\neg A \vee \neg B)))$. Thus $m \in V(< A? > B \leftrightarrow A \wedge B)$

- Let m be a node of the model $\mathcal{M}$ then:

– Either $existsk, m'$ such that $(m, m') \in R(\alpha^k)$ and $m' \in V(A)$, that is $m \in V(< \alpha^* > A)$, and then either $k = 0$ thus $m \in V(A)$ or $k \geq 1$ and $\exists m''$ such that $(m, m'') \in R(\alpha)$ and $(m'', m') \in R(\alpha^{k-1})$ thus $m \in V(< \alpha >< \alpha^* > A)$,

– or $\forall k$, $\forall m'$, $(m, m') \notin R(\alpha^k)$ or $m' \notin V(A)$, that is $m \in V([\alpha^*]\neg A)$. In particular, $m \notin V(A)$ and $\forall m''$, either $(m, m'') \notin R(\alpha)$ or $\forall k'$, $\forall m^{(3)}$, $(m'', m^{(3)}) \notin R(\alpha^k)$ or $m^{(3)} \notin V(A)$, that is $m \in V(\neg A \wedge [\alpha][\alpha^*]\neg A)$.

In all possible cases, $m \in V((< \alpha^* > A \wedge (A \vee < \alpha >< /alpha^* > A) \vee ([\alpha^*]\neg A \wedge \neg A \wedge [\alpha][\alpha^*]\neg A))$. Thus $m \in V(< \alpha^* > A \leftrightarrow A \vee < \alpha >< \alpha^* > A)$

- Let m be a node of the model $\mathcal{M}$ then:

  – Either $\exists m'$ such that $(m, m') \in R(\alpha)$ and $\forall m''$. $((m', m'') \notin R(\alpha^-)$ or $m'' \in V(A))$. As $(m, m') \in R(\alpha)$, $(m', m) \in R(\alpha^-)$ and thus $m \in V(A)$,

  – or $\forall m'$. $(m, m') \notin R(\alpha)$ or $\exists m''$. $((m', m'') \in R(\alpha^-)$ and $m'' \notin V(A))$ that is $m \in V([\alpha] < \alpha^- > \neg A)$.

  In all cases, $m \in V([\alpha] < \alpha^- > \neg A \vee A)$ thus $m \in V(< \alpha > [\alpha^-]A \rightarrow A)$.

- Let m be a node of the model $\mathcal{M}$, by definition of ν_Σ, $(m, \chi(c)) \in R(\nu_\Sigma)$ thus $m \in V(< \nu_\Sigma > c)$

- Let m be a node of the model $\mathcal{M}$ then:

  – either $\chi(c) \notin V(A)$ and thus $m \in V(< \nu_\Sigma > (c \wedge \neg A))$ but then $\forall m''$. $m'' \notin V(c) = \{\chi(c)\}$ or $m'' \notin V(A)$ thus $m \in V([\nu_\Sigma](\neg c \vee \neg A)$. Thus $m \in V([\nu_\Sigma](\neg c \vee \neg A) \wedge < \nu_\Sigma > (c \wedge \neg A))$,

  – or $\chi(c) \in V(A)$ and thus $\forall m', m' \notin V(c)$ or $m' \in V(A)$ thus $m \in V([\nu\_Sigma](\neg c \vee A)$. But then $\exists m'' = \chi(c)$ such that $m'' \in V(c \wedge A)$ and thus $m \in V(< \nu_\Sigma > (c \wedge A)$ thus $m \in V([\nu_\Sigma](\neg c \vee A) \wedge < \nu_\Sigma > (c \wedge A))$.

  In all possible cases, $m \in V(([\nu_\Sigma](\neg c \vee \neg A) \wedge < \nu_\Sigma > (c \wedge \neg A)) \wedge ([\nu_\Sigma](\neg c \vee A) \wedge < \nu_\Sigma > (c \wedge A)))$ that is $m \in V(< \nu_\Sigma > (c \wedge A) \leftrightarrow [\nu_\Sigma](c \rightarrow A))$

- Let m be a node of the model $\mathcal{M}$, S be a subset of Σ and c, d be elements of S. Then:

  – Either $m \in V(c) = \{\chi(d)\}$ and, as $(\chi(c), \chi(d)) \in \chi(S)^2$, $(m, \chi(d)) \in R(\nu_S)$. Moreover as $\chi(d) \in V(d)$, $m \in V(< \nu_S > d)$ and thus $m \in V(\neg c \vee < \nu_S > d)$,

  – or $m \notin V(c)$ and thus $m \in V(\neg c \vee < \nu_S > d)$.

  In all possible cases, $m \in V(\neg c \vee < \nu_S > d)$ that is $m \in V(c \rightarrow < \nu_S > d)$

- Let m be a node of the model $\mathcal{M}$, S be a subset of Σ and c, d be such that $\{c, d\} \not\subseteq S$. Then:
  - Either $c \notin S$ and then:
    - Either $m \in V(c) = \{\chi(c)\}$ which, as $\forall m'\ (\chi(c), m') \notin \chi(S)^2$, means $m \in V([\nu_S]\neg d)$ thus $m \in V(\neg c \vee [\nu_S]\neg d)$,
    - or $m \notin V(c)$ thus $m \in V(\neg c \vee [\nu_S]\neg d)$,
  - or $c \in S$ and $d \notin S$ and then:
    - Either $m \in V(c) = \{\chi(c)\}$ which, as $(\chi(c), \chi(d)) \notin \chi(S)^2$, $\forall m'.(m, m') \notin R(\nu_S)$ or $m' \notin V(d)$ thus $m \in V(\neg c \vee [\nu_S]\neg d)$,
    - or $m \notin V(c)$ and thus $m \in V(\neg c \vee [\nu_S]\neg d)$.

  In all possible cases, $m \in V(\neg c \vee [\nu_S]\neg d)$ that is $m \in V(c \rightarrow [\nu_S]\neg d)$
- Let m be a node of the model $\mathcal{M}$ then:
  - Either $\exists m', m''$ such that $(m, m') \in R(\nu_S)$, $(m', m'') \in R(\nu_S)$ and $m'' \in V(A)$, that is $m \in V(< \nu_S >< \nu_S > A$, and then, as $(m, m'') \in \chi(S)^2$, $(m, m'') \in R(\nu_S)$ and thus $m \in V(< \nu_S > A)$,
  - or $\forall m', m''.\ (m, m') \notin R(\nu_S)$ or $(m', m'') \notin R(\nu_S)$ or $m'' \notin V(A)$. But then, $\forall m'.\ (m, m') \notin R(\nu_S)$ or $\forall m''.\ (m', m'') \notin R(\nu_S)$ or $m'' \notin V(A)$, that is $m \in V([\nu_S][\nu_S]\neg A)$.

  In all possible cases, $m \in V([\nu_S][\nu_S]\neg A \vee < \nu_S > A)$ that is $m \in V(< \nu_S >< \nu_S > A \rightarrow < \nu_S > A)$
- Let m be a node of the model $\mathcal{M}$ then:
  - Either $m \in V(A)$ and then:
    - either $m \in \chi(S)$ and $\forall m'$ such that $(m, m') \in \nu_S$ then $m' \in \chi(S)$ and thus $(m', m) \in R(\nu_S)$ thus $\forall m'.\ (m, m') \notin R(\nu_S)$ or $\exists m'' = m$ such that $(m', m'') \in R(\nu_S)$ and $m'' \in V(A)$, that is $m \in V([\nu_S] < \nu_S > A)$,
    - or $m \notin \chi(S)$ and then $\forall m'.\ (m, m') \notin R(\nu_S)$, that is $m \in V([\nu_S] < \nu_S > A)$
  - or $m \notin V(A)$ and thus $m \in V(\neg A)$.

  In all possible cases, $m \in V(\neg A \vee [\nu_S] < \nu_S > A)$ that is $m \in V(A \rightarrow [\nu_S] < \nu_S > A)$
- Let m be a node of the model $\mathcal{M}$ then:
  - Either $m \in V(A)$ and then $\exists m' = m$ such that $(m, m') \in R(\nu_\Sigma)$ and $m' \in V(A)$ thus $m \in V(< \nu_\Sigma > A)$,
  - or $m \notin V(A)$ and then $m \in V(\neg A)$

In all possible cases, $m \in V(\neg A \vee < \nu_\Sigma > A)$ that is $m \in V(A \rightarrow < \nu_\Sigma > A) < \alpha > A \rightarrow < \nu_\Sigma > A$

- Let m be a node of the model $\mathcal{M}$ then:
  - Either $m \in V(< \alpha > A)$ and then $\exists m'$ such that $(m, m') \in R(\alpha)$ and $m' \in V(A)$ but then $(m, m') \in R(\nu_\Sigma)$ that is $m \in V(< \nu_\Sigma > A)$,
  - or $m \notin V(< \alpha > A)$ and then $m \in V([\alpha]\neg A)$

  In all possible cases, $m \in V([\alpha]\neg A \vee < \nu_\Sigma > A)$ that is $m \in V(< \alpha > A \rightarrow < \nu_\Sigma > A)$

- Let m be a node of the model $\mathcal{M}$, $\phi \in \Phi_0$ then:
  - Either $m \in V(c) = \{\chi(c)\}$ and then as $V(\phi) \subseteq \chi(\Sigma_E)$ and $\chi(\Sigma_E) \cap \chi(\Sigma_O) = \emptyset$, $m \notin V(\phi)$ and thus $m \in V(\neg\phi)$,
  - or $m \notin V(c)$ and thus $m \in V(\neg c)$.

  In all possible cases, $m \in V(\neg c \vee \neg\phi)$ that is $m \in V(c \rightarrow \neg\phi)$

- Let m be a node of the model $\mathcal{M}$, $\alpha \in Pi_0$ then:
  - Either $m \in V(c)$ and then as $R(\alpha) \subseteq \chi(\Sigma_E)^2$ and $\chi(\Sigma_E) \cap \chi(\Sigma_O) = \emptyset$, $\forall m'$, $(m, m') \notin R(\alpha)$ and $(m', m) \notin R(\alpha)$ thus $m \in V([\alpha]\bot \wedge [\alpha^-]\bot)$,
  - or $m \notin V(c)$ and thus $m \in V(\neg c)$.

  In all possible cases, $m \in V(\neg c \vee [\alpha]\bot \wedge [\alpha^-]\bot)$ that is $m \in V(c \rightarrow [\alpha]\bot \wedge [\alpha^-]\bot)$

- Assume $\vdash A$ and $\vdash A \rightarrow B$, then $\forall m$, $m \in V(A)$ and $m \in V(\neg A \vee B)$ thus $m \in V(B)$. That is $\vdash B$.

- Assume $\vdash [\gamma][\alpha^k]A$, for all $k < \omega$, then $\forall m$, $m \in V([\gamma][\alpha^k]A$, for all k, that is $\forall m'$ such that $(m, m') \in R(\gamma)$, $m' \in V([\alpha^k]A)$ for all k. As $\forall m''$, $(m', m'') \in R(\alpha^k)$ for some k or $(m', m'') \notin \bigcup_k R(\alpha^k) = R(\alpha^*)$, that is $(m', m'') \notin R(\alpha^*)$ or $m'' \in V(A)$. Thus $m' \in V([\alpha^*]A)$ and thus $m \in V([\gamma][\alpha^*]A)$. That is $\vdash [\gamma][\alpha^*]A$

(Cov) Assume $\vdash [\gamma]\neg c$, for all $c \in \Sigma$, then $\forall m$, $m \in V([\gamma]\neg c)$. Thus $forall m'$, $(m, m') \notin R(\gamma)$ or $m' \notin V(c)$ for all c. But, as $\chi(\Sigma) = M$, $\exists c'$ such that $m' = \chi(c')$ Thus $\forall m'$, $(m, m') \notin R(\gamma)$. Thus $m \in V([\gamma]\bot)$. That is $\vdash [\gamma]\bot$.

(Nec) Assume $\vdash A$, then $\forall m$, $m \in V(A)$, then $\forall m', m''$, $(m', m'') \notin R(\nu_\Sigma)$ or $m'' \in V(A)$, thus $m' \in V([\nu_\Sigma]A)$. That is $\vdash [\nu_\Sigma]A$

□

Theorem 7.3.3 (Completeness)**.** *If* $\vDash A$ *then* $\vdash A$.

Once again, the proof is drawn from [54].

Definition 7.3.1. *A simple extension of $\mathcal{C}2\mathcal{PDL}$, or logic (over $\mathcal{C}2\mathcal{PDL}$) is any set of $\mathcal{C}2\mathcal{PDL}$ formulae L such that:*

- *L contains all axioms of $\mathcal{C}2\mathcal{PDL}$*
- *L is closed under (MP), (Ind), (Cov) and (Nec).*

Definition 7.3.2. *Let L be a logic, an L-theory is any set $T \subseteq \Phi$ such that:*

- *$L \subseteq T$*
- *T is closed under (MP), (Ind) and (Cov).*

Definition 7.3.3. *A logic L (resp. a theory T) is consistent if $\bot \notin L$ (resp. $\bot \notin T$).*

Definition 7.3.4. *A formula A is said to be closed if $\exists B$ such that $\vdash A \leftrightarrow < \nu_\Sigma > B$.*

Definition 7.3.5. *A logic L (resp. a theory T) is maximal if $\forall A \in \Phi$ such that A is closed, either $A \notin L$ or $A \in L$ (resp. $\forall A \in \Phi$, either $A \notin T$ or $A \in T$).*

Definition 7.3.6. *By $log(\Gamma, A)$ (resp. $th(\Gamma, A)$), we denote the least logic (resp. theory) containing $\Gamma \cup A$.*

Lemma 8 (Deduction lemma for theories)**.** *Let T be an L-theory, $A, B \in \Phi$, then $A \to B \in T$ iff $B \in th(T, A)$.*

Proof. $\Leftarrow$ Assume $B \in th(T, A)$ and let T_0 be $\{D | A \to D \in T\}$. As $A \to A \in T$, $A \in T_0$. Let's prove that T_0 is an L-theory:

- $\forall D \in T$, $\neg A \vee D \in T$ and thus $L \subseteq T \subseteq T_0$.
- Assume $D_0 \in T_0$ and $D_0 \to D_1 \in T_0$. As $A \vee \neg A \in T$, $(A \wedge (D \vee \neg D)) \vee \neg A \in T$ thus $(A \wedge \neg D) \vee \neg A \vee (A \wedge D) \in T$ that is $(A \to D) \to (A \to (A \wedge D)) \in T$. By replacing D with D_0, we obtain $(A \to D_0) \to (A \to (A \wedge D_0)) \in T$ and, as $A \to D_0 \in T$ and T is closed under (MP), $A \to (A \wedge D_0) \in T$. By replacing D with $D_0 \to D_1$, we obtain $(A \to (D_0 \to D_1)) \to (A \to (A \wedge (D_0 \to D_1))) \in T$, $A \to (D_0 \to D_1) \in T$ and, as T is closed under (MP), $A \to (A \wedge (D_0 \to D_1)) \in T$. Similarly, $(A \wedge \neg D_1) \vee \neg A \vee \neg D_1 \in T$ thus $(A \wedge \neg D_1 \wedge (D_0 \vee \neg D_0)) \vee \neg A \vee \neg D_1 \in T$ thus $(A \wedge (\neg A \vee (D_0 \wedge \neg D_1))) \vee (A \wedge (\neg A \vee \neg D_0)) \vee \neg A \vee \neg D_1 \in T$ that is $(A \to (A \wedge (D_0 \to D_1))) \to ((A \to (A \wedge D_0)) \to (A \wedge D_1)) \in T$. Then, as $(A \to (A \wedge (D_0 \to D_1))) \to ((A \to (A \wedge D_0)) \to (A \wedge D_1)) \in T$, $A \to (A \wedge (D_0 \to D_1)) \in T$ and T is closed by (MP), $(A \to (A \wedge D_0)) \to (A \wedge D_1) \in T$. As $(A \to (A \wedge D_0)) \to (A \wedge D_1) \in T$, $(A \to (A \wedge D_0)) \in T$ and T is closed under (MP), $A \wedge D_1 \in T$ thus $D_1 \in T_0$ thus T_0 is stable by (MP).

- Assume $\forall k < \omega, [\gamma][\alpha^k]D \in T_0$, then $\forall k < \omega, A \to [\gamma][\alpha^k]D \in T$ and thus $\forall k < \omega, [A?;\gamma][\alpha^k]D \in T$. But, as T is stable by (Ind), $[A?;\gamma][\alpha^*]D \in T$ and thus $A \to [\gamma][\alpha^*]D \in T$ that is $[\gamma][\alpha^*]D \in T_0$. Thus T_0 is stable by (Ind).
- Assume $\forall c \in \Sigma, [\gamma]\neg c \in T_0$, then $\forall c \in \Sigma, A \to [\gamma]\neg c \in T$ and thus $\forall c \in \Sigma, [A?;\gamma]\neg c \in T$. But, as T is stable by (Cov), $[A?;\gamma]\bot \in T$ and thus $A \to [\gamma]\bot \in T$ that is $[\gamma]\bot \in T_0$. Thus T_0 is stable by (Cov).

Thus T_0 is an L-theory. As $th(T, A)$ is the smallest theory containing $T \cup A$ and T_0 contains $T \cup A$, $th(T, A) \subseteq T_0$ but, as $B \in th(T, A)$ then $B \in T_0$ that is $A \to B \in T$.

$\Rightarrow$ Assume $A \to B \in T$ then, as $th(T, A)$ is stable by (MP) and $A \in th(T, A)$, $B \in th(T, A)$.

□

Lemma 9. *If $\forall B$, ($B \in \Gamma \cup A$ implies $[\nu_\Sigma]B \in th(\Gamma, A)$), then $th(\Gamma, A)$ is a logic.*

Proof. As $th(\Gamma, A)$ is a theory, $L \subseteq th(\Gamma, A)$ and thus all axioms of $\mathcal{C}2\mathcal{PDL}$ are contained in $th(\Gamma, A)$ and $th(\Gamma, A)$ is closed under (MP), (Cov) and (Ind).

Assume $C \in th(\Gamma, A)$. We prove by induction that $[\nu_\Sigma]C \in th(\Gamma, A)$:

- If $C \in Gamma \cup A$, then $[\nu_\Sigma]C \in th(\Gamma, A)$.
- If C results from the application of (MP) on $D \in th(\Gamma, A)$ and $D \to C \in th(\Gamma, A)$, from the induction hypothesis, $[\nu_\Sigma]D \in th(\Gamma, A)$ and $[\nu_\Sigma]D \to C \in th(\Gamma, A)$. But $th(\Gamma, A)$ contains the axiom □ where $A = D$, $B = C$ and $\alpha = \nu_S$, that is $[\nu_\Sigma](D \to C) \to ([\nu_\Sigma]D \to [\nu_\Sigma]C) \in th(\Gamma, A)$. Then, as $th(\Gamma, A)$ is stable under (MP) with $A = [\nu_\Sigma](D \to C)$ and $B = ([\nu_\Sigma]D \to [\nu_\Sigma]C)$, $[\nu_\Sigma]D \to [\nu_\Sigma]C \in th(\Gamma, A)$. Then, as $th(\Gamma, A)$ is stable under (MP) with $A = [\nu_\Sigma]D$ and $B = [\nu_\Sigma]C$, $[\nu_\Sigma]C \in th(\Gamma, A)$.
- If C results from the application of (Ind) on $\forall k < \omega.[\gamma][\alpha^k]D \in th(\Gamma, A)$ then $C = [\gamma][\alpha^*]D$. From the induction hypothesis, $[\nu_\Sigma;\gamma][\alpha^k]D \in th(\Gamma, A)$. Then, as $th(\Gamma, A)$ is stable under (Ind), $[\nu_\Sigma;\gamma][\alpha^*]D \in th(\Gamma, A)$ and thus $[\nu_\Sigma]C \in th(\Gamma, A)$.
- If C results from the application of (Cov) on $\forall c \in \Sigma.[\gamma]\neg c \in th(\Gamma, A)$ then $C = [\gamma]\bot$. From the induction hypothesis, $\forall c \in \Sigma.[\nu_\Sigma;\gamma]\neg c \in th(\Gamma, A)$. Then, as $th(\Gamma, A)$ is stable under (Cov), $[\nu_\Sigma;\gamma]\bot \in th(\Gamma, A)$ and thus $[\nu_\Sigma]C \in th(\Gamma, A)$.

Thus $th(\Gamma, A)$ is stable under (Nec), thus $th(\Gamma, A)$ is a logic. □

Lemma 10. *Let L be a logic and A be a closed formula, $th(L, A) = log(L, A)$*

Proof. $\subseteq$ Let $B \in L \cup A$, if $B \in L$ by the stability of L under (MP), $[\nu_\Sigma]B \in L \subseteq th(L, A)$.

Otherwise, $B = A$ then, as A is closed, $\exists C$ such that $B \leftrightarrow < \nu_\Sigma > C \in L$ and thus $B \rightarrow < \nu_\Sigma > C \in L \subseteq th(L, A)$. Then, as $th(L, A)$ is stable under (MP), $B \in th(L, A)$ and $B \rightarrow < \nu_\Sigma > C \; in th(L, A)$, $< \nu_\Sigma > C \in th(L, A)$. Then, as $th(L, A)$ contains $(\nu_S 4)$ with $S = \Sigma$ and $A = < \nu_\Sigma > C$, it yields $< \nu_\Sigma > C \rightarrow [\nu_\Sigma] < \nu_\Sigma > < \nu_\Sigma > C$ and thus $[\nu_\Sigma] < \nu_\Sigma > < \nu_\Sigma > C \in th(\Gamma, A)$. Then, as $th(L, A)$ contains $(\nu_S 3)$ with $S = \Sigma$ and $A = C$, it yields $< \nu_\Sigma > < \nu_\Sigma > C \rightarrow < \nu_\Sigma > C \in th(\Gamma, A)$. Thus $[\nu_\Sigma] < \nu_\Sigma > C \in th(\Gamma, A)$ and thus $[\nu_\Sigma]B \in th(\Gamma, A)$. Thus, from Lemma 9, $th(L, A)$ is a logic containing $L \cup A$. Thus $log(L, A) \subseteq th(L, A)$.

$\supseteq$ By definition, a logic is a theory and $log(L, A)$ contains $L \cup A$ thus $th(L, A) \subseteq log(L, A)$.

□

Lemma 11 (Deduction lemma for logics)**.** *Let L be a logic and A be a closed formula. Then $A \rightarrow B \in L$ iff $B \in log(L, A)$*

Proof. As L is an L-theory, from Lemma 8, $A \rightarrow B \in L$ iff $B \in th(L, A)$. But, from Lemma 10, $log(L, A) = th(L, A)$ and thus $A \rightarrow B \in L$ iff $B \in log(L, A)$ □

Lemma 12 (Separation lemma for theories)**.** *Let T be a theory, $A \notin T$. Then there exists a maximal theory T^* such that $T \subseteq T^*$ and $A \notin T^*$.*

Proof. Let $T_0 = th(T, \neg A)$. As $A \notin T$, $\neg A \implies \bot \notin T$. Then, from Lemma 8, $\bot \notin T_0$ and thus T_0 is consistent. Let $B_0, B_1, \ldots,$ be an enumeration of Φ. By induction on n, we construct a chain $T_0 \subseteq T_1 \subseteq \ldots$ of consistent theories. Their union will yield the required T^*. The induction hypothesis is that T_n is a consistent theory. It is the case for T_0.

- If $th(T_n, B_n)$ is consistent, then $T_{n+1} = th(T_n, B_n)$ is consistent.
- If $th(T_n, B_n)$ is not consistent, then $\bot \in th(T_n, B_n)$ and, from Lemma 8, $B_n \implies \bot \in T_n$ and thus $\not B_n \in T_n$. Then:
  - Either $B_n \neq [\gamma]0$ and $B_n \neq [\gamma][\alpha^*]A$ and then $T_{n+1} = T_n$ is consistent,
  - or $B_n = [\gamma]0$. Let $B_{n,c} = [\gamma]\neg c$, if $\forall c \in \Sigma$, $B_{n,c} \in T_n$ then, because T_n is stable by (Cov), and $\neg B_n \in T_n$ then T_n is inconsistent which, due to the induction hypothesis, is not the case. Thus $\exists c \in \Sigma$ such that $B_{n,c} \notin T_n$. Then, from Lemma 8, $T_{n+1} = th(T_n, \neg B_{n,c})$ is consistent,
  - or $B_n = [\gamma][\alpha^*]A$. Let $B_{n,k} = [\gamma][\alpha^k]A$, if $\forall k < \omega$, $B_{n,k} \in T_n$ then, because T_n is stable by (Ind), and $\neg B_n \in T_n$ T_n is inconsistent which, due to the induction hypothesis, is not the case. Thus $\exists k < \omega$ such that $B_{n,k} \notin T_n$. Then, from Lemma 8, $T_{n+1} = th(T_n, \neg B_{n,k})$ is consistent.

Let $T^* = \bigcup\{T_n | n < \omega\}$. We have:

- $L \subseteq T \subseteq T_0 \subseteq T^*$
- Let C_0, C_1 be such that $C_0 \in T^*$ and $C_0 \to C_1 \in T^*$, then $\exists k_0, k_1 < \omega$ such that $C_0 \in T_{k_0}$ and $C_0 \to C_1 \in T_{k_1}$ that is $C_0 \in T_{max(k_0,k_1)}$ and $C_0 \to C_1 \in T_{max(k_0,k_1)}$. As $T_{max(k_0,k_1)}$ is closed under (MP), $C_1 \in T_{max(k_0,k_1)}$ and thus $C_1 \in T^*$. Then T^* is closed under (MP).
- As $\forall k$, $\neg A \in T_0 \subseteq T_k$ and T_k is consistent, $A \notin T^*$
- Assume $\bot \in T^*$ then, as $\bot \to A$ is a boolean tautology, both $\bot \in T^*$ and $\bot \to A \in T^*$. As T^* is closed under (MP), then $A \in T^*$. As it is not the case, T^* is consistent.
- By construction, $\forall B \in \Phi$ either $B \in T^*$ or $\neg B \in T^*$
- Let $D_c = [\gamma]\neg c$ and $D = [\gamma]0 = B_n$. Suppose $\forall c \in \Sigma. D_c \in T^*$ and $D \notin T^*$ then, by construction, for some $c_0 \in \Sigma$, $\neg B_{n,c_0} \in T_{n+1} \subseteq T^*$ but then $\neg B_{n,c_0} \in T^*$ and $B_{n,c_0} \in T^*$ which is impossible as T^* is consistent. Thus T^* is closed under (Cov).
- Let $D_k = [\gamma][\alpha^k]A$ and $D = [\gamma][\alpha^*]A = B_n$. Suppose $\forall k < \omega. D_k \in T^*$ and $D \notin T^*$ then, by construction, for some $k_0 \in \Sigma$, $\neg B_{n,k_0} \in T_{n+1} \subseteq T^*$ but then $\neg B_{n,k_0} \in T^*$ and $B_{n,k_0} \in T^*$ which is impossible as T^* is consistent. Thus T^* is closed under (Ind).

