université
PARIS DIDEROT
PARIS 7
ED386

Shape-Preserving Transformations of Higher-Order Recursion Schemes

Simulation of Evaluation Policies • Model Checking Problems

Axel Haddad

Thèse pour l'obtention du titre de Docteur de l'Université Paris 7 Spécialité Informatique. Le sujet est l'étude des transformations de schémas de récursion d'ordre supérieur préservant la structure. La thèse est soutenue publiquement le vendredi 6 décembre 2013 devant le jury suivant:

Arnaud Carayol	Encadrant de thèse
Andrzej Murawski	Examinateur
Luke Ong	Rapporteur
Olivier Serre	Directeur de thèse
Jean-Marc Talbot	Examinateur
Ralf Treinen	Président du Jury
Igor Walukiewicz	Rapporteur

Résumé

Les schémas de récursion d'ordre supérieur modélisent les programmes fonctionnels dans le sens qu'il décrivent les définitions récursives des fonctions créées par l'utilisateur sans interpréter les fonctions de base du langage. La sémantique d'un schémas est donc l'arbre potentiellement infini décrivant les exécutions d'un programme. Comme pour les langages de programmation, plusieurs politiques d'évaluation sont possibles, les deux plus importantes sont OI et IO, correspondant grossièrement aux politiques d'appel par nom et d'appel par valeur.

Cette thèse s'intéresse à des problèmes de vérification. Le problème de MSO-model-checking, c'est-à-dire décider si un schémas satisfait une formule de la logique monadique du second-ordre (MSO), a été résolu par Ong en 2006. En 2010, Broadbent, Carayol, Ong et Serre ont étendu ce résultat en montrant que l'on peut transformer un schémas de telle sorte que les nœuds de l'arbre associé satisfaisant une formule MSO donnée sont marqués, ce problème est appelé le problème de réflexion. Finalement, en 2012 Carayol et Serre ont résolu le problème de sélection : si l'arbre associé à un schéma donné satisfait une formule de la forme « il existe un ensemble de nœuds tels que… », alors on peut transformer le schéma tel qu'un ensemble témoin de la propriété est marqué dans l'arbre associé.

Ces deux derniers résultats ont été obtenus en utilisant des automates à pile avec effondrement, une classe d'automates equi-expressive des schémas pour générer des arbres infinis. Le revers d'une telle approche est que le schéma résultant est structurellement très différent de l'original : cela représente un problème lorsque l'on s'intéresse à la correction automatique de programme ou à la synthèse.

Dans cette thèse, nous suivons une approche sémantique pour étudier des problèmes de transformation de schémas. Notre objectif est de proposer des solutions qui conservent la structure du schémas donné en entrée. Dans cette idée, nous établissons un algorithme de simulation qui prend un schéma $\mathcal{G}$ et une politique d'évaluation $\pi \in \{OI, IO\}$ et produit un schémas $\mathcal{G}'$ tel que l'interprétation de $\mathcal{G}'$ avec la politique $\pi \neq \pi'$ est égal à l'interprétation de $\mathcal{G}$ avec la politique π. Ensuite nous donnons de nouvelles preuves de la réflexion et la sélection, sans passer par les automates à pile avec effondrement, qui conservent aussi la structure du schémas donné en entrée.

Abstract

Higher-order recursion scheme model functional programs in the sense that they describe the recursive definitions of the user-defined functions of a program, without interpreting the built-in functions. Therefore the semantics of a higher-order recursion scheme is the (possibly infinite) tree of executions of a program. As for programming languages several evaluation policies are possible, two of which are OI and IO, roughly corresponding to "call-by-name" and "call-by-value".

This thesis focus on verification related problems. The MSO model-checking problem, i.e. the problem of knowing whether the tree generated by a scheme satisfy an monadic second order logic (MSO) formula, has been solved by Ong in 2006. In 2010 Broadbent Carayol Ong and Serre extended this result by showing that one can transform a scheme such that the nodes in the tree satisfying a given MSO formula are marked, this problem is called the reflection problem. Finally in 2012 Carayol and Serre have solved the selection problem: if the tree of a given scheme satisfies a formula of the form "There exist a set of node such that…", one can transform the scheme such that a set witnessing the property is marked.

These two last results were obtained using collapsible pushdown automata, a class of automata equi-expressive with schemes to generate infinite trees. The drawback of going back and forth between collapsible pushdown automata and schemes is that the resulting scheme is structurally very far from the original one: this is a serious problem when one is interested in doing automated correction of programs or even synthesis.

In this thesis, we use a semantics approach to study scheme-transformation related problems. Our goal is to give shape-preserving solutions to such problems, i.e. solutions where the output scheme has the same structure as the input one. In this idea, we establish a simulation algorithm that takes a scheme $\mathcal{G}$ and an evaluation policy $\pi \in \{OI, IO\}$ and outputs a scheme $\mathcal{G}'$ such that the value tree of $\mathcal{G}'$ under the policy $\pi' \neq \pi$ is equal to the value tree of $\mathcal{G}$ under π. Then we give new proofs of the reflection and selection, that do not involve collapsible pushdown automata, and are again shape-preserving.