Thus T^* is a maximal theory and $T \subseteq T^*$ and $A \notin T^*$. □

Definition 7.3.7. $\mathcal{L}_T = \{A | [\nu_\Sigma]A \in T\}$.

Lemma 13. *If T is a maximal L-theory, then $\mathcal{L}_T$ is a maximal logic and $\mathcal{L}_T$ is the greatest logic included in T.*

Proof.
- Let $\mathcal{A}$ be an axiom of $\mathcal{C}2\mathcal{P}\mathcal{D}\mathcal{L}$ then $\mathcal{A} \in L$ as L is a logic. As L is closed under (MP), $[\nu_\Sigma]\mathcal{A} \in L$. As $L \subseteq T$, $[\nu_\Sigma]\mathcal{A} \in T$ and thus $\mathcal{A} \in \mathcal{L}_T$
- Assume $C \in \mathcal{L}_T$ and $C \to D \in \mathcal{L}_T$ then $[\nu_\Sigma]C \in T$ and $[\nu_\Sigma](C \to D) \in T$. As $L \subseteq T$ and L contains $(\Box)$ and T is closed under (MP), $[\nu_\Sigma]C \to [\nu_\Sigma]D \in T$ and then, by (MP), $[\nu_\Sigma]D \in T$ and thus $D \in \mathcal{L}_T$. Thus $\mathcal{L}_T$ is closed under (MP).
- Assume $\forall k < \omega.[\gamma][\alpha^k]A \in \mathcal{L}_T$, then $\forall k < \omega.[\nu_\Sigma; \gamma][\alpha^k]A \in T$. As T is closed under (Ind), $[\nu_\Sigma; \gamma][\alpha^*]A \in T$ and thus $[\gamma][\alpha^k]A \in \mathcal{L}_T$. Thus $\mathcal{L}_T$ is closed under (Ind).
- Assume $\forall c \in \Sigma.[\gamma]\neg c \in \mathcal{L}_T$, then $\forall c \in \Sigma.[\nu_\Sigma; \gamma]\neg c \in T$. As T is closed under (Cov), $[\nu_\Sigma; \gamma]\bot \in T$ and thus $[\gamma]\bot \in \mathcal{L}_T$. Thus $\mathcal{L}_T$ is closed under (Cov).

- Assume $C \in \mathcal{L}_T$ then $[\nu_\Sigma]C \in T$. Assume $< \nu_\Sigma > [\nu_\Sigma]C \in T$ then as $L \subseteq T$ and L contains $(\nu_S 3)$ with $S = \Sigma$ and $A = \neg C$, $[\nu_\Sigma]C \rightarrow [\nu_\Sigma][\nu_\Sigma]C \in T$. As T is closed under (MP), $[\nu_\Sigma][\nu_\Sigma]C \in T$ and thus $[\nu_\Sigma]C \in T$. Thus $\mathcal{L}_T$ is closed under (Nec).

- Let C be a closed formula. As T is maximal:

  - either $C \in T$ that is $existsB. < \nu_\Sigma > B \in T$. As $L \subseteq T$ and L contains $(\nu_S 4)$ for $S = \Sigma$ and $A =< \nu_\Sigma > B$, $< \nu_\Sigma > B \rightarrow [\nu_\Sigma] < \nu_\Sigma >< \nu_\Sigma > B \in T$ and then, by (MP), $[\nu_\Sigma] < \nu_\Sigma >< \nu_\Sigma > B \in T$. As $L \subseteq T$ and L contains $(\nu_S 3)$ for $S = \Sigma$ and $A = B$ and L is closed under (Nec), $[\nu_\Sigma] < \nu_\Sigma >< \nu_\Sigma > B \rightarrow [\nu_\Sigma] < \nu_\Sigma > B \in T$ and then, by (MP), $[\nu_\Sigma] < \nu_\Sigma > B \in T$ and thus $< \nu_\Sigma > B \in T$ that is $C \in \mathcal{L}_T$,
  - or $\neg C \in T$ that is $\exists B.[\nu_\Sigma]\neg B \in T$ and thus $\neg B \in \mathcal{L}_T$. But, as $\mathcal{L}_T$ is closed under (Nec), $[\nu_\Sigma]\neg B \in \mathcal{L}_T$ and thus $\neg C \in \mathcal{L}_T$.

  That is $\mathcal{L}_T$ is maximal.

- Let L' be a logic such that $L' \subseteq T$. Let $A' \in L'$. As L' is closed under (Nec), $[\nu_\Sigma]A' \in L'$ and thus $[\nu_\Sigma]A' \in T$ that is $A' \in \mathcal{L}_T$. Thus $\forall L' \subseteq T, L' \subseteq \mathcal{L}_T$.

- Let $C \in \mathcal{L}_T$ then $[\nu_\Sigma]C \in T$. As $L \subseteq T$ and L contains $(\nu_\Sigma 1)$ with $A = \neg C$, $\neg C \rightarrow < \nu_\Sigma > \neg C \in T$ or, written in another way, $[\nu_\Sigma]C \rightarrow C \in T$. As T is closed under (MP), $C \in T$. Thus $\mathcal{L}_T \subseteq T$.

Thus $\mathcal{L}_T$ is a maximal logic and it is the greatest included in T. □

Lemma 14 (Separation lemma for logics)**.** *Let L be a logic, $A \notin L$. Then there exists a maximal logic L^* such that $L \subseteq L^*$ and $A \notin L^*$.*

Proof. L is an L-theory thus, from Lemma 12, there exists a maximal theory T^* such that $L \subseteq T^*$ and $A \notin T^*$. Then, from Lemma 13, $\mathcal{L}_{T^*}$ is a maximal logic such that $\mathcal{L}_{T^*} \subseteq T^*$. Assume $A \in \mathcal{L}_{T^*}$ then $A \in T^*$ which is not the case. Thus $\mathcal{L}_{T^*}$ is a maximal L-logic, that is $L \subseteq \mathcal{L}_{T^*}$, and $A \notin \mathcal{L}_{T^*}$. □

Lemma 15 (Lindenbaum lemma)**.** *If L is a consistent logic (resp. T is a consistent theory) then there exists a maximal consistent logic L^* (resp. a maximal consistent theory T^*) such that $L \subseteq L^*$ (resp. $T \subseteq T^*$).*

Proof. It is a direct consequence of the Separation lemmata with $A = \bot$. □

Lemma 16. *If L is a consistent logic, then L has a model.*

Proof. From Lemma 15, there exists a maximal consistent logic L^* such that $L \subseteq L^*$. Let's define $c\ d =< \nu_\Sigma > (c \wedge d) \in L^*$.

- As L contains $(\Sigma 1)$, so does L^* and thus $< \nu_\Sigma > (c \wedge c) \in L^*$. Thus $c\ c$ that is is reflexive.

- Assume $c\ d$. As $\wedge$ is commutative, $< \nu_\Sigma > (c \wedge d) \leftrightarrow < \nu_\Sigma > (d \wedge c) \in L^*$ and thus, by (MP), $d\ c$ that is is symmetric.

- Assume $c\ d$ and $d\ e$. Then, as L^* contains $(\Sigma 2)$ and is closed under (MP), $[\nu_\Sigma](d \rightarrow e) \in L^*$ and $[\nu_\Sigma](c \rightarrow d) \in L^*$. Then, as $[\nu_\Sigma](c \rightarrow d) \rightarrow ([\nu_\Sigma](d \rightarrow e) \rightarrow ([\nu_\Sigma](c \rightarrow d) \wedge [\nu_\Sigma](d \rightarrow e)))$ is a boolean tautology and thus in L^*, by applying (MP) twice, $[\nu_\Sigma](c \rightarrow d) \wedge [\nu_\Sigma](d \rightarrow e) \in L^*$. But, as $([\nu_\Sigma](c \rightarrow d) \wedge [\nu_\Sigma](d \rightarrow e)) \rightarrow [\nu_\Sigma]((c \rightarrow d) \wedge (d \rightarrow e))$ is a boolean tautology and thus in L^*, by applying (MP), $[\nu_\Sigma]((c \rightarrow d) \wedge (d \rightarrow e)) \in L^*$ and thus $[\nu_\Sigma](c \rightarrow e) \in L^*$. Then, from $(\Sigma 2)$ using (MP), $< \nu_\Sigma > (c \wedge e) \in L^*$ and thus $c\ e$ that is is transitive.

Thus is an equivalence relation. Let $[c] = \{d | c\ d\}$, we construct $M = \Sigma_{/}$, $\chi(c) = [c]$, $V(A) = \{[c] | < \nu_\Sigma > (c \wedge A) \in L^*\}$ and $R^*(\alpha) = \{([c], [d]) | < \nu_\Sigma > (c \wedge < \alpha > d) \in L^*\}$. Let's show that $\mathcal{M} = (M, \chi, V, R)$ is a model.

- χ is obviously onto.

- Assume $m \in \chi(\Sigma_O) \cap \chi(\Sigma_E)$ that is $\exists c_O \in \Sigma_O, c_E \in \Sigma_E$ such that $[c_O] = [c_E]$ that is $< \nu_\Sigma > c_O \wedge c_E \in L^*$. As $\Sigma_0 \cup \Sigma_E = \emptyset$, $c_0 \notin \Sigma_E$ and thus, from $(\Sigma_S 2)$, $c_0 \rightarrow [\nu_{\Sigma_E}] \neg c_E$. Meanwhile, from $(\Sigma_S 1)$, $c_E \rightarrow < \nu_\Sigma > c_E$. Thus $< \nu_\Sigma > (< \nu_{\Sigma_E} > c_E \wedge [\nu_\Sigma] \neg c_E) \in L^*$ which is false. As L^* is consistent, $\chi(\Sigma_O) \cap \chi(\Sigma_E) = \emptyset$.

- Let $i \in \Sigma$, then $V(i) = \{[c] | < \nu_\Sigma > (c \wedge i) \in L^*\} = \{[i]\} = \{\chi(i)\}$.

- Let $\phi_0 \in \Phi_0$, assume $\exists c_O \in \Sigma_O$ such that $[c_O] \in V(\phi_0)$ then $< \nu_\Sigma > (c_O \wedge \phi_0) \in Ł^*$. As $\Sigma_O 1$ is in L^*, $c_O \rightarrow \neg \phi_0$ and thus $< \nu_\Sigma > (\neg \phi_0 \wedge \phi_0) \in Ł^*$ which is impossible as L^* is consistent. Thus $V(\phi_0) \in \mathcal{P}(\chi(\Sigma_E))$.

-
  - Assume $[c] \in M \backslash V(A)$ then $< \nu_\Sigma > (c \wedge A) \notin L^*$. As L^* is maximal, $[\nu_\Sigma](c \rightarrow \neg A) \in L^*$. As $(\Sigma 2)$ is in L^* so is $< \nu_\Sigma > (c \wedge \neg A) \notin L^*$ and thus $[c] \in V(\neg A)$. Thus $M \backslash V(A) \subseteq V(\neg A)$.
  - Otherwise $[c] \in V(A)$ then $< \nu_\Sigma > (c \wedge A) \in L^*$. If $[c] \in V(\neg A)$, then $< \nu_\Sigma > (c \wedge \neg A) \in L^*$. As $(\Sigma 2)$ is in L^* so is $[\nu_\Sigma](c \rightarrow \neg A) \in L^*$ and thus $< \nu_\Sigma > (A \wedge \neg A) \in L^*$ as L^* is consistent, it is impossible and thus $V(\neg A) \subseteq M \backslash V(A)$.

  Thus $V(\neg A) = M \backslash V(A)$.

-
  - Assume $[c] \in V(A) \cup V(B)$ then:
    - Either $[c] \in V(A)$ and then $< \nu_\Sigma > (c \wedge A) \in L^*$ and thus $< \nu_\Sigma > (c \wedge (A \vee B)) \in L^*$. Thus $[c] \in V(A \vee B)$,
    - or $[c] \in V(B)$ and then $< \nu_\Sigma > (c \wedge B) \in L^*$ and thus $< \nu_\Sigma > (c \wedge (A \vee B)) \in L^*$. Thus $[c] \in V(A \vee B)$.

  Thus $V(A) \cup V(B) \subseteq V(A \vee B)$

– Otherwise $[c] \notin V(A) \cup V(B)$ that is $[c] \in M \backslash (V(A) \cup V(B)) = (M \backslash V(A)) \cap (M \backslash V(B))$. Thus $[c] \in M \backslash V(A)$ that is $[c] \in V(\neg A)$ from the previous point. Thus $< \nu_\Sigma > (c \wedge \neg A) \in L^*$. Similarly, $[c] \in M \backslash V(A)$ and thus $< \nu_\Sigma > (c \wedge \neg B) \in L^*$. From $(\Sigma 2)$, $[\nu_\Sigma](c \rightarrow \neg B) \in L^*$ and thus $< \nu_\Sigma > (c \wedge \neg(A \vee B)) \in L^*$. Thus $[c] \in V(\neg(A \vee B))$ that is $[c] \in M \backslash V(A \vee B)$ and thus $[c] \notin V(A \vee B)$. Thus $V(A \vee B) \subseteq V(A) \cup V(B)$

Thus $V(A \vee B) = V(A) \vee V(B)$.

- – Assume $[c] \in \{s | \exists [d] \in M.((s,[d]) \in R(\alpha) \wedge [d] \in V(A)\}$ then $([c],[d]) \in R(\alpha)$ and $[d] \in V(A)$. Thus $< \nu_\Sigma > (c \wedge < \alpha > d) \in L^*$ and $< \nu_\Sigma > (d \wedge A) \in L^*$. As $(\Sigma 2)$ in L^*, $[\nu_\Sigma](d \rightarrow A) \in L^*$ and thus $< \nu_\Sigma > (c \wedge < \alpha > A) \in L^*$ that is $[c] \in V(< \alpha > A)$. Thus $\{s | \exists [d] \in M.((s,[d]) \in R(\alpha) \wedge [d] \in V(A)\} \subseteq V(< \alpha > A)$.
  – Otherwise $[c] \notin \{s | \exists [d] \in M.((s,[d]) \in R(\alpha) \wedge [d] \in V(A)\}$ then $\forall [d]$, $(([s],[d]) \notin R(\alpha)$ or $[d] \notin V(A))$ that is $\forall [d]$, $(< \nu_\Sigma > (c \wedge < \alpha > d) \notin L^*$ or $< \nu_\Sigma > (d \wedge A) \notin L^*)$. Then, by maximality of L^*, $\forall [d]$, $(\neg < \nu_\Sigma > (c \wedge < \alpha > d) \in L^*$ or $\neg < \nu_\Sigma > (d \wedge A) \in L^*$ that is $\forall [d]$, $([\nu_\Sigma](c \rightarrow [\alpha]\neg d) \in L^*$ or $[\nu_\Sigma](d \rightarrow \neg A) \in L^*$. Then $\forall [d]$, $([\nu_\Sigma](c \rightarrow [\alpha](\neg d \vee \neg A)) \in L^*$ or, by $(\nu_\Sigma 2)$ and $(\nu_S 3)$, $[\nu_\Sigma; c?; \alpha](d \rightarrow \neg A)) \in L^*$ that is $[\nu_\Sigma]c \rightarrow [\alpha](\neg c \vee (d \rightarrow \neg A)) \in L^*$. Thus $\forall [d]$, $[\nu_\Sigma; c?; \alpha; A?]\neg d \in L^*$. As L^* is stable under (Cov), $[\nu_\Sigma; c?; \alpha; A?]\bot \in L^*$ that is $[\nu_\Sigma](c \rightarrow [\alpha]\neg A) \in L^*$. By $(\Sigma 2)$, $< \nu_\Sigma > (c \wedge [\alpha]\neg A) \in L^*$ and thus $[c] \in V([\alpha]\neg A)$ that is $[c] \notin V(< \alpha > A)$. Thus $V(< \alpha > A) \subseteq \{s | \exists [d] \in M.((s,[d]) \in R(\alpha) \wedge [d] \in V(A)\}$.

  Thus $V(< \alpha > A) = \{s | \exists [d] \in M.((s,[d]) \in R(\alpha) \wedge [d] \in V(A)\}$.

- Let $\alpha_0 \in \Pi_0$, let $c_O \in \Sigma_O$, $c \in \Sigma$:
  – such that $([c_O],[c]) \in R(\alpha)$ then $< \nu_\Sigma > (c_O \wedge < \alpha_0 > c) \in L^*$. But, from $(\Sigma_O 2)$, $< \nu_\Sigma > (c_O \wedge [\alpha_0]\bot) \in L^*$ and thus, from $(\Sigma 2)$, $[\nu_\Sigma](c_O \rightarrow [\alpha_0]\bot) \in L^*$. Thus $< \nu_\Sigma > ([\alpha_0]\neg c \wedge < \alpha_0 > c) \in L^*$ which is impossible as L^* is consistent.
  – such that $([c],[c_O]) \in R(\alpha)$ then $< \nu_\Sigma > (c \wedge < \alpha_0 > c) \in L^*$. But, from $(\Sigma_O 2)$, $< \nu_\Sigma > (c_O \wedge [\alpha_0^-]\bot) \in L^*$ and thus, from $(\Sigma 2)$, $[\nu_\Sigma](c_O \rightarrow [\alpha_0^-]\bot) \in L^*$. Thus $< \nu_\Sigma > (c \wedge < \alpha_0 > [\alpha_0^-]\bot) \in L^*$. But then, from $(-)$, $< \nu_\Sigma > (c \wedge \bot) \in L^*$ which is impossible as L^* is consistent.

  Thus $\forall c_0 \in \Sigma_O, \forall c \in \Sigma$, $([c_0],[c]) \notin R(\alpha_0)$ and $([c],[c_0]) \notin R(\alpha_0)$. Thus $R(\alpha_0) \in \mathcal{P}(\chi(\Sigma_E))$.

- Let $S \subseteq \Sigma$:
  – Let $c_0, c_1 \in S$, from $(\Sigma_S 1)$, $< \nu_\Sigma > (c_0 \wedge < \nu_S > c_1) \in L^*$ and thus $([c_0],[c_1]) \in R(\nu_S)$. Thus $\chi(S)^2 \subseteq R(\nu_S)$.

– Otherwise, $\{c_0, c_1\} \not\subseteq S$ and then, from $(\Sigma_S 2)$, $< \nu_\Sigma > (c_0 \wedge [\nu_S]\neg c_1) \in L^*$. Assume $([c_0], [c_1]) \in R(\nu_S)$ then $< \nu_\Sigma > (c_0 \wedge < \nu_S > c_1) \in L^*$ and, from $(\Sigma 2)$, $[\nu_\Sigma](c_0 \rightarrow < \nu_S > c_1) \in L^*$ and thus $< \nu_\Sigma > (< \nu_S > c_1 \wedge [\nu_S]\neg c_1) \in L^*$ which is impossible as L^* is consistent. Thus $R(\nu_S) \subseteq \chi(S)^2$.

Thus $\chi(S)^2 = R(\nu_S)$

- – Let $([c], [d]) \in R(\alpha) \cup R(\beta)$ then:
    * either $([c], [d]) \in R(\alpha)$ and thus $< \nu_\Sigma > (c \wedge < \alpha > d) \in L^*$ and then, from $(\cup)$; $< \nu_\Sigma > (c \wedge < \alpha \cup \beta > d) \in L^*$ and thus $([c], [d]) \in R(\alpha \cup \beta)$,
    * or $([c], [d]) \in R(\beta)$ and thus $< \nu_\Sigma > (c \wedge < \beta > d) \in L^*$ and then, from $(\cup)$; $< \nu_\Sigma > (c \wedge < \alpha \cup \beta > d) \in L^*$ and thus $([c], [d]) \in R(\alpha \cup \beta)$

  Thus $R(\alpha) \cup R(\beta) \subseteq R(\alpha \cup \beta)$.
  – Let $([c], [d]) \in R(\alpha \cup \beta)$ then $< \nu_\Sigma > (c \wedge < \alpha \cup \beta > d) \in L^*$. Then, from $(\cup)$, $< \nu_\Sigma > (c \wedge (< \alpha > d \vee < \beta > d)) \in L^*$. Then:
    * either $< \nu_\Sigma > (c \wedge < \alpha > d) \in L^*$ and then $([c], [d]) \in R(\alpha)$,
    * or $< \nu_\Sigma > (c \wedge < \beta > d) \in L^*$ and then $([c], [d]) \in R(\beta)$,
    * or $< \nu_\Sigma > (c \wedge [\alpha]\neg d \wedge [\beta]\neg d) \in L^*$ and thus, from $(\Sigma 2$, $[\nu_\Sigma](c \rightarrow ([\alpha]\neg d \wedge [\beta]\neg d)) \in L^*$. Then $< \nu_\Sigma > ((< \alpha > d \vee < \beta > d) \wedge [\alpha]\neg d \wedge [\beta]\neg d) \in L^*$ which is impossible as L^* is consistent.

  Thus $R(\alpha \cup \beta) \subseteq R(\alpha) \cup R(\beta)$

Thus $R(\alpha \cup \beta) = R(\alpha) \cup R(\beta)$

- – Let $([c], [d]) \in \{(s,t) | \exists [e].((s, [e]) \in R(\alpha)$ and $([e], t) \in R(\beta))\}$ then $< \nu_\Sigma > (c \wedge < \alpha > e) \in L^*$ and $< \nu_\Sigma > (e \wedge < \beta > d) \in L^*$ that is, from $(\Sigma 2)$, $[\nu_\Sigma](e \rightarrow < \beta > d) \in L^*$. Then $< \nu_\Sigma > (c \wedge < \alpha >< \beta > d) \in L^*$. Then, from $(;)$, $< \nu_\Sigma > (c \wedge < \alpha; \beta > d) \in L^*$ and then $([c], [d]) \in R(\alpha; \beta)$. Thus $\{(s,t) | \exists [e].((s, [e]) \in R(\alpha)$ and $([e], t) \in R(\beta))\} \subseteq R(\alpha; \beta)$.
  – Let $([c], [d]) \notin \{(s,t) | \exists [e].((s, [e]) \in R(\alpha)$ and $([e], t) \in R(\beta))\}$ that is $\forall e.$ $(([c], [e]) \notin R(\alpha)$ or $([e], [d]) \notin R(\beta))$. Assume $< \nu_\Sigma > (c \wedge < \alpha > e) \in L^*$ and $< \nu_\Sigma > (e \wedge < \beta > d) \in L^*$ then $([c], [e]) \in R(\alpha)$ and $([e], [d]) \in R(\beta)$ which is not the case. Thus $\forall e.$ $(< \nu_\Sigma > (c \wedge < \alpha > e) \notin L^*$ or $< \nu_\Sigma > (e \wedge < \beta > d) \notin L^*)$. As L^* is maximal, $\forall e.$ $([\nu_\Sigma](\neg c \vee [\alpha]\neg e) \in L^*$ or $[\nu_\Sigma](\neg e \vee [\beta]\neg d) \in L^*)$. Thus $\forall e.$ $([\nu_\Sigma](c \rightarrow [\alpha](< \beta > d \rightarrow \neg e) \in L^*$ or, from $(\nu_\Sigma 2)$, $[\nu_\Sigma; c?; \alpha](< \beta > d \rightarrow \neg e) \in Then^*)$. Thus $\forall e.$ $[\nu_\Sigma; c?; \alpha; (< \beta > d)?]\neg e) \in L^*$. As L^* is closed under (Cov), $[\nu_\Sigma; c?; \alpha; (< \beta > d)?]\bot \in L^*$ that is, from $(?)$ and $(\Sigma 2)$, $[\nu_\Sigma](c \rightarrow [\alpha](< \beta > d \rightarrow \bot)) \in L^*$. Assume $([c], [d]) \in R(\alpha; \beta)$ then $< \nu_\Sigma > (c \wedge < \alpha >< \beta > d) \in L^*$. Thus $< \nu_\Sigma > ([\alpha](< \beta > d \rightarrow \bot) \wedge < \alpha >< \beta > d) \in L^*$ which is impossible

as L^* is consistent. Thus $R(\alpha;\beta) \subseteq \{(s,t)|\exists[e].((s,[e]) \in R(\alpha)$ and $([e],t) \in R(\beta))\}$.

Thus $R(\alpha;\beta) = \{(s,t)|\exists[e].((s,[e]) \in R(\alpha)$ and $([e],t) \in R(\beta))\}$.

- – Let $([c],[d]) \in \{(s,t)|(t,s) \in R(\alpha)\}$ then $< \nu_\Sigma > (d \wedge < \alpha > c) \in L^*$. From $(\Sigma 1)$, $< \nu_\Sigma > c \in L^*$. Assume $< \nu_\Sigma > (c \wedge [\alpha^-]\neg d) \in L^*$ then, from $(\Sigma 2)$, $[\nu_\Sigma](c \rightarrow [\alpha^-]\neg d) \in L^*$ and thus $< \nu_\Sigma > (d \wedge < \alpha > [\alpha^-]\neg d) \in L^*$. But, from $(-)$, $< \nu_\Sigma > (d \wedge \neg d) \in L^*$ which is impossible as L^* is consistent. Thus, as L^* is maximal, $[\nu_\Sigma](\neg c \vee < \alpha^- > d) \in L^*$ that is $[\nu_\Sigma](c \rightarrow < \alpha^- > d) \in L^*$ and thus $([c],[d]) \in R(\alpha^-)$. Thus $\{(s,t)|(t,s) \in R(\alpha)\} \subseteq R(\alpha^-)$.
  - Let $([c],[d]) \notin \{(s,t)|(t,s) \in R(\alpha)\}$. If $< \nu_\Sigma > (d \wedge < \alpha > c) \in L^*$, $([c],[d]) \in \{(s,t)|(t,s) \in R(\alpha)\}$ which is not the case thus, by maximality of L^*, $[\nu_\Sigma](\neg d \vee [\alpha]\neg c) \in L^*$. Assume $([c],[d]) \in R(\alpha^-)$, then $< \nu_\Sigma > (c \wedge < \alpha^- > d) \in L^*$ that is, using $(\Sigma 2)$, $[\nu_\Sigma](c \rightarrow < \alpha^- > d) \in L^*$ and thus $[\nu_\Sigma](d \rightarrow [\alpha] < \alpha^- > d) \in L^*$. Then, from $(-)$, $[\nu_\Sigma](d \rightarrow \neg d) \in L^*$ that is, from $(\Sigma 2)$, $< \nu_\Sigma > (d \wedge \neg d) \in L^*$ which is impossible as L^* is consistent. Thus $R(\alpha^-) \subseteq \{(s,t)|(t,s) \in R(\alpha)\}$.

  Thus $R(\alpha^-) = \{(s,t)|(t,s) \in R(\alpha)\}$.

- – Let $([c],[d]) \in R(\alpha^*)$. Then, $< \nu_\Sigma > (c \wedge < \alpha^* > d) \in L^*$. Assume $\forall k.\ < \nu_\Sigma > (c \wedge < \alpha^k > d) \notin L^*$ then, as L^* is maximal, $\forall k.\ [\nu_\Sigma](\neg c \wedge [\alpha^k]\neg d) \in L^*$ that is $[\nu_\Sigma; c?;][\alpha^k]\neg d \in L^*$. But L^* is closed under (Ind) and thus $[\nu_\Sigma; c?;][\alpha^*]\neg d \in L^*$ that is $[\nu_\Sigma](c \rightarrow [\alpha^*]\neg d) \in L^*$ and thus $< \nu_\Sigma > ([\alpha^*]\neg d \wedge < \alpha^* > d) \in L^*$ which is impossible as L^* is consistent. Thus $R(\alpha^*) \subseteq \bigcup_{k<\omega} R(\alpha^k)$
  - Let's prove by induction that $< \alpha^k > A \rightarrow < \alpha^* > A \in L^*$:
    - From $(*)$, $A \rightarrow < \alpha^* > A \in L^*$ thus $< \alpha^0 > A \rightarrow < \alpha^* > A \in L^*$
    - Assume $< \alpha^k > A \rightarrow < \alpha^* > A \in L^*$, then $[\alpha^{k+1}]\neg A \vee < \alpha^{k+1} > A \in L^*$ being a tautology, thus $[\alpha^{k+1}]\neg A \vee < \alpha > < \alpha^k > A \in L^*$ and then, from the induction hypothesis, $[\alpha^{k+1}]\neg A \vee < \alpha > < \alpha^* > A \in L^*$. But, from $(*)$, $< \alpha > < \alpha^* > A \rightarrow < \alpha^* > A \in L^*$ and thus $< \alpha^{k+1} > A \rightarrow < \alpha^* > A \in L^*$.

  Then, assume $([c],[d]) \in \bigcup_{k<\omega} R(\alpha^k)$. There exists k such that $([c],[d]) \in R(\alpha^k)$ and thus $< \nu_\Sigma > (c \wedge < \alpha^k > d) \in L^*$ but then, as $< \alpha^k > A \rightarrow < \alpha^* > A \in L^*$, $< \nu_\Sigma > (c \wedge < \alpha^* > d) \in L^*$ and thus $([c],[d]) \in R(\alpha^*)$. Thus $\bigcup_{k<\omega} R(\alpha^k) \subseteq R(\alpha^*)$.

  Thus $R(\alpha^*) = \bigcup_{k<\omega} R(\alpha^k)$.

- – Let $([c],[d]) \in R(A?)$ then $< \nu_\Sigma > (c \wedge < A? > d) \in L^*$. From $(?)$, $< \nu_\Sigma > (c \wedge A) \in L^*$ and $< \nu_\Sigma > (c \wedge d) \in L^*$. Thus $[c] = [d]$ and $[c] \in V(A)$ thus $([c],[d]) \in \{(s,s)|s \in V(A)\}$. Thus $R(A?) \subseteq \{(s,s)|s \in V(A)\}$

– Let $([c],[d]) \in \{(s,s)|s \in V(A)\}$ then $[c] = [d]$ and $[c] \in V(A)$ thus $< \nu_\Sigma > (c \wedge A) \in L^*$ and $< \nu_\Sigma > (c \wedge d) \in L^*$ thus $< \nu_\Sigma > (c \wedge < A? > d) \in L^*$ and thus $([c],[d]) \in R(A?)$. Thus $\{(s,s)|s \in V(A)\} \subseteq R(A?)$.

Thus $R(A?) \subseteq \{(s,s)|s \in V(A)\}$.

Thus $\mathcal{M}$ is a model.

Let $A \in L$ then, as L is closed under (Nec), $[\nu_\Sigma]A \in L$ and as L contains $(\Sigma 1)$, $< \nu_\Sigma > c \in L$ thus $< \nu_\Sigma > (c \wedge A) \in L \subseteq L^*$. Thus $\mathcal{M}, [c] \vDash A$ which is the case for each $c \in \Sigma$. Thus $\mathcal{M} \vDash A$. Thus $\mathcal{M}$ is a model of L. □

We can now prove the theorem itself:

Proof. Assume $\nvdash A$ then $A \notin L$ and also $[\nu_\Sigma]A \notin L$ thus $log(L, < \nu_\Sigma > \neg A)$ is consistent and thus, from Lemma 16, has a model $\mathcal{M}$. Thus $\mathcal{M} \vDash < \nu_\Sigma > \neg A$ i.e. $\mathcal{M} \nvDash A$ and thus $\nvdash A$. □

Theorem 7.3.4 (Decidability)**.** *The validity and satisfiability problems of $\mathcal{C}2\mathcal{PDL}$ are decidable.*

The idea is similar to the one for $\mathcal{CPDL}$, that is we prove that the ω-rules, (Ind) and (Cov), can be replaced so that the set of valid formulae is recursively enumerable. Then, we prove that if a formula is satisfiable then it is satisfied by a finite model. In this case, there is a procedure, that may not stop, deciding if the formula is valid and another one, that may not stop either, deciding if the formula is unsatisfiable. As the formula can't be both, one of them will reach a result eventually.

Definition 7.3.8. *Let $\mathcal{FC}2\mathcal{PDL}$ be the logic obtained from $\mathcal{C}2\mathcal{PDL}$ by dropping the rules (Ind) and (Cov) and adding the axiom (ind): $(A \wedge [\alpha^*](A \rightarrow [\alpha]A)) \rightarrow [\alpha^*]A$. Let $\vdash_F$ denote provability in $\mathcal{FC}2\mathcal{PDL}$.*

The theorems of $\mathcal{FC}2\mathcal{PDL}$ form a recursively enumerable set.

Lemma 17. *If $\vdash_F A$ then $\vdash A$*

Proof. It amounts to prove that $\vdash ind$. Let $\gamma = (A \wedge [\alpha^*](A \rightarrow [\alpha]A))?$. Assume there exists $k < \omega$ such that $< \gamma >< \alpha^k > \neg A$ that is $A \wedge [\alpha^*](A \rightarrow [\alpha]A) \wedge < \alpha^k > \neg A$. From $(*)$, applied k times, one obtains $A \wedge \bigwedge_{0 \leq l < k} [\alpha^l](A \rightarrow [\alpha]A) \wedge [\alpha^k](A \rightarrow [\alpha]A \wedge [\alpha][\alpha^*](A \rightarrow [\alpha]A)) \wedge < \alpha^k > \neg A$. Then, $A \wedge \bigwedge_{0 \leq l < k} [\alpha^l](A \rightarrow [\alpha]A)$ yields $\bigwedge_{0 \leq l \leq k} [\alpha^l]A$ and thus, adding $< \alpha^k > \neg A$ an impossibility is reached.

Thus $\forall k < \omega$, $[\gamma][\alpha^k]A$ which, from (Ind), yields $[\gamma][\alpha^*]A$ that is $(A \wedge [\alpha^*](A \rightarrow [\alpha]A)) \rightarrow [\alpha^*]A$. Thus (ind) is a theorem of $\mathcal{C}2\mathcal{PDL}$. □

Definition 7.3.9. *The Fischer-Ladner closure of a set of formulae Σ is the smallest set $\mathcal{FL}$ that satisfies:*

- $\Sigma \subseteq \mathcal{FL}$
- *$\mathcal{FL}$ is closed under sub-formulae*

- *If* $[\alpha \cup \beta]A \in \mathcal{FL}$, $[\alpha]A \in \mathcal{FL}$ *and* $[\beta]A \in \mathcal{FL}$
- *If* $[\alpha;\beta]A \in \mathcal{FL}$, $[\alpha][\beta]A \in \mathcal{FL}$
- *If* $[\alpha^*]A \in \mathcal{FL}$, $[\alpha][\alpha^*]A \in \mathcal{FL}$
- *If* $[\alpha^-]A \in \mathcal{FL}$, $[\alpha]\neg[\alpha^-]A \in \mathcal{FL}$

Lemma 18. *The Fischer-Ladner closure of a finite set is finite.*

Definition 7.3.10. *We name canonical quasi-model the model* $\mathcal{M}_c = (M_c, R_c, V_c)$ *where:*

- M_c *is the set of all maximal consistent sets of formulae*
- *for every program* α *and for all* $u, v \in M_c$, $u\ R_c(\alpha)\ v$ *iff, for every formula* A, *if* $[\alpha]A \in u$ *then* $A \in v$
- *for every atomic proposition* ϕ, $V_c(\phi) = \{u \in M_c | \phi \in u\}$
- *for every name* i, $V_c(i) = \{u \in M_c | i \in u\}$

$\mathcal{M}_c$ is named a quasi-model because it is not a model. It is the template of the model we will create to prove the correctness though.

Lemma 19. *For all* $u \in M_c$ *and all formulae* A, $\mathcal{M}_c, u \vdash A$ *iff* $A \in u$.

Proof. This is done by induction on the complexity of A.

- If A is an atomic proposition or a name, this is true by construction.
- If $A = \neg B$, by induction, $\mathcal{M}_c, u \nvdash B$ iff $B \notin u$. As u is maximal, $B \notin u$ iff $A \in u$ and thus $A \in u$ iff $\mathcal{M}_c, u \vdash A$.
- If $A = B \vee C$, $\mathcal{M}_c, u \vdash A$ iff either $\mathcal{M}_c, u \vdash B$ or $\mathcal{M}_c, u \vdash C$ iff , by induction, either $B \in u$ or $C \in u$ iff, as u is maximal, $B \vee C \in u$.
- If $A = [\alpha]B$,
  - Assume $A \in u$ then for all v such that $u\ R_c(\alpha)\ v$, $B \in v$ by construction. By induction, $\mathcal{M}_c, v \vdash B$ and thus $\mathcal{M}_c, u \vdash A$
  - Assume $\mathcal{M}_c, u \vdash A$, then if v is such that $u\ R_c(\alpha)\ v$, $\mathcal{M}_c, v \vdash B$. By induction, $B \in v$. Assume $A \notin u$, A does not introduce additional constraints and thus $u \cup A$ is consistent. This is impossible as u is maximal. Thus $A \in u$.

□

Lemma 20. *A is a theorem iff A is true in* $\mathcal{M}_c$.

Proof. If $\vDash A$, every maximal consistent set contains A and thus, from lemmaLemma 19, $\mathcal{M}_c, u \vdash A$ for all $u \in M_c$. Thus A is true in $\mathcal{M}_c$. Otherwise, $\{\neg A\}$ is consistent and, from Lemma 14, can thus be extended to a maximal consistent set x. As $x \in M_c$ and $A \notin x$, $\mathcal{M}_c, x \nvdash A$. □

Lemma 21. $\forall \alpha, \beta,\ R_c(\alpha \cup \beta) = R_c(\alpha) \cup R_c(\beta)$.

Proof. Assume $uR_c(\alpha \cup \beta)v$ and $[\alpha \cup \beta]A \in u$. Then, as u is maximal $[\alpha]A \in u$ and $[\beta]A \in u$. Thus $R_c(\alpha) \cup R_c(\beta) \subseteq R_c(\alpha \cup \beta)$.

Assume neither $uR_c(\alpha)v$ nor $uR_c(\beta)v$. Then, there exists A and B such that $[\alpha]A \in u$, $A \notin v$, $[\beta]B \in u$ and $B \notin v$. Then $[\alpha](A \vee B) \in u$ and $[\beta](A \vee B) \in u$ and thus, by maximality of u, $[\alpha \cup \beta](A \vee B) \in u$. But $A \vee B \notin v$ thus it is impossible that $uR_c(\alpha \cup \beta)v$ and thus $R_c(\alpha \cup \beta) \subseteq R_c(\alpha) \cup R_c(\beta)$. □

Lemma 22. $\forall \alpha, \beta,\ R_c(\alpha; \beta) = \{(x, y) | \exists z.(x, z) \in R_c(\alpha) \wedge (z, y) \in R_c(\beta)\}$.

Proof. Assume $uR_c(\alpha; \beta)v$ and $[\alpha; \beta]A \in u$. Then, as u is maximal $[\alpha][\beta]A \in u$. Thus $\{(x, y) | \exists z.(x, z) \in R_c(\alpha) \wedge (z, y) \in R_c(\beta)\} \subseteq R_c(\alpha; \beta)$.

Assume $uR_c(\alpha; \beta)v$. Let C_i be the formulae in v. We define a new set of formulae such that $B_0 = C_0$ and $B_i = B_{i+1} \wedge C_i$. We consider the set $\Delta = \{A : [\alpha]A \in u\} \cup \{\neg[\beta]\neg B_n : n < \omega\}$. Suppose Δ is inconsistent. Then there are $A_0, \ldots, A_n \in \{A : [\alpha]A \in u\}$ and $i_0, \ldots, i_m$ such that $\{A_0, \ldots, A_m, \neg[\beta]\neg B_{i_0}, \ldots, \neg[\beta]\neg B_{i_m}\}$ is an inconsistent set. Let $k = max(i_0, \ldots, i_m)$, then $\{A_0, \ldots, A_m, \neg[\beta]\neg B_k\}$ is inconsistent. Thus $\vDash A_0 \wedge \cdots \wedge A_m \Rightarrow [\beta]\neg B_k$ and thus $\vDash [\alpha]A_0 \wedge \cdots \wedge [\alpha]A_m \Rightarrow [\alpha][\beta]\neg B_k$. Then $[\alpha][\beta]\neg B_k \in u$ and thus $[\alpha; \beta]\neg B_k \in u$ and thus $\neg B_k \in v$. As v is consistent, $B_k \notin v$ which is contrary to the definition of B_k. Thus Δ is consistent. Thus, from Lemma 15, $\exists x$ such that $Delta \subseteq x$. Then, by definition of $\mathcal{M}_c$, $uR_c(\alpha)x$ and $xR_c(\alpha)y$. Thus $R_c(\alpha; \beta) \subseteq \{(x, y) | \exists z.(x, z) \in R_c(\alpha) \wedge (z, y) \in R_c(\beta)\}$. □

Lemma 23. $\forall \alpha, \beta,\ R_c(\alpha^-) = \{(x, y) | (y, x) \in R_c(\alpha)\}$.

Proof. Assume $uR_c(\alpha^-)v$. Pick A such that $[\alpha]A \in v$. It is then impossible that $[\alpha^-]\neg[\alpha]A \in u$ thus $\neg[\alpha^-]\neg[\alpha]A \in u$. Hence $A \in u$. Therefore $vR_c(\alpha)u$ and thus $\{(x, y) | (y, x) \in R_c(\alpha)\} \subseteq R_c(\alpha^-)$.

Assume $vR_c(u)u$. Pick A such that $[\alpha^-]A \in u$. It is then impossible that $\neg[\alpha^-]\neg[\alpha]A \in v$. Hence $A \in v$. Therefore $uR_c(\alpha^-)v$ and thus $R_c(\alpha^-) \subseteq \{(x, y) | (y, x) \in R_c(\alpha)\}$. □

Definition 7.3.11. *Let $\mathcal{M} = (M, R, \chi, V)$ be a model and let Γ be any set of formulae closed under sub-formulae. We define the equivalence relation $\sim_\Gamma$ on M by:*
$s \sim_\Gamma t$ iff $\forall \phi \in \Gamma$, $(\mathcal{M}, s \vDash \phi$ iff $\mathcal{M}, t \vDash \phi)$.

We note $[s]_\Gamma$ the equivalence class of s with respect to $\sim_\Gamma$. The structure $\mathcal{M}_\Gamma = (M_\Gamma, R_\Gamma, \chi_\Gamma, V_\Gamma)$ is called filtration of $\mathcal{M}_c$ with respect to Γ if:

- $M_\Gamma := \{[s]_\Gamma | s \in M_c\}$

- *for every program* $\alpha \in \Gamma$*, if* $sR_c(\alpha)t$*, then* $[s]_\Gamma R_\Gamma(\alpha)[t]_\Gamma$
- *for every program* $\alpha \in \Gamma$*, if* $[s]_\Gamma R_\Gamma(\alpha)[t]_\Gamma$*, then all formulae* A*,* $[\alpha]A \in s \cap \Gamma$ *only if* $A \in t$
- *for every name in* $o \in \Gamma$*, if* $o \in s$*,* $[s]_\Gamma \in \chi_\Gamma(o)$
- *for every atomic proposition* $\phi_0 \in \Gamma$*,* $V_\Gamma(\phi_0) = \{[s]_\Gamma | s \in V_c(\phi_0)\}$

Let's prove that χ_Γ is a function. Assume $o \in \Gamma$, $[s]_\Gamma$ and $[t]_\Gamma$ such that $[s]_\Gamma \in \chi_\Gamma(o)$ and $[t]_\Gamma \in \chi_\Gamma(o)$. Either $s \sim_\Gamma t$ and then $[s]_\Gamma = [t]_\Gamma$ or there is ϕ such that $\mathcal{M}, s \vDash \phi$ and $\mathcal{M}, t \nvDash \phi$. But then, s being maximal, $< \nu_{M_\Gamma} > (o \sqcap \phi) \in s$ and $< \nu_{M_\Gamma} > (o \sqcap \neg\phi) \in s$. This is impossible as s is consistent. Thus $\#(\chi_\Gamma(o)) \leq 1$. Moreover, $< \nu_{M_\Gamma} > o$ is consistent and thus $\#(\chi_\Gamma(o)) = 1$. All nodes not named with names occurring in Γ can be unnamed and renamed so that χ_Γ is onto.

Lemma 24. *Let* $\mathcal{M}_\Gamma$ *be the filtration of* $\mathcal{M}_c$ *with respect to a* Γ*. Then for each formula* $A \in \Gamma$ *and all* $s \in M_c$*,* $\mathcal{M}_c, s \vDash A$ *iff* $\mathcal{M}_\Gamma, [s]_\Gamma \vDash A$*.*

Proof. The proof is by induction on A.

- For atomic propositions and names, this is by construction.
- If $A = \phi \wedge \psi$, by the induction hypothesis, $\mathcal{M}_c, s \vDash \phi$ iff $\mathcal{M}_\Gamma, [s]_\Gamma \vDash \phi$ and $\mathcal{M}_c, s \vDash \psi$ iff $\mathcal{M}_\Gamma, [s]_\Gamma \vDash \psi$ and thus $\mathcal{M}_c, s \vDash A$ iff $\mathcal{M}_\Gamma, [s]_\Gamma \vDash A$.
- If $A = \neg\phi$, by the induction hypothesis, $\mathcal{M}, s \vDash \phi$ iff $\mathcal{M}_\Gamma, [s]_\Gamma \vDash \phi$ and thus $\mathcal{M}, s \vDash A$ iff $\mathcal{M}_\Gamma, [s]_\Gamma \vDash A$.
- If $A = [\alpha]B$, then:
  - Assume $\mathcal{M}_c, s \vDash A$ then, from Lemma 19, $A \in s$ and thus, by construction, $B \in t$ for all t such that $sR_c(\alpha)t$. Then, by induction, $\mathcal{M}_\Gamma, [t]_\Gamma \vDash B$. Thus, for all $[t]_\Gamma$ such that $[s]_\Gamma\ R_\Gamma(\alpha)[t]_\Gamma$, $\mathcal{M}_\Gamma, [t]_\Gamma \vDash B$ thus $\mathcal{M}_\Gamma, [s]_\Gamma \vDash A$
  - Assume $\mathcal{M}_\Gamma, [s] \vDash A$. Then for all $t \in M_c$ with $sR_c(\alpha)t$, $\mathcal{M}_\Gamma, [t] \vDash B$ and thus, by induction, $\mathcal{M}_c, t \vDash B$. Thus $\mathcal{M}_c, s \vDash A$.

□

Lemma 25. *If* Γ *is such that* $|\Gamma| = n$*, that is finite,* $|\mathcal{M}_\Gamma| \leq 2^n$*.*

Proof. There are at most 2^n equivalence classes for n formulae. □

Definition 7.3.12. *Let* Γ *be a set of formulae closed under sub-formulae. Let* $\mathcal{M}^\dagger = \{M_\Gamma, R^\dagger, V_\Gamma, \chi_\Gamma\}$ *be a model such that, for all atomic programs*$\pi \in \Psi$*,* $[u]_\Gamma R^\dagger [v]_\Gamma$ *iff* $\exists u_0 \sim_\Gamma u \exists v_0 \sim_\Gamma v.(u_0 R(\pi) v_0)$*.*

There can actually be a lot of them as there are no conditions on programs $\pi \notin \Gamma$.

Lemma 26. $\forall \alpha \in \Gamma$, *if* $uR_c(\alpha)v$, *then* $[u]_\Gamma R^\dagger(\alpha)[v]_\Gamma$.

Proof. This is done by induction on α.

- If $\alpha \in \Pi_0$, this true by construction.
- If $\alpha = \beta \cup \delta$. Assume $uR_c(\alpha)v$ then, by Lemma 21, $uR_c(\beta)v$ or $uR_c(\delta)v$ so, by induction, $[u]_\Gamma R^\dagger(\beta)[v]_\Gamma$ or $[u]_\Gamma R^\dagger(\delta)[v]_\Gamma$. In either case, $[u]_\Gamma R^\dagger(\alpha)[v]_\Gamma$.
- If $\alpha = \beta;\delta$. Assume $uR_c(\alpha)v$ then, by Lemma 22, there exists x such that $uR_c(\beta)x$ and $xR_c(\delta)v$ so, by induction, $[u]_\Gamma R^\dagger(\beta)[x]_\Gamma$ and $[x]_\Gamma R^\dagger(\delta)[v]_\Gamma$ that is $[u]_\Gamma R^\dagger(\alpha)[v]_\Gamma$.
- If $\alpha = \beta^-$. Assume $uR_c(\alpha)v$ then, by Lemma 23, $vR_c(\beta)u$ so, by induction, $[v]_\Gamma R^\dagger(\beta)[u]_\Gamma$ that is $[u]_\Gamma R^\dagger(\alpha)[v]_\Gamma$.
- If $\alpha = \beta^*$. Assume $uR_c(\alpha)v$ and not $[u]_\Gamma R^\dagger(\alpha)[v]_\Gamma$. Since $\mathcal{M}^\dagger$ is a model with a finite universe, there exists $B = o_0 \vee \cdots \vee o_n$ such that $\forall w$, $B \in w$ iff $[u]_\Gamma R^\dagger(\alpha)[w]_\Gamma$. In particular, $B \in u$. Moreover, as $B \notin v$, $[\alpha]B \notin u$, then, from (*ind*), $[\beta^*](B \to [\beta]B) \notin u$. Thus, there exists x, y such that $uR_c(\beta^*)x$, $B \in x$, $xR_c(\beta)y$ and $B \notin y$. Then, $[u]_\Gamma R^\dagger(\beta^*)[x]_\Gamma$ and, by induction, $[x]_\Gamma R^\dagger(\beta)[y]_\Gamma$ and thus $[u]_\Gamma R^\dagger(\beta^*)[y]_\Gamma$. Thus $B \in y$ which is not possible.

□

Lemma 27. $\forall \alpha \in \Gamma$, *if* $[u]_\Gamma R^\dagger(\alpha)[v]_\Gamma$, *then* $\forall A \in \Gamma, [\alpha]A \in u \cap \Gamma$ *only if* $A \in v$.

Proof. This is done by induction on α.

- Assume $\alpha = \beta \cup \delta$ and $[u]_\Gamma R^\dagger(\alpha)[v]_\Gamma$. Take any A such that $[\beta \cup \delta]A \in u \cap \Gamma$. As Γ is closed under Fisher-Ladner conditions, $[\beta]A \in u \cap \Gamma$ and $[\delta]A \in u \cap \Gamma$. As $\mathcal{M}^\dagger$ is a model, either $[u]_\Gamma R^\dagger(\beta)[v]_\Gamma$ or $[u]_c R^\dagger(\delta)[v]_\Gamma$. By induction, in either case, $A \in v$.
- Assume $\alpha = \beta;\delta$ and $[u]_\Gamma R^\dagger(\alpha)[v]_\Gamma$. Take any A such that $[\beta;\delta]A \in u \cap \Gamma$. Then, as Γ is closed under Fischer-Ladner conditions, $[\beta][\delta]A \in \Gamma$. Since $\mathcal{M}^\dagger$ is a model, there exists x such that $[u]_\Gamma R^\dagger(\beta)[x]_\Gamma$ and $[x]_c R^\dagger(\delta)[v]_\Gamma$. By induction, $[\delta]A \in x$ and thus $A \in v$.
- Assume $\alpha = \beta^-$ and $[u]_\Gamma R^\dagger(\alpha)[v]_\Gamma$. Take any A such that $[\beta^-]A \in u \cap \Gamma$. As Γ is closed under Fischer-Ladner conditions, $[\beta]\neg[\beta^-]A \in \Gamma$. Assume $[\beta]\neg[\beta^-]A \in v \cup \Gamma$. As $\mathcal{M}^\dagger$ is a model, $[v]_\Gamma R^\dagger(\beta)[u]_\Gamma$ and thus, by induction, $\neg[\beta^-]A \in u$ which is impossible. Thus $< \beta > [\beta^-]A \in v$ and thus $A \in v$.
- Assume $\alpha = \beta^*$ and $[u]_\Gamma R^\dagger(\alpha)[v]_\Gamma$. Take any A such that $[\beta^*]A \in u \cap \Gamma$. We prove that $\forall x, y, \forall i$, if $[x]_\Gamma (R^\dagger(\beta))^i [y]_\Gamma$ then $[\beta^*]A \in x$ only if $[\beta^*]A \in y$ by induction. The case $i = 0$ is trivial. Suppose the claim holds for n, $[x]_\Gamma (R^\dagger(\beta))^{n+1}[y]_\Gamma$ and $[\beta^*]A \in x$. Then, as Γ is closed under

Fischer-Ladner conditions, $[\beta][\beta^*]A \in x$. But then there is z such that $[x]_\Gamma R^\dagger(\beta)[z]_\Gamma$ and $[z]_\Gamma(R^\dagger(\beta))^n[y]_\Gamma$. By induction on the first hypothesis, $[\beta^*]A \in z$ and, by induction on the second hypothesis, $[\beta^*]A \in y$. As $\mathcal{M}^\dagger$ is a model, $[u]_\Gamma R^\dagger(\beta^*)[v]_\Gamma$ implies that there is j such that $[u]_\Gamma(R^\dagger(\beta))^j[v]_\Gamma$ and thus $[\beta^*]A \in v$ and thus $A \in v$.

□

Lemma 28. *Let Γ be a finite set closed under Fischer-Ladner conditions. Then $\mathcal{M}^\dagger$ is a filtration of $\mathcal{M}_c$ under Γ.*

Proof.

- $M_\Gamma := \{[s]_\Gamma | s \in M_c\}$ by construction
- for every program $\alpha \in \Gamma$, if $sR_c(\alpha)t$, then $[s]_\Gamma R_\Gamma(\alpha)[t]_\Gamma$ by Lemma 26.
- for every program $\alpha \in \Gamma$, if $[s]_\Gamma R_\Gamma(\alpha)[t]_\Gamma$, then all formulae A, $[\alpha]A \in s \cap \Gamma$ only if $A \in t$ by Lemma 27
- for every name in $o \in \Gamma$, if $o \in s$, $[s]_\Gamma \in \chi_\Gamma(o)$ by construction
- for every atomic proposition $\phi_0 \in \Gamma$, $V_\Gamma(\phi_0) = \{[s]_\Gamma | s \in V_c(\phi_0)\}$ by construction

□

Lemma 29. *If $\nvdash_F A$ then, for some finite model $\mathcal{M}$, $\mathcal{M} \nVdash A$.*

Proof. Assume $\nvdash_F A$ then, from Lemma 20, there exists x such that $A \notin x$. We call Γ the Fischer-Ladner closure of $\{A\}$. It is finite. We define accordingly $\mathcal{M}^\dagger$. By Lemma 28, $\mathcal{M}^\dagger$ is a filtration and, by Lemma 24, $\mathcal{M}^\dagger, [x]_\Gamma \nVdash A$ and $\mathcal{M}^\dagger$ is a model. □

$\mathcal{C}2\mathcal{PDL}$ can be compared to other logics and it is possible to translate its formulae into formulae of another logic. This can be used to prove decidability of the logic in another way. For instance, Hybrid μ-Calculus[61] is known to be decidable and can express most $\mathcal{C}2\mathcal{PDL}$ formulae.

Hybrid μ-Calculus cannot express the separation between nodes in Σ_1 and Σ_2, that is between nodes that currently exists and those that may be created in the future. It is possible to use the same method used in Chapter 4 to avoid creating and deleting nodes by having all nodes pre-exist and use a labeling indicating those that currently do not exist and adding to the pre and postcondition that nodes that do not exist do not satisfy atomic propositions and do not have incoming or outgoing edges.

Hybrid μ-Calculus contains almost the same constructors as $\mathcal{C}2\mathcal{PDL}$. It lacks the program closure but it allows for the μ and ν constructors. As $\langle\alpha^*\rangle\phi$ is equivalent to $\mu x.(\phi \vee \langle\alpha\rangle x)$, it is possible to rewrite the formulae of $\mathcal{C}2\mathcal{PDL}$ into formulae of Hybrid μ-calculus.

7.4 Checking the applicability condition

The formula $App(tag(\rho))$ expresses the applicability of the rule $tag(\rho)$. In other words, it expresses the existence of a match of the left-hand side of $tag(\rho)$. More precisely $App(tag(\rho))$ is defined as follows:

$App(tag(\rho)) = \phi_{nodes} \wedge \phi_{edges}$
where $\phi_{nodes} = \bigwedge_{u \in tag(N_{lhs_\rho})} \langle \nu_{\Sigma_1} \rangle (u \wedge \bigwedge_{\phi \in L_N^{tag(lhs_\rho)}(\theta(u))} \phi)$ and
$\phi_{edges} = \bigwedge_{e \in E | s(e) = \theta(u)} \langle \nu_{\Sigma_1} \rangle (u \wedge \langle L_E^{tag(lhs_\rho)}(e) \rangle tag(t(e)) \wedge$
$\bigwedge_{e \in E | t(e) = \theta(u)} \langle \nu_{\Sigma_1} \rangle (u \wedge \langle L_E^{tag(lhs_\rho)}(e)^- \rangle tag(s(e))$.

ϕ_{nodes} states that the first condition of the match is satisfied i.e., all formulae satisfied by a node of the left-hand side have to be satisfied by its image in the graph. ϕ_{edges} does the same with edges: the image of an edge is labeled the same as the edge, the image of its source is the source of its image and the image of its target is the target of its image.

Tagging a rule may seem to reduce its applicability. Indeed, by choosing a new name for each node of the left-hand side, the rule can now be applied only at the nodes of the graph named accordingly. Let ρ be a rule such that $LHS_\rho = (N_\rho = \{i_0, \dots i_n\}, E_\rho, \mathcal{C}_\rho, \mathcal{R}_\rho, L_{N_\rho}, L_{E_\rho}, s_\rho, t_\rho)$ is its left-hand side. In order to prove that the application of ρ on a graph $G = (N_G, E_G, \mathcal{C}_G, \mathcal{R}_G, L_{N_G}, L_{E_G}, s_G, t_G)$ is correct, one has to verify that for every match $h = (h_N, h_E)$, the postcondition is satisfied after the transformation associated with ρ is applied at the $h_N(i_k)$'s. Instead of showing that, the verification proves that for any graph, if the rule can be applied at the $\theta(u)$'s, where θ is the function that associates to each name of Σ a node of G and $u \in tag(\rho)$, the postcondition is satisfied after performing the transformation. As the u's are fresh names, they do not have any impact on the previous characterization of the graphs. Thus, the validity of $wp(\rho, Post)$ actually states that whatever the choice of θ, if the rule can be applied, then the postcondition will be satisfied after it is applied.

Theorem 7.4.1. $G \models \mathrm{App}(tag(\rho)) \Leftrightarrow$ *there exists a match* h *between* lhs_ρ *and* G.

Informally, the predicate *NApp(s)*, used in the definition of the verification conditions function *vc*, says that the strategy s cannot be applied. To express *NApp(s)* one may wonder whether one can use the negation of *App*(s). The answer is negative since *App* has been defined using dedicated names, that is to say a rule is applied at a specific place defined by the added names introduced by the function *tag*. Intuitively, if one wants to express that there is no match for a rule whose left-hand side contains a cycle (say it is the second graph of Figure 7.2), one needs to use universal quantification. For instance, in that case it would be $\forall i, j.[\nu_{\Sigma 1}](i \Rightarrow [R](j \Rightarrow [R]\neg i))$. One of them can be discarded to produce the expression $\forall i.[\nu_{\Sigma 1}](i \Rightarrow [R][R]\neg i)$. Alas, this cannot be expressed in $\mathcal{C}2\mathcal{PDL}$. The names only allow to express existential quantifiers. Indeed, to express universal quantifiers, one needs to extend the logic with the binder $\downarrow$ of hybrid logic which is enough to make the logic undecidable[4].

To express formally *NApp(s)* in $\mathcal{C}2\mathcal{PDL}$, one needs to introduce some additional definitions first.

Definition 7.4.1 (Explicitly Named Nodes, Non-Oriented Cycles)**.** *An* explicitly named node *is a node such that each disjunct of the disjunctive normal form of its label contains a name. A* non-oriented cycle*, c, is a finite list of nodes* $c = [n_0, \ldots, n_k]$ *such that (i)* $n_k = n_0$ *and (ii)* $\forall \kappa \in [0, k-1], \exists r \in \Pi_0$ *such that* $(n_\kappa, n_{\kappa+1}) \in R(r)$ *or* $(n_{\kappa+1}, n_\kappa) \in R(r)$*.*

Definition 7.4.2 (Grove and thicket)**.** *A* grove *is a disjoint union of thickets. A* thicket *is a connected graph such that it does not contain any non-oriented cycle composed only of non explicitely named nodes. We call a strategy* $\mathcal{S}$ *relative to a rewriting system* $\mathcal{R}$ *a* grove-strategy *if the left-hand sides of all the rules appearing under a closure are groves.*

Let *lhs* be a left-hand side. Let us split N_{lhs} into T_E, the set of explicitly named nodes, and $T_I = N_{lhs} \backslash T_E$. For each maximally connected subgraph composed only of nodes in T_I, a distinguished node r_i is selected. If there is a maximally connected subgraph composed only of explicitly named nodes, an r_i is also picked for it. Now, everything is ready to define *NApp(s)*.

1 $NApp(\rho) = \bigvee_{r_i} [\nu_{\Sigma_1}] NA(r_i, \emptyset)$

2 $NA(n, V) = (\bigvee_{\phi \in L_N(n)} \neg\phi) \vee (\bigvee_{e \in E | s(e) = n | s(e) \in T_E \cup (T_I \backslash V)} [L_E(e)] NA(t(e), V \cup \{n\})) \vee$
$(\bigvee_{e \in E | t(e) = n | t(e) \in T_E \cup (T_I \backslash V)} [L_E(e)^-] NA(s(e), V \cup \{n\}))$ if $n \notin V$

3 $NA(n, V) = \neg\mu(n)$ if $n \in T_E \cap V$

4 $NApp(\epsilon) = \top$

5 $NApp(s_0 \oplus s_1) = NApp(s_0) \wedge NApp(s_1)$

6 $NApp(s^*) = \bot$

7 $NApp(s_0; s_1) = NApp(s_0)$

Rule 2 is the most involved. $NA(n, V)$ is used to describe what as to be true for a node not to be n. V is used to track which nodes have already been visited. $\bigvee_{\phi \in L_N(n)} \neg\phi$ states that there is at least one of the formulae satisfied by n that is not satisfied while $\bigvee_{e \in E | s(e) = n | s(e) \in T_E \cup (T_I \backslash V)} [L_E(e)] NA(t(e), V \cup \{n\})$ means that for there is a neighbor of n using an outgoing edge that cannot find a match. $\bigvee_{e \in E | t(e) = n | t(e) \in T_E \cup (T_I \backslash V)} [L_E(e)^-] NA(s(e), V \cup \{n\})$ has the same signification but for incoming edges. Rule 3 is used to recall the names of the explicitly named nodes that have already been visited without creating a cycle in the execution. Rules 4 to 7 are exactly the same as for $NApp(s, G)$. Rule 1 says that for at least one r_i, it is not possible to find a node that would be a match. It is noteworthy that there is no rule for $N \in T_I \cap V$. This is because no such would need it as the left-hand side has to be grove.

Figure 7.2: These two graphs are indistinguishable without names

Theorem 7.4.2. *$G \models \mathrm{NApp} \leftrightarrow$ there does not exist a match h between lhs_ρ and G.*

Example 7.4.1. *The rules of the sudoku example are really simple, having only one-node left-hand side, and thus not very interesting as far as* NApp *is concerned. We will thus consider the rule, say ρ, of Figure 8.2. There is only one connected subgraph so one has to pick one distinguished node r_0. Let us choose, randomly, the node j as r_0. Then $\mathrm{NApp}(\rho) = [\nu_{\Sigma_1}]\mathrm{NA}(j, \emptyset)$. As j is not explicitly named, $\mathrm{NA}(j, \emptyset) = \neg B \vee [\beta]\mathrm{NA}(k, \{j\}) \vee [\alpha^-]\mathrm{NA}(i, \{j\})$. As i is not explicitly named either, $\mathrm{NA}(i, \{j\}) = \neg A$ as the only neighbor of i is j. Finally, $\mathrm{NA}(k, \{j\}) = [\beta^*]\neg C$ as the only neighbor of k is j. Thus $\mathrm{NApp}(\rho) = [\nu_{\Sigma_1}](\neg B \vee [\beta][\beta^*]\neg C \vee [\alpha^-]\neg A)$.*

7.5 Application to the hospital

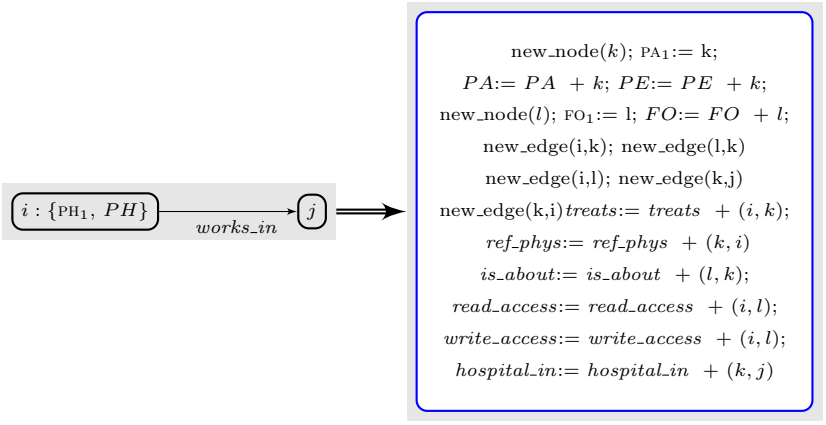
We now illustrate how a graph rewriting system is proven correct in $\mathcal{C}2\mathcal{PDL}$. In this example, illustrated in Figure 7.3, the second transformation that creates a new patient. It also creates a folder to go with it and assigned to a physician. For the sake of simplicity none of the precondition, postcondition and invariant feature all the properties that are required in Section 3.3. The chosen precondition is $(\langle\nu_{\Sigma_2}\rangle.\mathrm{PA}_1) \wedge (\langle\nu_{\Sigma_2}\rangle\ \mathrm{FO}_1) \wedge (\langle\nu_{\Sigma_1}\rangle(\ \mathrm{PH}_1 \wedge PH)) \wedge ([\nu_{\Sigma_1}](PA \vee\ MS) \Rightarrow PE)$ that can be read as PH_1 is an existing physician, PA_1 and FO_1 do not currently exist and all patients and medical staffers are persones. The chosen postcondition is $(\langle\nu_{\Sigma_1}\rangle.(\mathrm{PA}_1 \wedge \langle \mathit{is\_about}^-\rangle\ \mathrm{FO}_1 \wedge \langle\ \mathit{ref\_phys}\rangle(\ \mathrm{PH}_1 \wedge PH)) \wedge ([\nu_{\Sigma_1}](PA \vee\ MS) \Rightarrow PE)$ that means that FO_1 now exists and that it is about PA_1 that currently exists too and whose referent physicien is PH_1. In addition, patients and medical staffers are still persons.

Let us now check that the specification (Pre,s,$Post$) is correct. The correctness formula is $(Pre \Rightarrow wp(s, Post)) \sqcap vc(s, Post)$. As s is only one rule, $wp(s, Post) = App(tag(\rho)) \Rightarrow wp(tag(\alpha_\rho, Post)$ and $vc(s, Post) = \top$. Thus the correctness formula is $Pre \sqcap App(tag(\rho)) \Rightarrow wp(tag(\alpha_\rho, Post))$. As we only use one rule, tag is the identity. Thus $App(tag(\rho)) = \langle\nu_{\Sigma_1}\rangle(i \wedge PH \wedge \mathrm{PH}_1 \wedge \langle \mathit{works\_in}\rangle j$.

Let us now focus on $wp(tag(\alpha_\rho, Post))]$ that is $Post[\alpha_\rho]$. We will now look at the actions performed and how $Post$ is modified.

- *hospital_in* never occurs in $Post$ and thus $Post$ is left unaffected by $[\mathit{hospital\_in} := \mathit{hospital\_in} + (k, j)]$

Pre: $(\langle\nu_{\Sigma_2}\rangle \text{PA}_1) \wedge (\langle\nu_{\Sigma_2}\rangle\ \text{FO}_1) \wedge (\langle\nu_{\Sigma_1}\rangle(\ \text{PH}_1 \wedge PH)) \wedge ([\nu_{\Sigma_1}](PA \vee\ MS) \Rightarrow PE)$



Post: $(\langle\nu_{\Sigma_1}\rangle.(\text{PA}_1 \wedge \langle is\_about^-\rangle\ \text{FO}_1 \wedge \langle\ ref\_phys\rangle(\ \text{PH}_1 \wedge PH))$

$\wedge([\nu_{\Sigma_1}](PA \vee\ MS) \Rightarrow PE)$

Figure 7.3: An example using the second transformation

- *write_access* never occurs in *Post* and thus *Post* is left unaffected by $[write\_access{:=}\ write\_access{+}(i, l)]$
- *read_access* never occurs in *Post* and thus *Post* is left unaffected by $[read\_access{:=}\ read\_access{+}(i, l)]$
- *is_about* occurs in $\langle is\_about^-\rangle\ \text{FO}_1$. It is replaced by $\langle\ is\_about^- \cup (k?); \nu_{\Sigma_1}; (l?)\rangle\ \text{FO}_1$
- *ref_phys* occurs in $\langle ref\_phys\rangle(\text{PH}_1 \wedge\ PH)$. It is replaced by $\langle\ ref\_phys \cup k?; \nu_{\Sigma_1}; i?\rangle(\text{PH}_1 \wedge\ PH)$
- *treats* never occurs in *Post* and thus *Post* is left unaffected by $[treats{:=}\ treats{+}(i, k)]$
- None of the new_edge substitutions affects *Post*
- *FO* never occurs in *Post* and thus *Post* is left unaffected by $[FO{:=}\ FO{+}l]$
- FO_1 occurs in $\langle(k?); is\_about^-; (l?)\rangle\ \text{FO}_1$. It is replaced by $\langle\ is\_about^- \cup (k?); \nu_{\Sigma_1}; (l?)\rangle l$. It can also be rewritten as $\langle\ is\_about^-\rangle l \vee \langle k?\nu_{Sigma_1}\rangle l$ as $\langle\phi?\rangle\psi$ is equivalent to $\phi \wedge \psi$.
- All occurrences of ν_{Σ_1} in *Post* are replaced by $\nu_{\Sigma_1 \cup \{l\}}$ due to $[new\_node(l)]$

- PE occurs in $[\nu_{\Sigma_1 \cup \{l\}}(PA \vee\ MS) \Rightarrow PE$. It is replaced by $[\nu_{\Sigma_1 \cup \{l\}}(PA \vee\ MS) \Rightarrow (PE \vee k)$.
- PA occurs in $[\nu_{\Sigma_1 \cup \{l\}}(PA \vee\ MS) \Rightarrow (PE \vee k)$. It is replaced by $[\nu_{\Sigma_1 \cup \{l\}}](PA \vee k \vee\ MS) \Rightarrow (PE \vee k)$.
- PA_1 occurs in $\langle \nu_{Sigma_1 \cup \{l\}} \rangle\ \text{PA}_1 \wedge \phi$. It is replaced with $\langle \nu_{Sigma_1 \cup \{l\}} \rangle k \wedge \phi$.
- All occurrences of $\nu_{\Sigma_1 \cup \{l\}}$ in $Post$ are replaced by $\nu_{\Sigma_1 \cup \{k\} \cup \{l\}}$ due to $[new\_node(k)]$

We thus now have to prove that $Pre \wedge App(tag(\rho)) \Rightarrow ((\langle \nu_{\Sigma_1 \cup \{k\} \cup \{l\}} \rangle k \wedge (\langle is\_about^- \rangle l \vee \langle k? \nu_{\Sigma_1 \cup \{l\} \cup \{k\}} \rangle l) \wedge \langle\ ref\_phys \cup k?; \nu_{\Sigma_1 \cup \{l\} \cup \{k\}}; i? \rangle (i \wedge \text{PH}_1)) \wedge [\nu_{\Sigma_1 \cup \{l\} \cup \{k\}}](PA \vee k \vee\ MS) \Rightarrow (PE \vee k))$.

Let us work with $[\nu_{\Sigma_1 \cup \{l\} \cup \{k\}}](PA \vee k \vee\ MS) \Rightarrow (PE \vee k))$. As $Pre \Rightarrow [\nu_{\Sigma_1}](PA \vee k \vee\ MS) \Rightarrow (PE \vee k))$, only k and l actually interest us. k appears on both sides of the implication and l, not existing from Pre, cannot be labeled with PA or MS.

Let us deal with the first part now. $\langle \nu_{\Sigma_1 \cup \{k\} \cup \{l\}} \rangle k$ is trivially true. So is, $\langle \nu_{\Sigma_1 \cup \{k\} \cup \{l\}} \rangle k \wedge \langle k? \nu_{\Sigma_1 \cup \{k\} \cup \{l\}} \rangle l$ as k and l are in $\Sigma_1 \cup \{k\} \cup \{l\}$. Finally the same can be said for $\langle \nu_{\Sigma_1 \cup \{k\} \cup \{l\}} \rangle k \wedge \langle k? \nu_{\Sigma_1 \cup \{k\} \cup \{l\}}; i? \rangle (PH \wedge\ \text{PH}_1)$ as it is known from $App(tag(\rho))$ that $\langle \nu_{\Sigma_1} \rangle i \wedge PH \wedge\ \text{PH}_1$.

This transformation is thus correct.

7.6 Conclusion

In this chapter, we discussed another logic that can be used to describe graphs and their transformations. $\mathcal{C}2\mathcal{PDL}$ is an extension of dynamic logic that allows the use of more complex programs (namely converse), the use of nominals to uniquely identify nodes and a mechanism to handle nodes that do not currently exist.

$\mathcal{C}2\mathcal{PDL}$ allowed us to present an alternative logic to Description Logics as a possible choice to express graph properties and prove programs correct. It also gave us a better insight on what the conditions that were introduced in Chapter 5 imply as we have shown additional condition that rules that appear in closure strategies have to satisfy. Finally, and most importantly, $\mathcal{C}2\mathcal{PDL}$ allowed us to express reachability properties which are key in graph descriptions and transformations.

We proved that $\mathcal{C}2\mathcal{PDL}$ satisfies the conditions that we introduced is Chapter 5. It can express graph properties, it is closed under substitutions, for the actions that were introduced, it can express the application conditions given that restrictions on the left-hand sides of rules are enforced and its validity problem is decidable. We also gave an example proof for the running example of how to prove, intuitively, that a specification is correct.

Chapter 8

Graph Rewriting Systems

As explained in Section 1.3, imperative programming languages are far from being the only option to create graph transformations. In this chapter, the notion of *logically decorated rewriting systems*, LDRS, is introduced. It is the same as the one introduced in [14].

Simply told, a rewriting rule has two components: a left-hand side, sometimes called head of the rule, and a right-hand side, sometimes called body of the rule. The left-hand side of the rule is a graph that contains the nodes that the rule intends to modify while the right-hand side contains the actions are performed. In Section 8.1, the formal definition of a rule is given. Obviously, using just one rule does not provide much transforming power. LDRS's are sets of rules that are used to modify a graph. LDRS's are defined in Section 8.1 too.

The LDRS's that we use are extensions of graph rewriting systems as defined in [25] where graphs are attributed with logic formulae. In Section 8.1, details are given why the ability to use formulae as attributes is important. Among other things, it allows to express much more clearly and concisely transformations. The left-hand sides of the rules are thus logically decorated graphs whereas the right-hand sides are defined as sequences of atomic actions. These actions constitute the set of elementary transformations used in graph transformation processes as defined in Section 1.3.

Obviously, the rules that are used cannot refer to the actual graph that is modified. In order to find the pattern of the left)hand side of a rule inside an actual graph, matches are used. A match is a pair of functions (f_N, f_E) that associates to each node of the left-hand side a node of the actual graph and to each edge an edge of the actual graph. The match also checks that attributes of the elements of the left-hand side, both nodes and edges, correspond to the ones of the matched elements. The formal definition of a match is given in Section 8.2. Once each node and each edge of the left-hand side is matched to an element of the actual graph, the actions can be performed. Each action that should modify a node or an edge of the left-hand side modifies actually the image of that element by the match in the actual graph.

Now that it is clearer how to apply a rule, one needs to understand how the

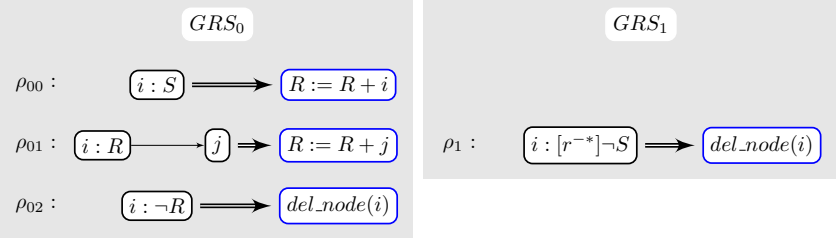


Figure 8.1: Two LDRS dealing with the suppression of unreachable nodes. Rules with atomic formulae are on the left. The rule with a non-atomic formula is on the right.

LDRS's modify graphs. An approach could be to choose to apply any rule for which a match can currently be found. This approach actually restricts drastically the power of the user as she is not able to tailor the application of the rules to its specific needs at a given point in the application of the transformation. A more suitable and adaptable approach is to define strategies that explain how the rules are to be applied. Strategies are formally introduced in Section 8.2 and examples are provided to help the reader grasp more easily how they work.

8.1 Graph Rewriting Systems

In this section, we introduce formally the notion of rewriting rule and of LDRS.

Definition 8.1.1 (Rule, LDRS)**.** *A* rule ρ *is a pair (LHS,*α*) where LHS, called the left-hand side, is an attributed graph with* $\mathcal{C}2\mathcal{PDL}$ *formulae as attributes and* α*, called the right-hand side, is an action. Rules are usually written* $LHS \to \alpha$*. A logically decorated rewriting system, LDRS, is a set of rules.*

It is noteworthy that the left-hand side of a rule is an attributed graph, that is it can contain nodes labeled with formulae. Once more, let us consider any possible logic. The ability to label nodes in rules with formulae is not insignificant. Indeed, these formulae can express reachability expression (closure of a program), non-local properties (universal program), ... These node labelings allow to write more concise and simpler rewriting systems. For instance, Figure 8.1 provides an example in which one wants to remove all unreachable nodes from a start state from an automaton. Two LDRSs, GRS_0 and GRS_1, for this problem are given. Without the closure constructor, one would point out that the start states are reachable (rule ρ_{00}), then say that every neighbor of a reachable node is reachable (rule ρ_{01}) and finally that all nodes that have not been reached by applying the first two rules as much as possible are to be removed (rule ρ_{02}). GRS_0 requires the explicit computation of the reachability making the algorithm more complex. On the other hand, GRS_1 only uses the one rule that says that all nodes that are not reachable from a start state are to be removed (rule ρ_{10}).

Figure 8.2: An example of LDRS where the first β after an α on a path toward a C becomes a γ. The dashed line represent the program α and the plain line the program β.

It is worth noting that the impact of formulae in node labels is not limited to graph rewriting. Indeed, let us consider the term-rewriting system of integer arithmetic with multiplication. The classical way to deal with 0s in such a case is to have rules saying that $0 \times x \rightsquigarrow 0$ and $x \times 0 \rightsquigarrow 0$. A wrong heuristic could lead to considerable calls in the case of a term like $a_0 \times \cdots \times a_k \times 0 \times a_{k+1} \times \cdots \times a_n$ where the a_is are terms. Considering terms as trees, and thus as graphs, it is possible to improve on this set of rules by using the one rule $i : \times \wedge \langle((L \cup R); \times ?)^* \rangle 0 \rightsquigarrow i : 0$ saying that a node i such that $i : \times \wedge \langle((L \cup R); \times ?)^* \rangle 0$, that is a node such that it is a multiplication $(i : \times)$ and there is a path of left- or right-operands of multiplications that leads to a 0 $(i : \langle((L \cup R); \times ?)^* \rangle 0)$, rewrites to 0.

Example 8.1.1. *Let us now give a small example of a slightly more involved rule in Figure 8.2. One possible interpretation would be that it is required to change the access policy in order to restrict acces to certain places. Once more, this can be done using several different rules instead of the one. For instance, a way to deal with that problem would be to label all nodes that access through paths composed only of edges labelled β a node labelled D in a first time. Then select those nodes labelled D that have an incoming edge labelled α and remove from all the outgoing edges of these nodes the label β and add the label γ.*

Rules allow to define more involved actions than atomic ones. Their power is still limited, in order to produce more interesting transformations, strategies, introduced in Section 8.2, are used. Nonetheless, up to now, each example was made of one single rule. In general, rewriting systems contain many more rules that are correspond to different actions.

Example 8.1.2. *As a more thorough example, let us provide a very simple graph rewriting system $\mathcal{R}$ that tries to produce a full and correct grid for sudoku. It contains 16 rules, that are summarized in Figure 8.3. The graph is defined such that all nodes correspond to one of the cells of the grid. The graph alphabet is $(\{P_1, P_2, P_3, P_4, 1, 2, 3, 4\}, \{R, C, SQ\})$. A node is labelled P_i's if i is considered a possible value for this cell of the grid. Meanwhile, a node is labelled i if i is known to be the value of this cell. Edges are never modified and define the relations between the cells. All cells on the same row are linked, from left to right, with edges labelled R, all cells on the same column are linked, from top to bottom, with edges labelled C and the top left cell of each square is linked to each other cell of the square via edges labelled SQ. The rules $\rho_{R,J}$ make sure that when a line (resp. a column or a square) contains a cell with value J $(i : \langle(r \cup r^-)^* \rangle J)$, P_J is no longer available for all the cells on the line (resp. column or square). The rules ρ_J are used to pick one choice among those that*

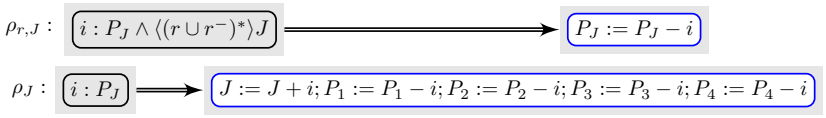


Figure 8.3: A summary of the rules of $\mathcal{R}$. In $\rho_{r,J}$, r must be replaced by either R, C or SQ and J by 1, 2, 3 or 4. In ρ_J, J must be replaced by 1, 2, 3 or 4

are available, namely J. It also removes all the possible choices are the value is now known.

Example 8.1.3. *We will use $\mathcal{C}2\mathcal{PDL}$ to state some properties of the Sudoku example. We are going to use a name for each cell of the grid (a_{ij} with $0 \leq i, j \leq 3$ where i is the row and j the column in which the cell can be found). We will also use three atomic programs R (resp. C and SQ) to state which cells are on the same row (resp. column and square). Finally, we will need eight atomic propositions* 1 *(resp.* 2,3 *and* 4*) and P_1 (resp. P_2,P_3 and P_4) to state that a cell* is known to *contain 1 (resp. 2, 3 or 4) and that a cell* may *contain 1 (resp. 2, 3 or 4). Thus we require that $\{a_{ij} | i, j \in [0, 3]\} \subset \Sigma$, $\{i | i \in [1, 4]\} \cup \{P_i | i \in [1, 4]\} \subset \Phi_0$ and $\{R, C, SQ\} \subset \Pi_0$. As we do not create or delete nodes, we do not make use of the possibility to change the set of definition of the total program. For this example, ν stands for ν_{Σ_1}.*

We define below a few relevant formulae.

- *The atomic program C should describe columns :*
  $c_j = \langle \nu \rangle (a_{0j} \wedge \langle C \rangle (a_{1j} \wedge \langle C \rangle (a_{2j} \wedge \langle C \rangle a_{3j})))$ for $j \in [0, 3]$. c_j describes the successive elements of a column.
  $\overline{c}_j = \langle \nu \rangle (a_{0j} \wedge [C](a_{1j} \wedge [C](a_{2j} \wedge [C](a_{3j} \wedge [C]\bot))))$ for $j \in [0, 3]$. $\overline{c}_j$ says that there are no more elements in a column than those specified by c_j. Thus a column is specified by $c_j \wedge \overline{c}_j$.
- *A cell should contain, at most, one value: $u_{I,J} = [\nu](I \Rightarrow \neg J)$ for $I, J \in [1, 4], I \neq J$ and there is no doubt about it (e.g., Once a cell is assigned the value I, it is no longer a candidate for any future potential assignement P_J): $\overline{u}_{I,J} = [\nu](I \Rightarrow \neg P_J)$ for $I, J \in [1, 4]$*
- *If a cell has a value J, J cannot be the value of any other cell on the same row, column or square $v_{r,J} = [\nu](J \Rightarrow (([r][r^*]\neg J) \wedge ([r^-][r^{-*}]\neg J)))$ for $J \in [1, 4]$ and $r \in \{R, C, SQ\}$.*

Example 8.1.4. *Finally, let us come back to the running Hospital example. Figure 8.4 contains one rule for each of the actions defined in Section 3.3.*

One can observe that the rules of Example 8.1.4 differ slightly in their definition from what has been introduced in Definition 8.1.1 in that the rules have parameters, for instance PH_1 and D_1 for New_Phys. This is actually due to a difference in the purpose of the transformations. In Example 8.1.1, the transformation is purely structural: one wants to change not a specific location but

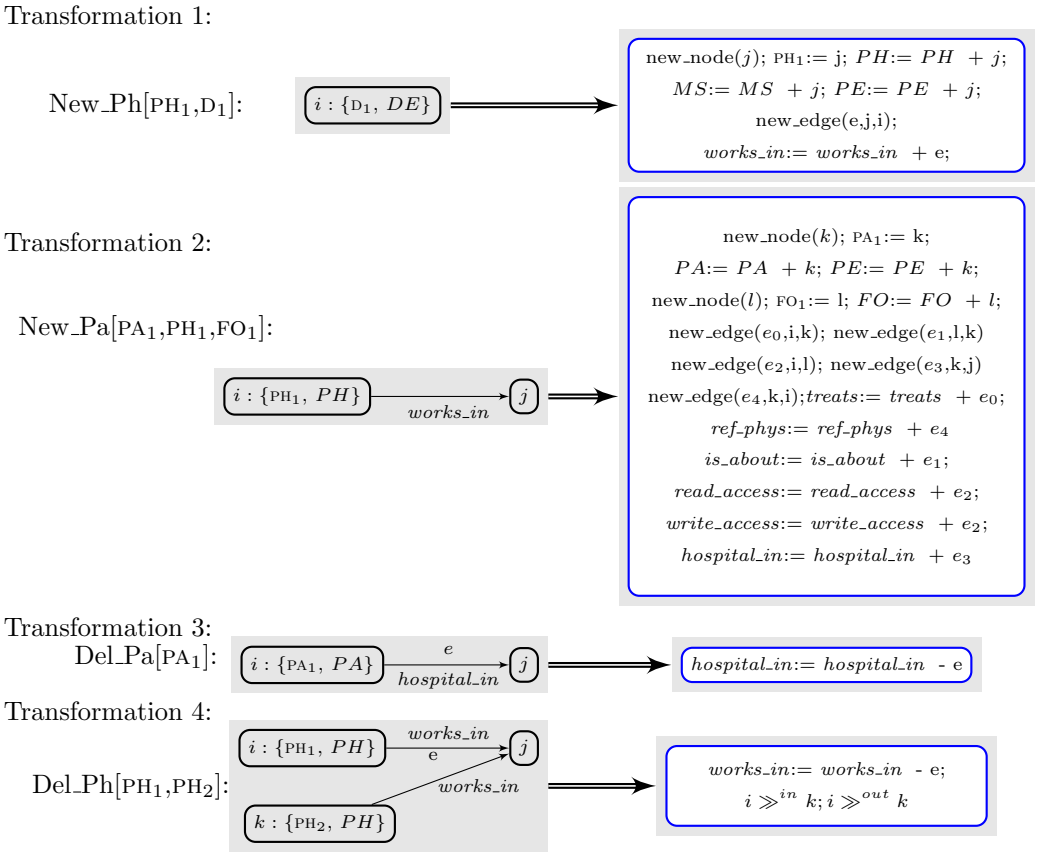


Figure 8.4: Transformation rules for the sample hospital model

all the accesses so that they are safer. Similarly, the goal of Example 8.1.2 is to globally affect the grid so that it becomes filled. On the other hand, in Example 8.1.4, the action are specific to a given set of individuals that are not chosen a priori by the programmer but will be selected depending on the use case. The definition of a rule is thus slightly modified.

Definition 8.1.2 (Parametrized Rule)**.** *A* parametrized rule $\rho[\vec{c}]$ *is a pair* (LHS,α) *where* LHS*, called the left-hand side, is an attributed graph with* $\mathcal{C}2\mathcal{PDL}$ *formulae as attributes and* α*, called the right-hand side, is an action.* $\vec{c}$ *is a vector of atomic concepts that are used to identify individuals. They can occur in the left-hand side to select a given individual or in the right-hand side to be set to be satisfied by a new element.*

8.2 The problem of the match

Now that it has been made clear what rules are, one should consider how to apply them. This is done through a match that scans the graph under study to check wether or not there exists a set of nodes that match with the left-hand side of a rule.

Definition 8.2.1 (Match)**.** *A* match h *between a left-hand side* LHS *and a graph* G *is a pair of functions* $h = (h_N, h_E)$*, with* $h_N : N_{LHS} \to N_G$ *and* $h_E : E_{LHS} \to E_G$ *such that:*

1. $\forall n \in N_{LHS}, \forall c \in L_{N_{LHS}}(n), h_N(n) \models c$

2. $\forall e \in E_{LHS}, L_{E_{LHS}}(e) = L_{E_G}(e)$

3. $\forall e \in E_{LHS}, s_G(h_E(e)) = h_N(s_{LHS}(e))$

4. $\forall e \in E_{LHS}, t_G(h_E(e)) = h_N(t_{LHS}(e))$

The third and the fourth conditions are classical and say that the source and target functions and the match have to agree. The first condition says that for every node n of the left-hand side, the node to which it is associated, $h(n)$, in G has to satisfy every concept that n satisfies. This condition clearly expresses additional negative and positive conditions which are added to the "structural" pattern matching. The second one ensures that the match respects edge labeling.

Definition 8.2.2 (Rule application)**.** *A graph* G *rewrites to graph* G' *using a rule* $\rho = (LHS, \alpha)$ *iff there exists a match* h *from* LHS *to* G*.* G' *is obtained from* G *by performing actions in* $h(\alpha)$[1]*. Formally,* $G' = G[h(\alpha)]$*. We write* $G \to_\rho G'$ *or* $G \to_{\rho,h} G'$*.*

[1] $h(\alpha)$ is obtained from α by replacing every node name,n, of LHS by $h(n)$.

Confluence of graph rewriting systems is not easy to establish. For instance, orthogonal graph rewrite systems are not always confluent, see e.g.,[25]. That is why we use a notion rewrite strategies to control the use of possible rules. Informally, a strategy specifies the application order of different rules. It does not ensure confluence, however. It does not point to where the matches are to be found and it contains a choice constructor that is inherently non-confluent. This is not a problem as scores of interesting rewriting problems are non-confluent.

Definition 8.2.3 (Strategy)**.** *Given a graph rewriting system $\mathcal{R}$, a* strategy *is a word of the following language defined by s:*

$s :=$ ϵ *(Empty strategy)* | $s \oplus s$ *(Choice)* | ρ *(Rule)* | s^* *(Closure)* | $s; s$ *(Composition)*

where ρ is any rule in $\mathcal{R}$.

We write $G \Rightarrow_{\mathcal{S}} G'$ when G rewrites to G' following the rules given by the strategy $\mathcal{S}$.

Informally, the strategy "$\rho_1; \rho_2$" means that rule ρ_1 should be applied first, followed by the application of rule ρ_2. The strategy "$\rho_0^*; (\rho_1 \oplus \rho_2)$" means that rule ρ_0 is applied as far as possible, then followed either by ρ_1 or ρ_2. It is worth noting that the closure is the standard "while" construct: if the strategy we use is s^*, the strategy s is used as long as it is possible and not an undefined number of times.

Example 8.2.1. *For the sudoku example introduced in Example 8.1.2, a possible strategy $\mathcal{S}_1$ could be: "As long as one can eliminate possibilities, do it. Then, when it is no longer the case, make a choice in one of the blank cells and go back to the first step". $\mathcal{S}_1$ may be defined as $\mathcal{S}_1 = ((\bigoplus_{r\in\{R,C,SQ\},J\in[1,4]} \rho_{r,J})^+; (\bigoplus_{J\in[1,4]} \rho_J))^*$ where s^+ stands for $s; s^*$.*

In Figure 8.5, we provide the rules that specify how strategies are used to rewrite a graph. For that we use the following two litterals: $\mathbf{App}(\mathtt{s}, G)$ which holds whenever graph G can be rewritten by the strategy $\mathtt{s}$ whereas $\mathbf{NApp}(\mathtt{s}, G)$ holds whenever graph G cannot be rewritten by strategy $\mathtt{s}$. These litterals are defined below

$\mathbf{App}(\rho, G) = \top$ iff there exists a match h from the left-hand side of ρ to G

$\mathbf{App}(\epsilon, G) = \top$ $\quad$ $\mathbf{App}(s_0 \oplus s_1, G) = \mathbf{App}(s_0) \vee \mathbf{App}(s_1)$

$\mathbf{App}(s_0^*, G) = \top$ $\quad$ $\mathbf{App}(s_0; s_1, G) = \mathbf{App}(s_0)$

It is worth noting that $\mathbf{App}(s, G)$ is not meant to denote that the whole strategy can be applied to G, just that the next step can be applied. Indeed, let's assume the strategy $s = (s_0; s_1)^*; s_2$ where s_0 can be applied but may yield a state where s_1 cannot. $\mathbf{App}(s)$ says that the strategy can be applied which is what we want as the fact that s_0 may yield a state where s_1 cannot be applied indicates a shortcoming of the programmation (such should not be the case) and the program will be considered incorrect. $\mathbf{NApp}$ is defined as $\neg\mathbf{App}$.

In addition, $App(\rho[\vec{c}])$ must be defined in $\mathcal{L}$. Obviously, this depends immensely on the rules one wants to use. It is thus possible, for a given problem, to use a logic that may not be powerful enough for other problems. Nonetheless, one of the requirements this entails on $\mathcal{L}$ is that it must allow some kind

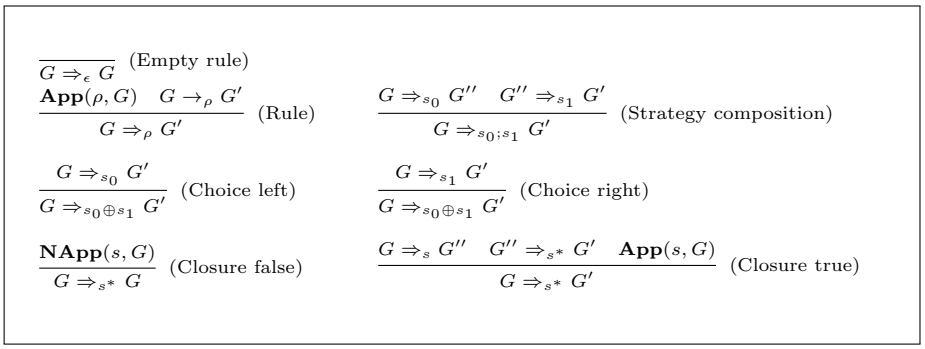


Figure 8.5: Strategy application rules

of existential quantification so that the graph can be traversed to look for a match. Obviously, the $\exists$-quantifier of first-order logic is a prime candidate but some other mechanisms like individual assertions $a : C$ in Description Logics[5] or the @ operator of hybrid logic[4] can be used.

Requirement 5. *$\mathcal{L}$ must be able to express $App(\rho[\vec{c}])$ for all rules $\rho[\vec{c}]$ of the graph rewrite system under study.*

8.3 Hoare calculus

As previously, in order to show the correctness of a specification, we follow a Hoare-calculus style and compute the weakest precondition $wp(\mathcal{S}, Post)$. For that, we define the weakest preconditions of a formula $Post$ induced by a strategy, a rule, an action and an atomic action. The weakest precondition of an elementary action, say a, and a postcondition Q is once again defined as $wp(a, Q) = Q[a]$ where $Q[a]$ stands for the precondition consisting of Q to which is applied a substitution induced by the action a that we denote by $[a]$. The rules for atomic actions are the same as the ones in Figure 5.1. The weakest precondition calculus is presented in Figure 8.6 for actions and strategies.

$$wp(\epsilon,\ Q) = Q \qquad wp(a; \alpha,\ Q) = wp(a, wp(\alpha, Q))$$
$$wp(\epsilon,\ Q) = Q \qquad wp(s_0; s_1,\ Q) = wp(s_0, wp(s_1,\ Q))$$
$$wp(s_0 \oplus s_1,\ Q) = wp(s_0, Q) \wedge wp(s_1, Q) \qquad wp(s^*,\ Q) = inv_s$$
$$wp(\rho,\ Q) = App(tag(\rho)) \Rightarrow wp(tag(\alpha_\rho), Q)$$

Figure 8.6: Weakest preconditions for strategies. ϵ appears twice for the empty list of actions and the empty strategy.

Apart from $wp(\rho,\ Q)$, the weakest preconditions defined in here are the usual

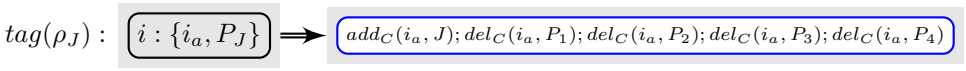


Figure 8.7: The rule ρ_J is modified into $tag(\rho_J)$ with $tag(i) = i_a$

ones. As in any other framework using Hoare logic, it requires the definition of an invariant for each loop that has to be provided by the user.

The weakest precondition of a rule $\rho = (lhs_\rho, \alpha_\rho)$ and a postcondition Q is given by $wp(\rho,\ Q) = App(tag(\rho)) \Rightarrow wp(tag(\alpha_\rho), Q)$ where $tag : N_{lhs_\rho} \to \Sigma$ is a function which associates to every node, n, of the left-hand side of rule ρ, a fresh name in Σ. These new names are used to keep track of the matching locations within potential graphs rewritten by different instances of ρ during the execution of a strategy. $tag(\rho) = (tag(lhs_\rho), tag(\alpha_\rho))$ where $tag(lhs_\rho)$ is a named graph which consists of the graph lhs_ρ where the node labeling function is augmented by tag, i.e., for all nodes, n, of lhs_ρ, $L_{N_{tag(lhs_\rho)}}(n) = L_{N_{lhs_\rho}}(n) \cup tag(n)$. $tag(\alpha_\rho)$ is obtained from α_ρ by substituting every node (in N_{lhs_ρ}), say i, by $tag(i)$. Fig.8.7 gives an example turning the left-hand side of a rule into a named graph via a function tag.

The formula $App(tag(\rho))$ expresses the applicability of the rule $tag(\rho)$. In other words, it expresses the existence of a match of the left-hand side of $tag(\rho)$.

Tagging a rule may seem to reduce its applicability. Indeed, by choosing a new name for each node of the left-hand side, the rule can now be applied only at the nodes of the graph named accordingly. Let ρ be a rule such that $LHS_\rho = (N_\rho = \{i_0, \ldots i_n\}, E_\rho, \mathcal{C}_\rho, \mathcal{R}_\rho, L_{N_\rho}, L_{E_\rho},$
$s_\rho, t_\rho)$ is its left-hand side. In order to prove that the application of ρ on a graph $G = (N_G, E_G, \mathcal{C}_G, \mathcal{R}_G, L_{N_G}, L_{E_G}, s_G, t_G)$ is correct, one has to verify that for every match $h = (h_N, h_E)$ (as defined in Definition 8.2.1), the postcondition is satisfied after the transformation associated with ρ is applied at the $h_N(i_k)$'s. Instead of showing that, the verification procedure proves that for any graph, if the rule can be applied at the $\theta(u)$'s, where θ is the function that associates to each name of Σ a node of G and $u \in tag(\rho)$, the postcondition is satisfied after performing the transformation. As the u's are fresh names, they do not have any impact on the previous characterization of the graphs. Thus, the validity of $wp(\rho, Post)$ actually states that whatever the choice of θ is, if the rule can be applied, then the postcondition will be satisfied after it is fired.

The weakest precondition associated to the closure of a strategy, say s^*, and a postcondition Q is defined as $wp(s^*,\ Q) = inv_s$ where inv_s is an invariant associated to the strategy s. Roughly speaking s^* could be seen as a while-loop which needs invariants associated with additional proof obligations defined by means of the function vc (verification conditions) given in Figure 8.8.

These definitions allow us to generate once more the formula $Corr$ which we have to prove valid.

$$vc(\rho,\ Q) = \top \qquad vc(s_0; s_1,\ Q) = vc(s_0, wp(s_1,\ Q)) \wedge vc(s_1, Q)$$
$$vc(s_0 \oplus s_1,\ Q) = vc(s_0, Q) \wedge vc(s_1, Q)$$
$$vc(s^*,\ Q) = (inv_s \wedge NApp(s) \Rightarrow Q) \quad \wedge (inv_s \Rightarrow wp(s, inv_s)) \wedge vc(s, inv_s)$$

Figure 8.8: Verification conditions

8.4 Conclusion

In this chapter, we introduced a transformation framework using rewrite rules and strategies. This is an approach that looks closer to the rewrite systems used in category theory yet with a more algorithmic flavour. Among its most interesting features, this frameworks allows to introduce explicit conditions on edge and node labels. It allows more flexibility on the side of the user.

As usual when using rewrite rules, the key to the transformations lies in the finding of a match. This introduces a new condition on the logic: in order to be able to prove the correctness of a rewrite system, a logic has to be able to express the existence of a match. This condition is the counterpart of the condition on the expressibility of the select that was introduced in Chapter 4.

Moreover, the user usually has a clear understanding of what transformations should be performed and simply applying any rule for which there is a match could yield many results that are unwanted. To tackle this problem, strategies are introduced to reduce the possible transformations to the ones that are intended.

Chapter 9

First-order logic and (some of) its fragments

Up to this chapter, several different logics have been featured that we deemed interesting to describe graphs and their transformations be them Description Logics, as in Chapter 6, or the logic we introduced in Chapter 7 that is closer to dynamic logics and thus allows more flexibility in term of edge description. For all these logics, we have observed that being able to express the applicability of the rules was the harshest condition that we would like to impose on them. This comes as no surprise as the definition of the match is a second-order formula with quantification on node and edge labels.

It is possible to express the existence or absence of a match in first-order logic though. In this chapter, we thus start by proving that first-order logic meets all the requirements that we have put forward. Except that it is not decidable.

In order to enquire a little further the condition on the expressibility of the existence or absence of match, we study in this chapter two well-known fragments of first-order logic that are known to be decidable yet close to the boundary with undecidable logics. The first logic we study is $\mathcal{C}^2$[34], the two-variable fragment of first-order logic with counting. This logic has been proven decidable yet, even when dropping the counting quantifiers, equality and constants, adding a third variable yields an undecidable logic[63]. This is a logic that is particularly interesting in that it contains all the Description Logics we considered extended with much more expressiveness in terms of edge descriptions. Its main shortcoming is, obviously, that it allows only for two variables and thus limits the graphs that can be described. The second logic we consider is $\exists^*\forall^*$[16], that is the fragment of first-order logic that contains, when written in prenex normal form, only formulae with every existential quantifiers preceding every universal quantifiers. These logics have been used in [12].

We prove that these logics are closed under substitutions, by proving that first-order logic is and showing that the translation used in that case can be

applied to these sub-logics, but focus much more on the fact that they can express the applicability of a rule provided that, different, restrictions on the rules are enforced. This, in turn, gives more insight on whether or not they are suitable for any given system of rules whose correctness one is interested in.

None of these three logics comes with a mechanism to create or remove elements. In order to circumvent this limitation, we add to the signature of the logic, that is to the set of node and edge labels, an atomic formula *Active*. Creating a new node becomes adding it to the *Active* nodes. This also requires to add that $\forall x, y. \neg Active(x) \Rightarrow (\bigwedge_{\psi \text{ an atomic unary predicate}} \neg\psi(x) \wedge \bigwedge_{r \text{ an atomic binary predicate}} \neg r(x,y) \wedge \neg r(y,x))$.

9.1 First-order logic

First-order logic, $\mathcal{FO}$, is a natural choice when one needs to pick a logic and verification of graph transformations is not an exception. Yet, it is far from perfect. First, it is not decidable and it thus goes against our stated goal to be able to fully automatize the decision procedure. The other thing one has to remember is that first order logic does not allow for parametrized quantifiers and thus one has to use the trick presented in Chapter 4 to avoid creating and deleting nodes instead adding a label *Active* that identifies the nodes that do exist and stating that nodes that are not labeled as existing cannot be labeled and have neither incoming or outgoing edges. With this limitation, one can then prove that $\mathcal{FO}$ is a suitable logic for the verification of graph transformations.

Theorem 9.1.1. *$\mathcal{FO}$ is closed under substitutions.*

Proof. Let c, c' be unary predicates, r be binary predicates, i, j be nominals, ϕ, ψ be formulae, σ an atomic action. We consider whether or not a formula is satisfiable at a node n,

- $(\exists x.\phi)[\sigma]) \rightsquigarrow \exists x.(\phi[\sigma])$ as the substitutions do not modify the existence or not of a node.
- $(\phi \wedge \psi)[\sigma] \rightsquigarrow \phi[\sigma] \wedge \psi[\sigma]$ as if $\phi \wedge \psi$ is satisfied after performing σ, so must be ϕ and ψ and the other way round.
- $(\neg\phi)[\sigma] \rightsquigarrow \neg(\phi[\sigma])$ as if ϕ is not satisfied after performing σ, it is not possible that ϕ be satisfied after performing σ.
- $\top[\sigma] \rightsquigarrow \top$ as no matter what action is performed, $\top$ is satisfied.
- $c'(x)[c := \phi] \rightsquigarrow c'(x)$ as the valuation of c' is left untouched.
- $c(x)[c := \phi] \rightsquigarrow \phi(x)$ as the $c^{I'}$ after performing $c := \phi$ is ϕ^I.
- $c'(x)[c := i] \rightsquigarrow c'(x)$ as the valuation of c' is left untouched.
- $c(x)[c := i] \rightsquigarrow i(x)$ as the $c^{I'}$ after performing $c := i$ is i^I.

- $c'(x)[c := c + i] \rightsquigarrow c'(x)$ as the valuation of c' is left untouched.
- $c(x)[c := c+i] \rightsquigarrow c(x) \vee i(x)$ as the $c^{I'}$ after performing $c := c+i$ is $c^I \cup i^I$.
- $c'(x)[c := c - i] \rightsquigarrow c'(x)$ as the valuation of c' is left untouched.
- $c(x)[c := c - i] \rightsquigarrow c(x) \wedge \neg i(x)$ as the $c^{I'}$ after performing $c := c - i$ is $c^I \backslash i^I$.
- $c(x)[r := (i, j)] \rightsquigarrow c(x)$ as the valuation of c is left untouched.
- $c(x)[r := \alpha] \rightsquigarrow c(x)$ as the valuation of c is left untouched.
- $c(x)[r := r + (i, j)] \rightsquigarrow c(x)$ as the valuation of c is left untouched.
- $c(x)[r := r - (i, j)] \rightsquigarrow c(x)$ as the valuation of c is left untouched.
- $c(x)[new(i)] \rightsquigarrow c(x)$ for $c \neq Active$ as the valuation of c is left untouched.
- $Active(x)[new(i)] \rightsquigarrow Active(x) \vee i(x)$ as the valuation of $Active$ becomes $c^{\mathcal{I}} \cup i^{\mathcal{I}}$.
- $c(x)[del(i)] \rightsquigarrow c(x) \wedge \neg i(x)$ as $c^{I'} = c^I \backslash i^I$.
- $c(x)[i \gg^{in} j] \rightsquigarrow c(x)$ as the valuation of c is left untouched.
- $c(x)[i \gg^{out} j)] \rightsquigarrow c(x)$ as the valuation of c is left untouched.
- $r(x, y)[c := i] \rightsquigarrow r(x, y)$ as the valuation of r is left untouched.
- $r(x, y)[c := \phi] \rightsquigarrow r(x, y)$ as the valuation of r is left untouched.
- $r(x, y)[c := c + i] \rightsquigarrow r(x, y)$ as the valuation of r is left untouched.
- $r(x, y)[c := c - i] \rightsquigarrow r(x, y)$ as the valuation of r is left untouched.
- $r'(x, y)[r := (i, j)] \rightsquigarrow r'(x, y)$ as the valuation of r' is left untouched.
- $r(x, y)[r := (i, j)] \rightsquigarrow i(x) \wedge j(y)$ as the valuation of r becomes $\{i^I, j^I\}$.
- $r'(x, y)[r := \alpha] \rightsquigarrow r'(x, y)$ as the valuation of r' is left untouched.
- $r(x, y)[r := \alpha] \rightsquigarrow \alpha(x, y)$ as the valuation of r becomes α^I.
- $r'(x, y)[r := r + (i, j)] \rightsquigarrow r'(x, y)$ as the valuation of r' is left untouched.
- $r(x, y)[r := r + (i, j)] \rightsquigarrow r(x, y) \vee (i(x) \wedge j(y))$ as $r^{I'}$ is $r^I \cup (i^I, j^I)$.
- $r'(x, y)[r := r - (i, j)] \rightsquigarrow r'(x, y)$ as the valuation of r' is left untouched.
- $r(x, y)[r := r - (i, j)] \rightsquigarrow r(x, y) \wedge (\neg i(x) \vee \neg j(y))$ as $r^{I'}$ is $r^I \backslash (i^I, j^I)$.
- $r(x, y)[new(i)] \rightsquigarrow r(x, y)$ as the valuation of r is left untouched.
- $r(x, y)[del(i)] \rightsquigarrow r(x, y) \wedge \neg i(x) \wedge \neg i(y)$ as $r^{I'} = r^I \backslash \{(a, b) | a \in i^I \text{ or } b \in i^I\}$.

- $r(x,y)[i \gg^{in} j] \rightsquigarrow (r(x,y) \wedge \neg i(y)) \vee (r(x,i) \wedge j(y))$ as $r^{I'} = r^I \cup \{(a,j) | (a,i) \in r^I\} \backslash \{(a,i) \in r^I\}$.
- $r(x,y)[i \gg^{out} j)] \rightsquigarrow (r(x,y) \wedge \neg i(x)) \vee (r(i,y) \wedge j(x))$ as $r^{I'} = r^I \cup \{(j,b) | (i,b) \in r^I\} \backslash \{(i,b) \in r^I\}$.

□

Additionally, $\mathcal{FO}$ is enough to express the existence or absence of a match. Let $App(\rho) = \exists_{n \in L_\rho} x_n . \bigwedge_{c \in L_N(n)} c(x_n) \wedge \bigwedge_{e \in L_\rho} L_E(e)(x_{s(e)}, s_{t(e)})$. $App(\rho)$ is equivalent to the existence of a match as $\bigwedge_{c \in L_N(n)} c(x_n)$ makes sure that the first condition of Definition 8.2.1 is satisfied and $\bigwedge_{e \in L_\rho} L_E(e)(x_{s(e)}, s_{t(e)})$ takes care of the other three.

Example 9.1.1. *First, let us observe that all of the invariants that we defined can be expressed in first-order logic (Formulae on the right).*

Property 1:		
$MS = NU \oplus PH$	$\rightsquigarrow$	$\forall x.\ MS(x) \Leftrightarrow (NU(x) \wedge \neg\ PH(x)) \vee (\neg NU(x) \wedge\ PH(x))$
Property 2:		
$PA \cup MS \subseteq PE$	$\rightsquigarrow$	$\forall x. PA(x) \vee MS(x) \Rightarrow PE(x)$
Property 3:		
write_access $\subseteq$ read_access	$\rightsquigarrow$	$\forall x, y.\text{write\_access}(x,y) \Rightarrow \text{read\_access}(x,y)$
Property 4:		
read_access $\circ$ is_about $\subseteq$ treats	$\rightsquigarrow$	$\forall x, y, z.\text{read\_access}(x,y) \wedge \text{is\_about}(y,z) \Rightarrow \text{treats}(x,z)$
Property 5:		
treats $\subseteq MS \times PA$	$\rightsquigarrow$	$\forall x, y.\text{treats}(x,y) \Rightarrow MS(x) \wedge PA(y)$
Property 6:		
$PA \Rightarrow \exists^{=1}$ ref_phys	$\rightsquigarrow$	$\forall x. PA(x) \Rightarrow (\exists y.\ \text{ref\_phys}(x,y) \wedge \forall z.\text{ref\_phys}(x,z) \Rightarrow z = y)$

First-order logic is not decidable though, and thus one may want to use a different logic in order to be able to decide the correctness of the considered properties. In the following, we use the 2-variable fragment of first-order logic with counting ($\mathcal{C}^2$)[34] and $\exists^*\forall^*$, the fragment of first-order logic whose formula in prenex form are of the form $\exists i_0, \ldots, i_k . \forall j_0, \ldots, j_l . A(i_0, \ldots, i_k, j_0, \ldots, j_l)$.

Let S be the specification $(Pre, s, Post)$ associated to the hospital example. Assume the strategy is $s = New\_Ph[\text{NPH,NEONAT}]; Del\_Pa[\text{OPA}]$ while $\mathcal{R}$ is the one from Figure 8.4. This program $\mathcal{P}$ creates a new physician NPH and lets the patient OPA leave the hospital. Let inv denote the conjunction of the expected properties. Let the precondition Pre be $inv \wedge \exists x.(\text{NEONAT}(x) \wedge \text{DE}(x)) \wedge \exists x.(\text{OPA}(x) \wedge \text{PA}(x)) \wedge \forall x. \neg \text{NPH}(x)$. Let the postcondition $Post$ be $inv \wedge \exists x, y.(\text{NPH}(x) \wedge \text{PH}(x) \wedge works\_in(x,y) \wedge \text{NEONAT}(y) \wedge DE(y))$. Proving the correctness of SPH amounts to proving that $Pre \Rightarrow wp(\mathcal{S}, Post)$ is valid. This is a formula in first-order logic. In the following two sections, this specification is proven to be

correct using two different decidable logics that are able to express part of Pre and $Post$.

9.2 $\mathcal{C}^2$

$\mathcal{C}^2$ is the two-variable fragment of first-order logic with counting. Its formulae are those of first-order logic than can be expressed with only two variables and using the counting quantifier constructor $\exists^{<n} r.C$. This constructor is such that a node m satisfies $\exists^{<n} r.C$ if there exist less than n different nodes $m_0, \ldots, m_n$ such that $r(m, m_i)$ and $C(m_i)$.

Definition 9.2.1. *Let $\mathcal{U}$ be a set of binary predicate, $u \in \mathcal{U}$, $\mathcal{B}$ be a set of binary predicate, $b \in \mathcal{B}$, v, w be either x or y, n an integer. A formula ϕ of $\mathcal{C}^2$ is defined as:*

$$\begin{array}{lcl} \phi & := & \top \mid \phi \wedge \phi \mid \neg\phi \mid \exists^{<n} v.\phi_v \\ \phi_v & := & \top \mid \phi_v \wedge \phi_v \mid \neg\phi_v \mid u(v) \mid \exists^{<n} v.\phi_v \mid \exists^{<n} w.\phi_{x,y} \text{ where } w \neq v \\ \phi_{x,y} & := & \top \mid \phi_{x,y} \wedge \phi_{x,y} \mid \neg\phi_{x,y} \mid u(v) \mid b(v,w) \mid \exists^{<n} v.\phi_{x,y} \end{array}$$

As usual, $\bot$ means $\neg\top$, $\phi \vee \psi$ means $\neg(\neg\phi \wedge \neg\psi)$, $\phi \Rightarrow \psi$ means $\neg\phi \vee \psi$, $\exists^{\geq n} v.\phi$ means $\neg\exists^{<n} v.\phi$, $\exists v.\phi$ means $\exists^{\geq 1} v.\phi$, $\forall v.\phi$ means $\neg\exists v.\neg\phi$.

Definition 9.2.2. *Let $\mathcal{G} = (N, E, \mathcal{C}, \mathcal{R}, \phi_N, \phi_E, s, t)$ be a graph. We defined the valuation of formulae:*

$$\begin{array}{lcl} \top^I & = & N \\ (\phi \wedge \psi)^I & = & \phi^I \cap \psi^I \\ (\neg\phi)^I & = & N \backslash \phi^I \\ (\exists^{<n} v.\phi_v)^I & = & \begin{cases} N & \text{if there does not exist } n \text{ nodes} v_1, \ldots, v_n \text{ such that } v_i \models \phi_v \\ \emptyset & \text{otherwise} \end{cases} \end{array}$$

Let us now focus on $v \models \phi_v$:

$$\begin{array}{lll} v \models \top & \text{iff} & \text{true} \\ v \models (\phi_v \wedge \psi_v) & \text{iff} & v \models \phi_v \text{ and } v \models \psi_v \\ v \models \neg\phi_v & \text{iff} & v \not\models \phi_v \\ v \models u(v) & \text{iff} & u \in \phi_N(v) \\ v \models \exists^{<n} v.\phi_v & \text{iff} & \text{there does not exist } n \text{ nodes } v'_1, \ldots, v'_n \\ & & \text{such that } v'_i \models \phi_v \\ v \models \exists^{<v} w.\phi_{x,y} & \text{iff} & \text{there does not exist } n \text{ nodes } w_1, \ldots, w_n \\ & & \text{such that } \begin{cases} (v, w_i) \models \phi_{x,y} & \text{if } w_i = y \\ (w_i, v) \models \phi_{x,y} & \text{if } w_i = x \end{cases} \end{array}$$

Let us now focus on $(x, y) \models \phi_{x,y}$:

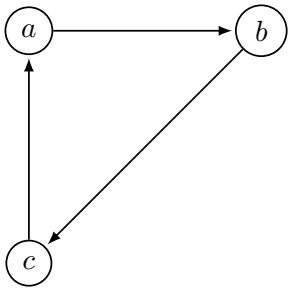


Figure 9.1: An example of graph that cannot be expressed in $\mathcal{C}^2$.

$(x,y) \models \top$	*iff*	*true*
$(x,y) \models (\phi_{x,y} \wedge \psi_{x,y})$	*iff*	$(x,y) \models \phi_{x,y}$ *and* $(x,y) \models \psi_{x,y}$
$(x,y) \models \neg\phi_{x,y}$	*iff*	$(x,y) \not\models \phi_{x,y}$
$(x,y) \models u(v)$	*iff*	$u \in \phi_N(v)$
$(x,y) \models b(v,w)$	*iff*	*there exists* $e \in E.s(e) = v$, $t(e) = w$ *and* $b \in \phi_E(e)$
$(x,y) \models \exists^{<n} x.\phi_{x,y}$	*iff*	*there does not exist* n *nodes* $x'_1, \ldots, x'_n$ *such that* $(x'_i, y) \models \phi_{x,y}$
$(x,y) \models \exists^{<n} y.\phi_{x,y}$	*iff*	*there does not exist* n *nodes* $y'_0, \ldots, y'_n$ *such that* $(x, y'_i) \models \phi_{x,y}$

Theorem 9.2.1 ([34])**.** *The validity problem of* $\mathcal{C}^2$ *is decidable.*

Ley us now check the requirements. $\mathcal{C}^2$ contains unary predicate that are interpreted on nodes and binary predicates that are interpreted on edges. *Pre* and *Post* are interpreted on graphs.

Theorem 9.2.2. $\mathcal{C}^2$ *is closed under substitutions.*

One can simply reuse the proof for $\mathcal{FO}$.

Theorem 9.2.3. $\mathcal{C}^2$ *cannot express App*(ρ)

The graph of Figure 9.1 cannot be expressed using only two variables and thus *App* for a rule which would have this graph as a left-hand side would not be expressible in $\mathcal{C}^2$.

$\mathcal{C}^2$ seems thus to be unfit for the verification of graph transformations. Actually, as was the case previously, it just means that one has to restrict the rules that one wants to verify to those that have left-hand sides that can be expressed in the logic.

Example 9.2.1. $\mathcal{C}^2$ *can express all the App*(ρ) *for the problem under study.*

Proof.

- $App(N\_Ph(\mathrm{PH}_1,\mathrm{D}_1)) = \exists x.(\mathrm{D}_1(x) \wedge \mathrm{DE}(x)) \wedge \exists x.(\neg Active(x) \wedge \mathrm{PH}_1(x))$

- $App(N\_Pa(\text{PA}_1,\text{PH}_1,\text{FO}_1,\mathsf{X})) = \exists x,y.(\text{PH}_1(x) \wedge \text{PH}(x) \wedge works\_in(x,y)) \wedge \exists x.(\neg Active(x) \wedge \text{PA}_1(x)) \wedge \exists x.(\neg Active(x) \wedge \text{FO}_1(x))$
- $App(D\_Pa(\text{PA}_1)) = \exists x,y.(\text{PA}_1(x) \wedge \text{PA}(x) \wedge hospital\_in(x,y))$
- $App(D\_Ph(\text{PH}_1,\text{PH}_2)) = \exists x,y.(\text{PH}_1(x) \wedge \text{PH}(x) \wedge works\_in(x,y) \wedge \exists x.(\text{PH}_2(x) \wedge \text{PH}(x) \wedge works\_in(x,y)))$

□

One could be interested in what are the class of graphs that can actually be expressed in $\mathcal{C}^2$. As shown in Figure 9.1, it is enough to have an undirected cycle of length at least three. If the biggest undirected cycle is of length at most two, it is possible to pick one node r as the "root" of the graph and state $App(\rho)$ as: $App(rho) = \exists x.App(r,x,\emptyset)$ where:

- $App(n,\kappa,\mathfrak{S}) = \bigwedge_{c \in L_N(n)} c(\kappa)$
  $\wedge \bigwedge_{e|s(e)=t(e)=n} L_E(e)(\kappa,\kappa)$
  $\wedge \bigwedge_{e,n' \in E_{n,\mathfrak{S}}} \exists \overline{\kappa}.((\bigwedge_{e'|s(e')=n \wedge t(e')=n'} L_E(e')(\kappa,\overline{\kappa}))$
  $\wedge (\bigwedge_{s(e')=n' \wedge t(e')=n} L_E(e')(\overline{\kappa},\kappa))$
  $\wedge App(n',\overline{\kappa},\mathfrak{S}))$
- $\kappa \in \{x,y\}$
- $\overline{x} = y$ and $\overline{y} = x$
- $E_{n,\mathfrak{S}} = \{e,n' | n' \notin \mathfrak{S} \wedge ((s(e) = n \wedge t(e) = n')) \vee (t(e) = n \wedge s(e) = n'))\}$

Example 9.2.2. *The computation of App is not very complex yet it needs an example to make it clearer. We compute step by step App for the graph of Figure 9.2. The node selected as the root is a. Thus App is* $\exists x.App(a,x,\emptyset)$. *a is labeled with A, does not have a self-loop and has two neighbours b and c. App is thus* $\exists x.(A(x) \wedge (\exists y.R(x,y) \wedge App(b,y,\{a\})) \wedge \exists y.(R(y,x) \wedge R'(y,x) \wedge App(c,y,\{a\})))$. *b is labeled with B and B′, does not have a self-loop and has only one neighbour different from a, e. Thus* $App(b,y,\{a\})$ *is* $B(y) \wedge B'(y) \wedge \exists x.(R(x,y) \wedge App(e,x,\{a,b\})$. *e is labeled with E, has an R-self-loop and one neighbour different from b and a, f. Thus* $App(e,x,\{a,b\})$ *is equal to* $E(x) \wedge R(x,x) \wedge \exists y.(R(x,y) \wedge App(f,y,\{a,b,e\})$. *e is labeled with E, does not have a self-loop and has no neighbour different from a, b and e. Thus* $App(f,y,\{a,b,e\}) = F(y)$. *c is labeled with C, does not have a self-loop and has one neighbour, d, different from a. Thus* $App(c,y,\{a\})$ *is equal to* $C(y) \wedge existsx.(R'(y,x) \wedge R(x,y) \wedge App(d,x,\{a,c\})$. *d is labeled with D, has no self-loop and no neighbour different from a and c. Thus* $App(d,x,\{a,c\}) = D(x)$. *Thus* $App = \exists x.(A(x) \wedge \exists y.(R(x,y) \wedge B(y) \wedge B'(y) \wedge \exists x.(R(x,y) \wedge E(x) \wedge R(x,x) \wedge \exists y.(R(x,y) \wedge F(y)))) \wedge \exists y.(R(y,x) \wedge R'(y,x) \wedge C(y) \wedge \exists x.(R(x,y) \wedge R'(y,x) \wedge D(x))))$.

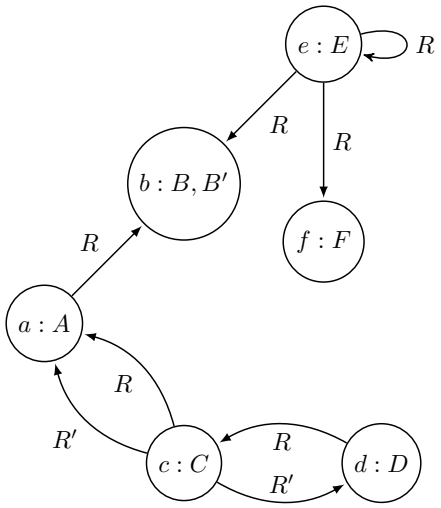


Figure 9.2: An example for the computation of *App*

These conditions are enough to be able to express the existence of the matches. Nonetheless, one could want to use $\mathcal{C}^2$ with more complex rules. In order to tackle this problem, the same idea as was used in Chapter 7 is used. We rename rules using special unary concepts o_i. These concepts are such that it is possible to uniquely identify elements inside the application of a rule so the condition that $\exists^{=1}x.o_i(x)$ is added for each o_i that is used this way. One could see that this condition states that the nodes where the actions will be performed are, so to speak, pre-selected that is the original model already contains nodes labeled with these concepts. This may seem wrong as one wants to prove that the post-condition will stand no matter where the match is found. The calculus actually solves the problem as it requires that the weakest precondition be valid. As these new concepts never occur outside of the scope of the match, given two graphs that differ only on which nodes satisfy these concepts, that is two graphs such that the nodes where the match are found are different, one can prove that the post-condition will stand.

As said in previously, $\mathcal{C}^2$ is not able to keep track of all variables so one needs to use concepts to keep track of the modified nodes. The actual concepts used are those obtained from *tag*. The expressions of *App* become:

- $App(N\_Ph(\text{PH}_1,\text{D}_1)) = \exists x.(\text{D}_1(x) \wedge \text{DE}(x) \wedge i(x)) \wedge \exists x.(\neg Active(x) \wedge \text{PH}_1(x) \wedge j(x))$
- $App(N\_Pa(\text{PA}_1,\text{PH}_1,\text{FO}_1)) = \exists x, y.(\text{PH}_1(x) \wedge \text{PH}(x) \wedge i(x) \wedge works\_in(x, y) \wedge j(y)) \wedge \exists x.(\neg Active(x) \wedge \text{PA}_1(x) \wedge k(x)) \wedge \exists x.(\neg Active(x) \wedge \text{FO}_1(x) \wedge l(x))$
- $App(D\_Pa(\text{PA}_1)) = \exists x, y.(\text{PA}_1(x) \wedge \text{PA}(x) \wedge i(x) \wedge hospital\_in(x, y) \wedge j(y))$

- $App(D\_Ph(\text{PH}_1,\text{PH}_2)) = \exists x,y.(\text{PH}_1(x) \wedge \text{PH}(x) \wedge i(x) \wedge works\_in(x,y) \wedge j(y) \wedge \exists x.(\text{PH}_2(x) \wedge \text{PH}(x) \wedge k(x) \wedge works\_in(x,y)))$

The same problem occurs also that these are not actually *App* and thus such rules cannot be used under the Kleene star in a strategy.

Furthermore, $\mathcal{C}^2$ is not able to express Property 4: $read\_access \circ is\_about \subseteq treats$ as one would need to keep track of three variables at a time. On the other hand, Property 6: $\forall x.\text{PA}(x) \Rightarrow \exists^{=1} ref\_phys.\top$ is a formula of $\mathcal{C}^2$.

9.3 $\exists^*\forall^*$

The logic $\exists^*\forall^*$ is the fragment of first-order logic such that its prefix in prenex normal form is composed of a sequence of existential quantifiers and then a sequence of universal quantifiers.

Definition 9.3.1. *Let $\mathcal{U}$ be a set of binary predicate, $u \in \mathcal{U}$, $\mathcal{B}$ be a set of binary predicate, $b \in \mathcal{B}$, $x_1,\ldots,x_k,a_1,\ldots,a_l$ be variables, v,w be two of them. A formula ϕ of $\exists^*\forall^*$ is defined as:*

$$\begin{array}{lll} \phi & := & \exists x_0,\ldots,x_k,\forall a_0,\ldots,a_l.\psi(x_1,\ldots,x_k,a_1,\ldots,a_l) \\ \psi & := & \top \mid \psi \wedge \psi \mid \neg\phi \mid u(v) \mid b(v,w) \end{array}$$

As usual, $\bot$ means $\neg\top$, $\phi \vee \psi$ means $\neg(\neg\phi \wedge \neg\psi)$, $\phi \Rightarrow \psi$ means $\neg\phi \vee \psi$.

Definition 9.3.2. *Let $\mathcal{G} = (N,E,\mathcal{C},\mathcal{R},\phi_N,\phi_E,s,t)$ be a graph. We defined the valuation of formulae: $(\exists x_1,\ldots,x_k,\forall a_1,\ldots,a_l.\psi(x_0,\ldots,x_k,a_0,\ldots,a_l))^I = N$ iff there exist k nodes $(x_1,\ldots,x_k)$ such that for all choices of l nodes $(a_1,\ldots,a_l)$, $(x_1,\ldots,x_k,a_1,\ldots,a_l) \models \psi$.*
Let us define $(x_1,\ldots,x_k,a_1,\ldots,a_l) \models \psi$:

$$\begin{array}{lll} (x_1,\ldots,a_l) \models \top & \textit{iff} & \textit{true} \\ (x_1,\ldots,a_l) \models (\phi \wedge \psi) & \textit{iff} & (x_1,\ldots,a_l) \models \phi \textit{ and } (x_1,\ldots,a_l) \models \psi \\ (x_1,\ldots,a_l) \models (\neg\phi) & \textit{iff} & (x_1,\ldots,a_l) \not\models \phi \\ (x_1,\ldots,a_l) \models u(v) & \textit{iff} & u \in \phi_N(v) \\ (x_1,\ldots,a_l) \models b(v,w) & \textit{iff} & \textit{there exists } e \in E.\ s(e) = v,\ t(e) = w \textit{ and } b \in \phi_E(e) \end{array}$$

Theorem 9.3.1. *The validity problem of $\exists^*\forall^*$ is decidable.*

This is a well-known result.

One now wants to check all the requirements. $\exists^*\forall^*$ contains unary predicate that are interpreted on nodes and binary predicates that are interpreted on edges.

Theorem 9.3.2. *$\exists^*\forall^*$ is closed under substitutions.*

Proof. Once again, one just needs to reuse the proof for $\mathcal{FO}$. □

The handling of the definition of a match is much more complex. To be more precise, $\exists^*\forall^*$ cannot always express $NApp(\rho)$ that is the absence of a match.

Theorem 9.3.3. *$\exists^*\forall^*$ cannot express all the $NApp(\rho)$.*

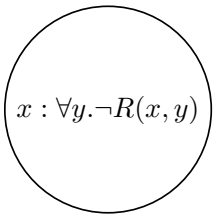


Figure 9.3: An example of graph not expressible in $\exists^*\forall^*$.

Proof. $\exists^*\forall^*$ is obviously not closed under negation and thus let us assume the existence of a rule ρ such that $App(\rho) = \exists x.\forall y.\phi(x, y)$ with $\phi(x, y)$ a quantifier-free formula using both x and y. Then $NApp(\rho) = \neg App(\rho) = \forall x.\exists y.\neg\phi(x, y)$ is not a formula of $\exists^*\forall^*$.

One thus just has to prove that such a rule exists. A rule whose left-hand side is the one presented in Figure 9.3 is a rule whose absence of a match cannot be expressed in $\exists^*\forall^*$. □

Once more, one could be interested in the class of graphs that can be expressed in $\exists^*\forall^*$. Let's assume none of the labels of the nodes contains quantifiers. Then, using the same construction as for $\mathcal{FO}$, one gets an essentially existentially quantified formula, that is a formula of $\exists^*\forall^*$. On the other hand, $\neg App$ is then essentially universally quantified and is thus also a formula of $\exists^*\forall^*$. It is possible to weaken this restriction by allowing nodes to be labeled with existential quantifiers but this does not increase the expressiveness as variables introduced that way could be replaced with additional nodes.

Example 9.3.1. $\exists^*\forall^*$ *can express all the* $App(\rho)$ *for the hospital problem.*

Proof.

- $App(N\_Ph(\mathrm{PH}_1,\mathrm{D}_1)) = \exists x.(\mathrm{D}_1(x)\wedge\mathrm{DE}(x)) \wedge \exists x.(\neg Active(x)\wedge\mathrm{PH}_1(x))$
- $App(N\_Pa(\mathrm{PA}_1,\mathrm{PH}_1,\mathrm{FO}_1)) = \exists x, y.(\mathrm{PH}_1(x)\wedge\mathrm{PH}(x) \wedge works\_in(x, y))$ $\wedge \exists x.(\neg Active(x)\wedge\mathrm{PA}_1(x)) \wedge \exists x.(\neg Active(x)\wedge\mathrm{FO}_1(x))$
- $App(D\_Pa(\mathrm{PA}_1)) = \exists x, y.(\mathrm{PA}_1(x)\wedge\mathrm{PA}(x) \wedge hospital\_in(x, y))$
- $App(D\_Ph(\mathrm{PH}_1,\mathrm{PH}_2)) = \exists x, y, z.(\mathrm{PH}_1(x)\wedge\mathrm{PH}(x)\wedge works\_in(x, y)\wedge\mathrm{PH}_2(z)\wedge$ $\mathrm{PH}(z) \wedge works\_in(z, y)))$

□

It is worth noting that the definition of $App(\rho)$ introduces new existential quantifiers as it checks for the existence of a match. This could seem to lead to a problem as the formula no longer is in $\exists^*\forall^*$. Actually, as the existentially quantified variables do not depend on the previously defined universally quantified variables, it is possible to move them at the beginning thus yielding a formula in $\exists^*\forall^*$.

Once again, one has to look at whether or not the properties of any given problem are expressible in the logic under study. As could be expected, there are properties that $\exists^*\forall^*$ cannot express.

Example 9.3.2. $\exists^*\forall^*$ *is not able to express Property 6:*$PA \Rightarrow \exists^{=1}$ ref_phys *as it needs an existential quantifier after the universal ones to express the existence of an edge labeled with* ref_phys. *On the other hand, Property 4:*$\forall x, y, z.\text{read\_access}(x, y) \wedge \text{is\_about}(y, z) \Rightarrow \text{treats}(x, z)$ *is part of* $\exists^*\forall^*$.

9.4 Conclusion

In this chapter, we explore the implications of the condition that the logic be able to express the existence of a match. We saw that this can result in either undecidability (as is the case for full first-order logic) or in the introduction of further restrictions on the rules in the case of sub-logics as $\mathcal{C}^2$ or $\exists^*\forall^*$.

These logics are both closed under substitution and can express the applicability of a restricted family of rules in addition to being decidable. They allow to express different properties of the graphs and are thus complementary and can be used in conjunction with other logics to express complex properties of rewriting systems that are in the intersection of the family of rules that can be expressed by each.

Chapter 10

Implementation

In the previous chapters, several different logics and forms of graph transformation have been introduced. We also proved that they satisfied the requirements that we put forward and thus that the verification could be fully automated. In addition, we would like that, when a specification is incorrect, a counter-example be shown. Nevertheless, we didn't show any system that would implement such proofs.

The reasons for that are multiple. First and foremost, our goal was to be able to reuse existing provers for the logics that we used. Several of them do not actually have any. Indeed, a $\mathcal{C}2\mathcal{PDL}$ formula, for instance, can be proven to be satisfiable using one of the two methods shown in Section 7.3 that is either concurrently trying to build a model for it and proving that its negation is valid or translating it to the Hybrid μ-calculus and using the algorithm for it. Neither, to the best of our knowledge, has been fully implemented. In a similar way, the fragments of first-order logic that have been studied in Chapter 9 are known to be decidable but there is no actual prover for them that uses the algorithm used to prove them. Indeed, similarly to $\mathcal{C}2\mathcal{PDL}$ it is proven that they have finite models but give no way to build the correct one.

One could argue that such provers, as for instance Vampire[1]Z3, E-Prover[2] or Spass[3], exist for first-order logic itself. The most obvious problem is the undecidability of the logic that yields an algorithm that could fail even on a decidable problem and thus strongly hinders our aim at fully automated proofs. Even when the logic actually provides a prover for a decidable logic, as is the case for the Description Logics for instance, the problem studied is not always the one we are aiming at. Indeed, in most provers as Fact++[66] or HermiT[32], the problem is to prove that a Knowledge Base is consistent. While we would like our proofs to output counter-examples, these provers are given an example and they want to prove that it makes sense. Thus they do not actually fulfill all

[1] http://www.vprover.org/

[2] http://www4.informatik.tu-muenchen.de/~schulz/E/E.html

[3] http://www.mpi-inf.mpg.de/departments/automation-of-logic/software/spass-workbench/

```
datatype ('nr, 'nc, 'ni) concept =
  Top
  | Bottom
  | AtomC bool 'nc
  | NegC "('nr, 'nc, 'ni) concept)"
  | BinopC binop "('nr, 'nc, 'ni) concept)" "('nr, 'nc, 'ni) concept)"
  | NumRestrC "(numres_ord)" "(nat)" "'nr" "('nr, 'nc, 'ni) concept)"
  | SubstC "('nr, 'nc, 'ni) concept)" "('nr, 'nc, 'ni) subst)"

datatype numres_ord = Lt | Ge

datatype binop = Conj | Disj

datatype ('nr, 'nc, 'ni) subst =
    RSubst "'nr" subst_op "('ni * 'ni)"
  | CSubst "'nc" subst_op "'ni"

datatype subst_op = SDiff | SAdd
```

Figure 10.1: The syntax of $\mathcal{ALCQ}$ concepts

our needs as they are not able to produce graphs that would be incorrect input for our specifications.

In this chapter, we present an implementation that we have performed to try to solve this problem. It slightly diverges from our stated goals as it uses $\mathcal{ALCQ}$ which has been proven not to be closed under substitutions. It is used in conjonction with the imperative language defined in Chapter 4. This work comes from [15] and [6].

10.1 Using Isabelle to prove soundness of the Hoare-like calculus

Our stated goal is to be able to prove that programs are correct. In such a situation, one expects that the proofs that are performed have some properties. Among those, we may expect that the automated generation of the proofs is actually sound itself, that is that you do not consider as theorems things that can be false.

In order to achieve such a property, we used Isabelle[51]. Isabelle is a proof assistant. We used it to define the syntax and semantics of $\mathcal{ALCQ}$ in addition to the substitutions associated to part of the actions introduced in Section 3.1.

One may remember that it was proven in Section 6.3 that $\mathcal{ALCQ}$ is not closed under substitutions. The same way, it lacks the expressivity needed to state the

existence or absence of a match. It is thus needed to change our approach a little. In order to tackle this problem, the logic is extended with quantifiers that introduce new variables. We then make sure that we generate only essentially universally quantified formulae as was explained in Section 5.4.

To be more precise, instead of working at concept level as we did previously when working with Description Logics, we use facts and formulae. A fact, in the present situation, is what is usually called an ABox assertion in the Description Logic. Formulae in addition use quantifiers and substitutions. They also use an explicit constructor for equality which makes the calculus considerable more complex but is essential to deal with aliasing problems.

Definition 10.1.1. *Given that ϕ is a concept, α is a role, a, b are nodes and f_0 and f_1 are facts, a* fact *is one of:*

- $a : \phi$ *that means that a is labeled with ϕ*
- $a\ \alpha\ b$ *that means that the edge (a, b) is labeled with α*
- $a\ \neg\alpha\ b$ *that means that the edge (a, b) is not labeled with α*
- $a = b$ *that means that a and b are equal*
- $a \neq b$ *that means that a and b are different*
- $\neg f_1$ *that means that f_1 is false*
- $f_0 \wedge f_1$ *that means that both f_0 and f_1 are true*
- $f_0 \vee f_1$ *that means that f_0 or f_1 is true*

Given that σ is a substitution, f is a fact, a is a node and F_0 and F_1 are formulae, a formula *is one of:*

- f *that means that f is true*
- $\forall a.F_1$ *that means that for all node n, F_1 is true when all occurrences of a are replaced by instances of n*
- $F_1\sigma$ *that means that F_1 is true after performing the atomic action associated to σ*
- $\neg F$ *that means that F_1 is false*
- $F_0 \wedge F_1$ *that means that both F_0 and F_1 are true*
- $F_0 \vee F_1$ *that means that F_0 or F_1 is true*

The syntax of $\mathcal{ALCQ}$ is fairly simple yet there are already proofs that have to be made to prove that the syntax is coherent. Indeed, more than the minimal amount of needed constructors are defined for $\mathcal{ALCQ}$. For instance, both $\phi \sqcap \psi$ and $\neg(\phi \sqcup \psi)$ are formulae that can be written using the defined constructors yet they should have the exact same interpretation. It is thus important to prove

```
record ('d, 'nr, 'nc, 'ni) Interp =
  interp_d :: 'd set
  interp_c :: 'nc ⇒ 'd set
  interp_r :: 'nr ⇒ ('d * 'd) set
  interp_i :: 'ni ⇒ 'd

fun interp_concept :: "('nr, 'nc, 'ni) concept ⇒
 ('d, 'nr, 'nc, 'ni) Interp ⇒ 'd set" where
  "interp_concept Bottom i = {}"
  | "interp_concept Top i = interp_d i"
  | "interp_concept (AtomC sign a) i = (if sign then (interp_c i a)
    else interp_d i - (interp_c i a))"
  | "interp_concept (BinopC bop c1 c2) i =
   interp_binopC bop (interp_concept c1 i) (interp_concept c2 i)"
  | "interp_concept (NegC c) i = interp_d i - (interp_concept c i)"
  | "interp_concept (NumRestrC nro n r c) i = {x ∈ interp_d i.
   interp_numres_ord nro (Range (rel_restrict (interp_r i r) {x}
    (interp_concept c i))) n}"
  | "interp_concept (SubstC c sb) i = interp_concept c (interp_subst sb i)"
```

Figure 10.2: The semantics of $\mathcal{ALCQ}$ concepts

that the semantic is the same for both. In order to spare space, only part of this implementation is shown in the following but a more complete version can be found at https://www.irit.fr/Climt/Software/smalltalc.html.

The first thing to do in Isabelle is to define the syntax of the logic itself. It is shown in Figure 10.1. Three sets are defined with for the concepts ('nc), roles ('nr) and individuals (or nodes) ('ni). There is only one unary constructor NegC for the negation, a constructor for the binary operators for conjunction and disjunction, another one for counting quantifier that uses a natural, a role, a concept and another one for the substitutions. The only elementary actions that are considered are $C := C+i$, $C := C-i$, $r := r+(i,j)$ and $r := r-(i,j)$. It is worth noting that, for simplicity of writing, the definition of atomic concepts can be negative or positive. The actual syntax is more complex as the logic does not provide nominals to deal with individuals. Facts are added to state whether or not nodes are equal, whether or not they belong to a concept and whether or not they are linked by a role.

One then has to introduce the semantics of the logic and prove that it is coherent. Figure 10.2 shows the semantics while Figure 10.3 shows some proofs. The first thing to introduce in the models is that nodes are interpreted as the basic elements of data ('d), that concepts are interpreted as sets and roles are interpreted as sets of pairs. interp_d returns the whole set of elements and is used to access the whole valuation. One can then introduce the valuation of

```
lemma Bottom_NumRestrC:
   "interp_concept Bottom i = interp_concept (NumRestrC Lt 0 r c) i"
 by simp

lemma Top_NumRestrC:
   "interp_concept Top i = interp_concept (NumRestrC Ge 0 r c) i"
 by simp
```

Figure 10.3: Short proofs about the semantics of $\mathcal{ALCQ}$ concepts

```
fun push_csubst_concept :: "'nc ⇒ subst_op ⇒ 'ni ⇒ 'ni
  ⇒ ('nr, 'nc, 'ni) concept ⇒ ('nr, 'nc, 'ni) form" where
 "push_csubst_concept cr rop v x (AtomC sign a) =
   (if cr = a then (case rop of
    SDiff ⇒ neg_norm_form sign
     (ConjFm (FactFm (Inst x (AtomC True a))) (FactFm (Eq False x v)))
   | SAdd ⇒ neg_norm_form sign
     (DisjFm (FactFm (Inst x (AtomC True a))) (FactFm (Eq True x v))))
  else (FactFm (Inst x (AtomC sign a))))"
```

Figure 10.4: Part of the elimination of substitutions

any concept of the logic. It is then interesting to use Isabelle to prove that our definitions are coherent. In particular, Figure 10.3 shows the proof that non node can have strictly less than 0 r-neighbours satisfying c and that all nodes have at least 0 r-neighbours satisfying c. This is done using only the definitions of the semantics.

One now has to deal with substitutions. This is where formulae become necessary to be able to express the result of substitutions. The tableau procedure that proves that a specification is correct actually uses only facts as it is the structure generated by the substitution elimination process. A few rules on substitutions, namely the ones dealing with atomic concepts and addition or removal of elements, are shown in Figure 10.4. This figure states that if the concept that is modified (cr) is different from the one in the formula (a) then the formula is left unmodified. On the other hand, if they are equal, and the substitutions removed an element then the formula is modified by adding the fact that the element that is removed is not the one that is considered. Mutatis mutandis, when adding an element it is added that either the formula was already true or the current element is the one that is added.

Once the removal of substitutions is done, one has to prove that it is correct and that it terminates. Partial proofs for these two properties can be found in

```
lemma interp_form_SubstFm_FactFm_Rel_AtomR_SDiff:
"interp_fact (AtomR sign r x y) (interp_subst (RSubst r SDiff (v1, v2)) i)
= interp_form (subst_AtomR_SDiff sign r x y v1 v2) i"
by (simp add: subst_AtomR_SDiff_def interp_r_modif_def) fast

fun subst_height_concept :: "('nr, 'nc, 'ni) concept ⇒ nat" where
  "subst_height_concept Bottom = 0"
 | "subst_height_concept Top = 0"
 | "subst_height_concept (AtomC sign a) = 0"
 | "subst_height_concept (BinopC bop c1 c2) =
    max (subst_height_concept c1) (subst_height_concept c2)"
 | "subst_height_concept (NegC c) = subst_height_concept c"
 | "subst_height_concept (NumRestrC nro n r c) =subst_height_concept c"
  | "subst_height_concept (SubstC c sb) =
    height_concept c + subst_height_concept c"

lemma height_concept_positive [simp]: "0 < height_concept c"
by (induct c) auto

lemma push_subst_fact_decr: "extract_subst fct = None ⟹
  subst_height_form (push_subst_fact fct sb) sbsts'
  < subst_height_fact fct (sb # sbsts')"
apply (case_tac sb)
apply (rule push_subst_fact_decr_rsubst) apply assumption+
apply (rule push_subst_fact_decr_csubst) apply assumption+
done
```

Figure 10.5: Proofs about the elimination of substitutions

```
fun wp_dl :: "('r, 'c, 'i) stmt ⇒ ('r, 'c, 'i) qform ⇒ ('r, 'c, 'i) qform" where
 "wp_dl Skip Qd = Qd"
  | "wp_dl (NAdd v c) Qd = QSubstFm Qd (CSubst c SAdd (Free v))"
  | "wp_dl (NDel v c) Qd = QSubstFm Qd (CSubst c SDiff (Free v))"
  | "wp_dl (EAdd v1 r v2) Qd =
    QSubstFm Qd (RSubst r SAdd (Free v1, Free v2))"
  | "wp_dl (EDel v1 r v2) Qd =
    QSubstFm Qd (RSubst r SDiff (Free v1, Free v2))"
  | "wp_dl (SelAss vs b) Qd =
    bind_list QAll vs (QImplFm (qform_of_form b) Qd)"
  | "wp_dl (c1 ; c2) Qd = wp_dl c1 (wp_dl c2 Qd)"
  | "wp_dl (IF b THEN c1 ELSE c2) Qd =
    QIfThenElseFm (qform_of_form b) (wp_dl c1 Qd) (wp_dl c2 Qd)"
  | "wp_dl (WHILE {iv} b DO c) Qd = (qform_of_form iv)"
```

Figure 10.6: Weakest preconditions in Isabelle

Figure 10.5. The first part is done by simply checking for each rule that the formula on the left and the one on the right agree on models. To prove the termination, a height is introduced that is proven to be natural, positive and strictly decreasing and thus convergent toward 0. It is worth noting that the definition of height used here is not the same as the one in Section 6.2 because the problems are slightly different.

The next step is proving that the weakest preconditions and verification conditions that were introduced in Section 5.2 are correct. This is done by first defining the functions that compute the weakest preconditions and the verification conditions. The weakest preconditions can be found in Figure 10.6. One then has to prove that the Hoare-like calculus is sound. This proof can be found in Figure 10.7. Actually, what is proven is that whenever $vc(s, Post)$ is valid, $\{wp(s, Post)\}s\{Post\}$ is a correct specification. It is easy knowing this theorem to prove that if $vc(s, Post) \wedge (Pre \Rightarrow wp(s, Post))$ then $\{Pre\}s\{Post\}$ is a correct specification.

10.2 Tableau procedure in Isabelle

Now that it has been proven that the Hoare-like calculus that we want to use is sound, it becomes possible to designe a procedure that checks whether or not a specification is correct. The only part that has not been shown yet is the one that determines whether a formula is valid or not. This is done using a tableau procedure. As previously, one of the main difficulties is that $\mathcal{ALCQ}$ is not closed under substitutions and cannot express the existence or absence of a match and that we are therefore forced to use facts and formulae as the basic blocks of the

```
lemma vc_sound:
  "valid_qform TYPE('d) (vc c Q) ⟹⊢ TYPE('d) {wp_dl c Q} c {Q}"
proof(induction c arbitrary: Q)
case (While iv b c)
show ?case
thm While'
proof(simp, rule While')
from ‘valid_qform TYPE('d) (vc (While iv b c) Q)‘
 have vc: "valid_qform TYPE('d) (vc c (qform_of_form iv))"
 and IQ: "valid_qform TYPE('d) (QImplFm (QConjFm (qform_of_form iv)
    (QNegFm (qform_of_form b))) Q)"
 and pre: "valid_qform TYPE('d) (QImplFm (QConjFm (qform_of_form iv)
    (qform_of_form b)) (wp_dl c (qform_of_form iv)))"
by (simp_all add: valid_qform_def Let_def)
 have "⊢ TYPE('d) {wp_dl c (qform_of_form iv)} c {(qform_of_form iv)}"
   by (rule While.IH [OF vc])
  with pre show "⊢ TYPE('d)
    {QConjFm (qform_of_form iv) (qform_of_form b)} c
    {(qform_of_form iv)}"
by(rule strengthen_pre)
show "valid_qform TYPE('d)(QImplFm (QConjFm (qform_of_form iv)
    (QNegFm (qform_of_form b))) Q)" by(rule IQ)
qed
next
case (Seq c1 c2) thus ?case by (auto simp add: valid_qform_def)
next
case (If b c1 c2) thus ?case apply (auto intro: hoare.conseq simp
    add: valid_qform_def lift_impl_def lift_ite_def)
apply (rule hoare.conseq) prefer 2 apply blast apply (clarsimp simp
    add: valid_qform_def QIfThenElseFm_def)+
apply (rule hoare.conseq) prefer 2 apply blast by (clarsimp simp
    add: valid_qform_def QIfThenElseFm_def)+
qed simp_all
```

Figure 10.7: Proof of the soundness of the Hoare-like calculus

```
definition "disjfm_applicable_branch br f1 f2 =
  (let fs = all_forms br in ¬(List.member fs f1) ∧ ¬(List.member fs f2))"

definition "apply_disjfm_branch br f f1 f2 =
  (if disjfm_applicable_branch br f1 f2
  then
    AppRes 2 (Some (DisjFmRule_rep f))
      [add_new_form f1 (add_inactive_composite f br),
      add_new_form f2 (add_inactive_composite f br) ]
  else AppRes 0 None [add_inactive_composite f br])"
```

Figure 10.8: Rule for the disjunction of formulae

tableau procedure.

The tableau procedure contains several rules that have been written in Isabelle. It starts from a set of formulae that contains only the formula whose satisfiability we are interested in and it generates a set of true formulae that can be deduced from that formula. Rules like the one for the disjunction of formulae, shown in Figure 10.8, create branches. To be more precise, this rule is applied when there is a fact of the form $F_0 \vee F_1$, but neither F_0 nor F_1 is present. A choice has thus to be made and two branches are created, one with F_0 and one with F_1. Branches allow to track choices in order to explore all possible cases. When one looks at a branch, there are two possible cases either the branch is saturated, that is it is no longer possible to apply any of the rules of the tableau or it is possible to go on. If one branche is saturated, it closes and the formula is satisfiable. On the other hand, if all branches yield a clash, that is something that cannot be a model, the formula was unsatisfiable. The list of possible clashs can be found in Figure 10.9.

Most clashes are self-evident. contains_bottom_branch occurs when the set of formulae contains $a : \bot$, contains_contr_concept_branch occurs when the set of formulae contains both $a : \phi$ and $a : \neg\phi$, contains_contr_role_branch occurs when the set of formulae contains both $a\ \alpha\ b$ and $a\ \neg\alpha\ b$, contains_contre_eq_branch occurs when the set of formulae contains $a \neq a$ and contains_falsefm_branch occurs when the set of formulae contains $\bot$. Clashes linked to counting quantifiers are a little bit more tricky: contains_numrestrc_clash_branch occurs when the set of formulae contains $a : (< n\ \alpha\ \phi)$ and there is a set of cardinality at least n that contains only α-neighbors of a labeled with ϕ.

This tableau procedure is proven to be sound, as shown in Figure 10.10. Indeed, it proves that if a branch contains no clash, is finite and saturated then it is satisfiable. In addition, the tableau procedure is proven to terminate and to be complete, that is that all invalid formulae yield only branches with clashes. It is thus possible to decide whether a formula is valid or not. Applied to the

```
definition "contains_bottom_branch br =
(case br of (Branch(n, alf, asf, ap, Inactive_form(ico, iac, iar, ie, icl))) ⇒
 contains_bottom_list icl) "

definition "contains_contr_concept_branch br =
(case br of (Branch(n, alf, asf, ap, Inactive_form(ico, iac, iar, ie, icl))) ⇒
 contains_contr_concept_list iac)"

definition "contains_contr_role_branch br =
(case br of (Branch(n, alf, asf, ap, Inactive_form(ico, iac, iar, ie, icl))) ⇒
 contains_contr_role_list iar)"

definition "contains_contr_eq_branch br =
(case br of (Branch(n, alf, asf, ap, Inactive_form(ico, iac, iar, ie, icl))) ⇒
 contains_contr_eq_list icl)"

definition "contains_falsefm_branch br =
(case br of (Branch(n, alf, asf, ap, Inactive_form(ico, iac, iar, ie, icl))) ⇒
 contains_falsefm_list icl)"

definition "contains_numrestrc_clash_branch br =
(case br of (Branch(n, alf, asf, ap, ia)) ⇒
 List.list_ex
  (λ f. case f of
   FactFm(Inst x (NumRestrC Lt n r c)) ⇒
    if n = 0 then True
    else exist_outgoing_r_c_distincts_from_branch br x n r c
   | _ ⇒ False) ap)"

definition "contains_clash_branch br = (case br of
 (Branch([], _, _, _, _)) ⇒
   (contains_bottom_branch br ∨
   contains_contr_concept_branch br ∨
   contains_contr_role_branch br ∨
   contains_contr_eq_branch br ∨
   contains_falsefm_branch br ∨
   contains_numrestrc_clash_branch br)
 | _ ⇒ False)"
```

Figure 10.9: Possible clashes

```
lemma not_contains_clash_sati: "
 ¬ contains_clash (ab:: ('nr, 'nc, 'ni::new_var_set_class) abox) ⟹
 finite ab ⟹
 is_neg_norm_abox ab ⟹
 saturated_abox ab alc_rule ⟹
 satisfiable TYPE('ni) ab"
apply (simp only: satisfiable_def)
apply(rule_tac x =
   "canon_interp (ab::('nr, 'nc, 'ni::new_var_set_class) abox)" in exI)
by (clarsimp simp add: canon_interp_satisfies_form)
```

Figure 10.10: Soundness of the tableau procedure

correctness formula that was generated by the Hoare-like calculus, it allows to decide whether a specification is correct or not.

In addition, the tableau procedure creates the canonical interpretation associated with a saturated branch, that is a branch in which it is no longer possible to apply any rule. This canonical interpretation can then be translated toward a graph language in order to be displayed. As it is a model that satisfies the negation of the formula that we aimed to prove valid, this interpretation gives us a counter-model that can be used to understand why our specification was incorrect and try to correct it.

10.3 Conclusion

In this chapter, we introduced an implementation that aims at bridging the gap between the theoretical results that have been shown in the previous chapters and our stated goal of having a fully automated proof system.

It actually uses a logic that we have proven not expressive enough to satisfy the conditions that we set in term of closure under substitutions. Nonetheless, by modifying $\mathcal{ALCQ}$ so that it becomes more expressive and using a dedicated tableau method, we managed to produce an implementation that is able to decide whether a program with conditions written in that logic is correct or not.

In addition to providing an answer, this implementation produces counter-models in case the program is incorrect so that it is possible for the user to see what goes wrong and correct either a problem in the program or a case that should have been treated in the precondition or the postcondition. On bigger programs, this becomes much harder to find the source of the problem and we are thus currently working on using the trace of the Hoare-like calculus to find out what part of the program lead to the problem.

It is worth noting that this implementation does not correspond to our stated

goal. Another implementation is being developed currently but is not mature enough to be featured in this work. As there exists a lot of SAT solvers for first-order logic, we decided to implement a program that would take a specification for a set of rules and tries to prove them correct by generating the correctness formula and checking it using one of these solvers. This is still slightly outside of our objective as first-order logic is undecidable but it allows us to give a better coverage of the various solutions that we have shown in the previous chapters.

Chapter 11

Conclusion and future work

11.1 Results

In this work, we discussed the verification of graph transformation. Graph transformation is ubiquitous and its correctness is thus of particular interest. Nevertheless, a lot of graph properties require an expressivity that leads to undecidability of their satisfiability problem. Our goal being to be able to decide automatically whether or not a graph transformation specification is correct, we restricted ourselves to problems that we could prove to be decidable. To do so, we used mainly decidable logics and we only studied partial correctness of a specification.

In order to showcase the transformations, we introduced two ways to modify graphs that are closely linked. Both rely on a set of atomic actions that perform elementary actions as creation or deletion of edges and nodes, relabeling and so on. Some of the atomic actions, as global redirections, can be seen as being more elaborate. Yet defining them as a sequence of atomic edge deletions and creations would be at best cumbersome. We combine these atomic actions in two different ways: either by using an imperative language[15] or using sets of rules and strategies[14].

The imperative language contains, in addition to the atomic actions, some usual constructs of imperative languages, as sequencing, if-then-else statements and while loops, and more specific constructs as the non-deterministic select that effectively instantiates a set of variables so that they match a pattern.

The other approach is closer to term-rewriting systems in that it defines rules and strategies. This approach is also closer to the categorial approach to graph transformations in its use of rules. Nonetheless, our rewriting systems strongly differ from this approach in that the right-hand side is composed of a sequence of atomic actions instead of containing a graph. This allows us to have a more algorithmic view of the transformation instead of relying on the definition of a single, double or sesqui-pushouts. Another key departure from usual methods is the use of complex formulae as labels in the left-hand side of the rules. This

allows for much more expressivity for the rules.

We defined a Hoare-like calculus for both approaches that introduces weakest preconditions to prove that a specification is correct. As both approaches contain loops, either under the form of while-loops in the imperative case or under the form of the closure strategy, the user is required to provide, in addition to the specification, invariants for them. They also force the introduction of verification conditions whose task is to prove that the invariants define correct specifications for the transformations inside the loops.

All in all, this Hoare-like calculus is not much more complex than the classical one. As usual, it introduces substitutions, that allow to forward the atomic actions to the underlying logic, that have to be accounted for. Its most complex feature is the need for the ability to express the existence of a match for the pattern defined by either a select-statement in the case of the imperative language or by the left-hand side of a rule in the graph rewriting system approach. Both substitutions and the existence of a match are unusual graph properties and thus need to be discussed when a logic is introduced to describe the graph properties[12].

What is obvious throughout the definition of the transformations, no matter the format, and the introduction of the Hoare-like calculus is that everything is parametrized by the logic, or logics, that have been chosen to define the properties of graphs. In this work, we aimed at being able as much as possible to provide definitive answers to the question of whether or not a specification for a given transformation is correct and we thus focused on decidable logics and looked at what we were requiring in addition from them.

The first set of logics that we proposed was the Description Logics family[13]. It is one of the most widely used family of logics to build ontologies, that is describing graphs, and it offers a huge array of possibilities from the tractable but little expressive $\mathcal{EL}$, DLP and $DL-Lite$ to the very expressive $\mathcal{SROIQ}$. We didn't consider all these logics but we focused on the central ones, that is the extensions of $\mathcal{ALC}$ that do not provide role axioms. These logics were instrumental in our endeavour to identify what are the required features for a logic to be able to deal with substitutions.

We also studied $\mathcal{C}2\mathcal{PDL}$, an extension of dynamic logics[14]. Our main goal in using this logic was to be able to express reachability properties that are key in the definition of a lot of structures, very interesting in conditions of graph transformations and outside of first-order logic. It was also helpful in looking more thoroughly at what where the implication of the requirement that the existence of a match be expressible in the logic.

Finally, several of the most expressive fragments of first-order logic, namely $\mathcal{C}^2$, the two-variable fragment with counting, and $\exists^*\forall^*$, the fragment that contains only formulae in which the prenex normal form is such that all existential quantifiers occur before every universal quantifiers were studied[12]. Once more, we proved that these logics were closed under substitutions and that, provided some restrictions on the rules, they could express the existence or absence of a match.

In order to make our work more concrete, we have started implementing

several different aspects of our systems. We wrote a tool that rewrites transformations written in our imperative language into Java code that can then be used more easily[7]. Closer to our goal of proving the correctness of graph transformation, we wrote a system for $\mathcal{ALCQ}$ using the imperative language that proves that a specification is correct and outputs a counter-example if it is not[15]. We also created a tool presented in [6] that combines the modification of graphs and the proof that it is correct.

11.2 Perspectives

This work is self-contained and gives an overview of a possible algorithmic approach to graph transformation, shows how to prove specifications for such transformation correct and provides keys, and possible choices, to determine the best logic or logics that one can use to express a given problem. Nonetheless, this is only the first step toward the original objective of proving correctness of graph transformation in all generality.

The main shortcoming of our results is that it does not allow for data. The most basic example of graph handling is list ordering that rewrites a list according to a given order notwithstanding any knowledge of the elements of the list. We do not allow for such comparisons either in the properties or the transformation itself as of today and it is a very important step that we need to be able to perform.

Another direction is the implementation. Heretofore, only a small part of the logics that we have presented are equipped with programs that actually prove whether or not a specification is correct. This is unsatisfactory as we would both like to be able to provides proofs no matter the choice of the - obviously decidable - logic but also because we would like to be able to give a tool that takes the specification and, using provers that it has been given, computes the proof without having to develop a different tool for each one. The main obstacle there is that even though the logic has to be closed under substitutions the provers for the logics do not remove them and thus it is mandatory to add a step, aware of the actual logic used, that removes the substitutions and this step can hardly be automated without knowledge of the logic.

It would be also interesting to try and extend the available atomic actions. Despite not being presented here, work has been started on the study of node cloning and its relation to the various logics we discussed in addition to considerations about how to handle edges in case of cloning. In particular, self-loops can be handled in several different manners that yield different atomic actions.

Other important directions are in the increase of the expressivity of the logic in order to be able to prove correct more interesting specifications and attempts at reducing the complexity of the proofs that are intractable in all cases considered thus far. This would go completely at reverse from what has been done here as the requirements that we introduced require expressiveness outside of the tractable logics but it would be particularly interesting to find a smaller set of actions such that specifications can be proven correct in low-

complexity logics.

Bibliography

[1] Shqiponja Ahmetaj, Diego Calvanese, Magdalena Ortiz, and Mantas Simkus. Managing change in graph-structured data using description logics. In *Proc. of the 28th AAAI Conf. on Artificial Intelligence (AAAI 2014)*, pages 966–973. AAAI Press, 2014.

[2] Carlos E. Alchourrón, Peter Gärdenfors, and David Makinson. On the logic of theory change: Partial meet contraction and revision functions. *J. Symb. Log.*, 50(2):510–530, 1985.

[3] Andrew W. Appel. Verismall: Verified smallfoot shape analysis. In *Certified Programs and Proofs - First International Conference, CPP 2011, Kenting, Taiwan, December 7-9, 2011. Proceedings*, pages 231–246, 2011.

[4] Carlos Areces and Balder ten Cate. Hybrid logics. In *Studies in Logic and Practical Reasoning*, volume 3, pages 821–868. Elsevier, 2007.

[5] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA, 2003.

[6] Nadezhda Baklanova, Jon Haël Brenas, Rachid Echahed, Amani Makhlouf, Christian Percebois, Martin Strecker, and Hanh Nhi Tran. Coding, executing and verifying graph transformations with small-t$\mathcal{ALCA}$e. *Graph Computation Models*, 2016.

[7] Nadezhda Baklanova, Jon Haël Brenas, Rachid Echahed, Christian Percebois, Martin Strecker, and Hanh Nhi Tran. Provably correct graph transformations with small-talc. In *Proceedings of the 11th International Conference on ICT in Education, Research and Industrial Applications: Integration, Harmonization and Knowledge Transfer, Lviv, Ukraine, May 14-16, 2015.*, pages 78–93, 2015.

[8] Philippe Balbiani, Rachid Echahed, and Andreas Herzig. A dynamic logic for termgraph rewriting. In *5th International Conference on Graph Transformations (ICGT)*, volume 6372 of *LNCS*, pages 59–74. Springer, 2010.

[9] Luciano Baresi and Paola Spoletini. *Procs. of ICGT 2006*, chapter On the Use of Alloy to Analyze Graph Transformation Systems, pages 306–320. Springer, 2006.

[10] Patrick Blackburn, Johan Van Benthem, and Frank Wolter, editors. *Handbook of Modal Logic*, volume 3 of *Studies in Logic and Practical Reasoning*. Elsevier, 2007.

[11] Elena Botoeva, Diego Calvanese, Valerio Santarelli, Domenico Fabio Savo, Alessandro Solimando, and Guohui Xiao. Virtual OBDA over expressive ontologies: Rewritings and approximations. In *Proceedings of the 29th International Workshop on Description Logics, Cape Town, South Africa, April 22-25, 2016.*, 2016.

[12] Jon Haël Brenas, Rachid Echahed, and Martin Strecker. Ensuring correctness of model transformations while remaining decidable. In *13th International Colloquium on Theoretical Aspects of Computing, ICTAC 2016*, 2016.

[13] Jon Haël Brenas, Rachid Echahed, and Martin Strecker. On the closure of description logics under substitutions. In *Proceedings of the 29th International Workshop on Description Logics, Cape Town, South Africa, April 22-25, 2016.*, 2016.

[14] Jon Haël Brenas, Rachid Echahed, and Martin Strecker. Proving correctness of logically decorated graph rewriting systems. In *1st International Conference on Formal Structures for Computation and Deduction, FSCD 2016, June 22-26, 2016, Porto, Portugal*, pages 14:1–14:15, 2016.

[15] Jon Haël Brenas, Rachid Echahed, and Martin Strecker. A Hoare-like calculus using the $SROIQ^{\sigma}$ logic on transformations of graphs. In Josep Diaz, Ivan Lanese, and Davide Sangiorgi, editors, *Theoretical Computer Science*, volume 8705 of *Lecture Notes in Computer Science*, pages 164–178. Springer Berlin Heidelberg, 2014.

[16] Egon Börger, Erich Grädel, and Yuri Gurevich. *The classical decision problem*. Springer, 2000.

[17] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Dl-lite: Tractable description logics for ontologies. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 602–607, 2005.

[18] Liang Chang, Fen Lin, and Zhongzhi Shi. A dynamic description logic for representation and reasoning about actions. In *Knowledge Science, Engineering and Management*, pages 115–127. Springer, 2007.

[19] Peter P. Chen. The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.

[20] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logics of Programs, Workshop, Yorktown Heights, New York, May 1981*, pages 52–71, 1981.

[21] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.

[22] R. Cyganiak, M. Lanthaler, and D. Wood. RDF 1.1 Concepts and Abstract Syntax. http://www.w3.org/TR/rdf11-concepts, 2014.

[23] Giuseppe De Giacomo, Maurizio Lenzerini, Antonella Poggi, and Riccardo Rosati. On instance-level update and erasure in description logic ontologies. *J. Log. Comput.*, 19(5):745–770, 2009.

[24] Ali Rezaei Divroodi and Linh Anh Nguyen. On bisimulations for description logics. *Information Sciences*, 295:465 – 493, 2015.

[25] Rachid Echahed. Inductively sequential term-graph rewrite systems. In *4th International Conference on Graph Transformations, ICGT*, volume 5214 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 2008.

[26] Cristina Feier, Antti Kuusisto, and Carsten Lutz. Fo-rewritability of expressive ontology-mediated queries. In *Proceedings of the 29th International Workshop on Description Logics, Cape Town, South Africa, April 22-25, 2016.*, 2016.

[27] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18(2):194–211, 1979.

[28] Giorgos Flouris, Dimitris Plexousakis, and Grigoris Antoniou. On applying the AGM theory to dls and OWL. In *The Semantic Web - ISWC 2005, 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10, 2005, Proceedings*, pages 216–231, 2005.

[29] Dov M. Gabbay, Agi Kuricz, Frank Wolter, and Michael Zakharyaschev, editors. *Many-dimensional modal logics: theory and applications.* Studies in Logic and the Foundations of Mathematics. Elsevier, 2003.

[30] Jelle Gerbrandy and Willem Groeneveld. Reasoning about information change. *Journal of Logic, Language and Information*, 6(2):147–169, 1997.

[31] Amir Hossein Ghamarian, Maarten de Mol, Arend Rensink, Eduardo Zambon, and Maria Zimakova. Modelling and analysis using GROOVE. *STTT*, 14(1):15–40, 2012.

[32] Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. Hermit: An OWL 2 reasoner. *J. Autom. Reasoning*, 53(3):245–269, 2014.

[33] Herman Heine Goldstine and John Von Neumann. Planning and coding of problems for an electronic computing instrument. *Institute for Advanced Study*, 1948.

[34] Erich Grädel, Martin Otto, and Eric Rosen. Two-Variable Logic with Counting is Decidable. In *Proceedings of 12th IEEE Symposium on Logic in Computer Science LICS '97, Warschau*, 1997.

[35] Fabio Grandi. On expressive number restrictions in description logics. In Carole Goble, Deborah L. McGuinness, Ralf Möller, and Peter F. Patel-Schneider, editors, *Proc. of the 14th Int. Workshop on Description Logics (DL 2001)*, volume 49 of *CEUR Workshop Proceedings*, pages 56–65. CEUR-WS.org, 2001.

[36] Fabio Grandi. On expressive description logics with composition of roles in number restrictions. In Matthias Baaz and Andrei Voronkov, editors, *Proc. of the 9th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2002)*, volume 2514 of *Lecture Notes in Computer Science*, pages 202–215. Springer, 2002.

[37] Annegret Habel and Karl-Heinz Pennemann. Correctness of high-level transformation systems relative to nested conditions. *Mathematical Structures in Computer Science*, 19(2):245–296, 2009.

[38] Terry Halpin. The ORM foundation. http://www.w3.org/2001/sw/wiki/OWL. Accessed: 2016-03-22.

[39] Jaakko Hintikka. Reasoning about knowledge in philosophy: The paradigm of epistemic logic. In *Proceedings of the 1st Conference on Theoretical Aspects of Reasoning about Knowledge, Monterey, CA, March 1986*, pages 63–80, 1986.

[40] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.

[41] Ian Horrocks and Ulrike Sattler. Decidability of $\mathcal{SHIQ}$ with complex role inclusion axioms. *Artificial Intelligence*, 160(1–2):79–104, 2004.

[42] Kazuhiro Inaba, Soichiro Hidaka, Zhenjiang Hu, Hiroyuki Kato, and Keisuke Nakano. Graph-transformation verification using monadic second-order logic. In *International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP)*, pages 17–28, 2011.

[43] ISO. UML, the unified modeling language. http://www.uml.org. Accessed: 2016-05-15.

[44] Shachar Itzhaky, Anindya Banerjee, Neil Immerman, Aleksandar Nanevski, and Mooly Sagiv. Effectively-propositional reasoning about reachability in linked data structures. In *Procs of CAV 2013*, pages 756–772, 2013.

[45] Daniel Jackson. *Software Abstractions*. MIT Press, 2011.

[46] Donald E. Knuth. Robert W. Floyd, In Memoriam. 2001.

[47] Barbara König and Javier Esparza. Verification of graph transformation systems with context-free specifications. In *5th International Conference on Graph Transformations (ICGT 2010)*, volume 6372 of *LNCS*, pages 107–122. Springer, 2010.

[48] K. Rustan M. Leino. Dafny: An automatic program verifier for functional correctness. In *Procs. of LPAR 2010*, pages 348–370. Springer, 2010.

[49] Xavier Leroy. Formal certification of a compiler back-end or: programming a compiler with a proof assistant. In *Proceedings of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2006, Charleston, South Carolina, USA, January 11-13, 2006*, pages 42–54, 2006.

[50] Greg Manning and Detlef Plump. The GP programming system. *ECEASST*, 10, 2008.

[51] Tobias Nipkow, Lawrence Paulson, and Markus Wenzel. *Isabelle/HOL. A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer Berlin / Heidelberg, 2002.

[52] Peter W. O'Hearn. A primer on separation logic (and automatic program verification and analysis). In *Software Safety and Security - Tools for Analysis and Verification*, pages 286–318. IOS Press, 2012.

[53] Fernando Orejas, Hartmut Ehrig, and Ulrike Prange. A logic of graph conditions extended with paths. *Graph Computation Models*, 2016.

[54] Solomon Passy and Tinko Tinchev. An essay in combinatory dynamic logic. *Inf. Comput.*, 93(2):263–332, 1991.

[55] Ruzica Piskac, Leonardo Mendonça de Moura, and Nikolaj Bjørner. Deciding effectively propositional logic using DPLL and substitution sets. *J. Autom. Reasoning*, 44(4):401–424, 2010.

[56] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society.

[57] Christopher M. Poskitt and Detlef Plump. A hoare calculus for graph programs. In *Procs. of ICGT 2010*, pages 139–154, 2010.

[58] Christopher M. Poskitt and Detlef Plump. Hoare-style verification of graph programs. *Fundamenta Informaticae*, 118(1-2):135–175, 2012.

[59] Christopher M. Poskitt and Detlef Plump. Verifying monadic second-order properties of graph programs. In *Procs. of ICGT 2014*, pages 33–48, 2014.

[60] John C. Reynolds. An overview of separation logic. In *Verified Software: Theories, Tools, Experiments, First IFIP TC 2/WG 2.3 Conference, VSTTE 2005, Zurich, Switzerland, October 10-13, 2005, Revised Selected Papers and Discussions*, pages 460–469, 2005.

[61] Ulrike Sattler and Moshe Y. Vardi. The hybrid μ-calculus. In *Proceedings of the First International Joint Conference on Automated Reasoning*, IJCAR '01, pages 76–91, London, UK, UK, 2001. Springer-Verlag.

[62] Oszkár Semeráth, Ágnes Barta, Zoltán Szatmári, Ákos Horváth, and Dániel Varró. Formal validation of domain-specific languages with derived features and well-formedness constraints. *International Journal on Software and Systems Modeling*, 07/2015 2015.

[63] Janos Surànyi. Zur reduktion des eitscheidungsproblem des logischen funktioskalküls. *Mathematikai és Fizikai Lapok*, 50:51–74, 1943.

[64] James Joseph Sylvester. On an application of the new atomic theory to the graphical representation of the invariants and covariants of binary quantics, - with three appendices. *American Journal of Mathematics, Pure and Applied*, 1:64–90, 1878.

[65] Stephan Tobies. The complexity of reasoning with cardinality restrictions and nominals in expressive description logics. *J. Artif. Intell. Res. (JAIR)*, 12:199–217, 2000.

[66] Dmitry Tsarkov. Incremental and persistent reasoning in fact++. In *Informal Proceedings of the 3rd International Workshop on OWL Reasoner Evaluation (ORE 2014) co-located with the Vienna Summer of Logic (VSL 2014), Vienna, Austria, July 13, 2014.*, pages 16–22, 2014.

[67] Julian Tschannen, Carlo A. Furia, Martin Nordio, and Nadia Polikarpova. *Procs. of TACAS 2015*, chapter AutoProof: Auto-Active Functional Verification of Object-Oriented Programs, pages 566–580. Springer, 2015.

[68] Alan M Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London mathematical society*, 42(2):230–265, 1936.

[69] Dániel Varró. Automated formal verification of visual modeling languages by model checking. *Software and System Modeling*, 3(2):85–113, 2004.

[70] W3C. SPARQL query language. http://www.ormfoundation.org/. Accessed: 2016-05-15.

[71] W3C. SPARQL query language. https://www.w3.org/2001/sw/wiki/SPARQL. Accessed: 2016-03-22.

[72] W3C. Word wide web consortium. https://www.w3.org. Accessed: 2016-03-22.

[73] Michael Wessel. Obstacles on the way to qualitative spatial reasoning with description logics: Some undecidability results. In *Working Notes of the 2001 International Description Logics Workshop (DL-2001), Stanford, CA, USA, August 1-3, 2001*, 2001.

[74] Frank Wolter and Michael Zakharyaschev. Dynamic description logics. *Advances in Modal Logic*, 2:431–446, 1998.

In computer science as well as multiple other fields, graphs have become ubiquitous. They are used to represent data in domains ranging from chemistry to architecture, as abstract structures or as models of the data or its evolution. In all these domains, graphs are expected to evolve over time due to chemical reactions, update of the knowledge or programs. Being able to deal with such transformations is an extremely important and difficult task. In this work, our aim is to study the verification of such graph transformation, that is how to prove that a graph transformation is correct. Correctness of a graph transformation is more precisely defines as correctness of a specification for the transformation containing additionally a precondition and a postcondition. We decided to use a Hoare-like calculus generating the weakest precondition for a postcondition and a transformation. If this weakest precondition is implied by the actual precondition, the specification is correct. We chose a more algorithmic approach to graph transformation by using atomic actions. We chose to define two ways to build graph transformations: using an imperative programming language and using rule-base rewriting systems. The main ingredient of the verification of graph transformation is the logic that is chosen to represent the precondition, the postcondition and the possible conditions internal to the transformation. So that the logic can interact with the calculus, we require that the decision problem be decidable, that the logic be closed under the substitutions introduced by the Hoare-like calculus and that it has to be able to express the existence and absence of a match for the transformation. The core result of this work is the identification and explanation of these conditions.

En informatique comme dans de multiples autres domaines, les graphes peuvent être trouvés partout. Ils sont utilisés pour représenter des données dans des domaines allant de la chimie à l'architecture, en tant que structures abstraites ou que modèles des données et de leurs évolutions. Dans tous ces domaines, il est prévisible que les graphes évoluent au cours du temps suite à des réactions chimiques, une mise à jour des connaissance ou l'exécution d'un programme. Être capable de traiter ces transformations est une tâche particulièrement importante et difficile. Dans ce travail, notre objectif est d'étudier la vérification de telles transformations de graphes, c'est à dire comment prouver qu'une transformation de graphes est correcte. La correction d'une transformation est plus précisément définie comme la correction d'une spécification pour cette transformation contenant en plus une précondition et une postcondition. Nous avons décidé d'utiliser un calcul à la Hoare générant une plus faible précondition pour une postcondition et une transformation. Si cette plus faible précondition est impliquée par la précondition, la spécification est correcte. Nous avons choisi une approche plus algorithmique pour les transformation de graphes utilisant des actions atomiques. Nous définissons deux moyens de construire des transformations de graphes: en utilisant un langage impératif ou en utilisant des systèmes de règles de réécriture. Le principal ingrédient est la logique qui est choisie pour représenter la précondition, la postcondition et les possibles conditions internes. Pour que la logique puisse interagir avec le calcul, nous demandons que le problème de décision soit décidable, qu'elle soit fermée par substitutions et qu'elle soit capable d'exprimer l'existence ou l'absence d'un sous-graphe affecté par la transformation. Le résultat central de ce travail est l'identification et l'explication de ces conditions.