Remerciements

J'aimerais remercier mes encadrants Arnaud Carayol et Olivier Serre, qui m'ont guidé dans le monde de la recherche depuis mon stage de M2 jusqu'à maintenant. J'ai beaucoup appris à leurs côtés. Je remercie également mes rapporteurs Luke Ong et Igor Walukiewicz pour leurs remarques pertinentes et les examinateurs Andrzej Murawski, Jean-Marc Talbot et Ralf Treinen, pour avoir accepté de participer à cette soutenance de thèse. J'ai eu l'occasion de rencontrer au LIAFA et au LIGM certains chercheurs qui ont pu me donner de bons conseils aux bons moments, Claire, Florian et Paul-André, merci ! Ces deux labos ont pour point commun une très bonne entente entre les doctorants : merci Nath pour tes conseils, relectures, aides, discussions etc., Vincent pour tes blagues, Denis pour tes énigmes, Luc pour m'avoir donné tes points au projet PROLOG, Charles pour tes circuits et tout et tout, Charles pour tes traductions Melliès → Français, Sven et Vincent pour l'ambiance si cool à Marne, Elie pour ton état de prophète, et à tous les autres qui ont contribué à cette si bonne ambiance !

Rédiger une thèse n'est pas tous les jours facile, et je ne sais pas si j'y serais arrivé sans ce jour au croisement du boulevard de l'hôpital et de la rue Jeanne d'Arc où My tu m'as dit « T'as pensé à faire des timebox ? ». Donc merci beaucoup My pour avoir dit ça à ce moment là, et aussi pour m'avoir encouragé, de toute ta badassitude pendant cette rédaction. Et puisque je parle de timebox, vraiment merci Timo pour m'avoir accompagné dans ce délire là, et aussi pour toutes ces discussions qu'on a pu avoir sur plein de sujets, toutes plus intéressantes les unes que les autres.

Merci à chacun des membres de ma famille, à chacun de mes amis, pour m'avoir donné une part d'eux même. En plus de votre soutien, socle de ma thèse, ce sont toutes mes réflexions et mon caractère que vous avez participé à forger. De manière non exhaustive : merci maman pour m'avoir transmis le plaisir de parler tout seul, merci papa pour Carlos Santana, merci Marine de m'écouter comme tu le fais, Valentin pour ton sens du concept, Nonna pour l'Egypte des années cinquante, Geddo pour l'Alaska, Yas pour les vampires, merci Joelle pour l'épée magique, Jean-Pierre pour cet après-midi à faire de la voile, merci Patrick pour cette histoire de la balle qui t'a frôlé à la sortie d'un avion, Bernadette, pour le chech. Merci Raph pour la chambresque tapinière, merci Lionel pour cette discussion sur les «bofs», Bertrand pour ce solo, une fois, dans la salle de répèt., Pierre pour ce café à l'âge d'or, Mimi pour la bataille de chocopops au Portugal, Cléo pour «au bord de la mer», merci Tojo pour le Low Tech, Coco pour «Aretha !!!», Cathy et Sharrif pour

votre hospitalité, merci à Mélanie pour les petits prétentieux, merci à mes neveux Lazare Samuel Basile et Thomas d'être mes neveux, et merci à ceux que j'oublie sans doute !

Enfin, merci Djou, merveilleuse, avec qui j'ai la chance de partager ma vie. Ton intelligence, ton intégrité, ton honnêteté, m'émerveillent chaque jour, et sont pour moi une source d'espoir : ce sont les gens comme toi qui vont rendre ce monde meilleur !

Contents

1. Introduction **11**
- 1.1. A brief overview . . . 11
  - 1.1.1. Higher-order recursion schemes . . . 11
  - 1.1.2. Model-checking . . . 13
  - 1.1.3. Global model-checking, witness generation . . . 14
- 1.2. Model-checking results . . . 15
  - 1.2.1. The model-checking problem . . . 15
  - 1.2.2. Global model-checking, the endogenous approach . . . 17
- 1.3. About scheme transformations . . . 19
- 1.4. Structure of the thesis . . . 20

2. Preliminaries **23**
- 2.1. Introduction . . . 23
- 2.2. General background . . . 27
  - 2.2.1. Words . . . 27
  - 2.2.2. Relations, partial orders . . . 28
  - 2.2.3. Abstract reduction systems . . . 29
  - 2.2.4. Trees . . . 31
  - 2.2.5. Logic and automata . . . 33
  - 2.2.6. Term rewriting systems . . . 39
- 2.3. Higher order recursion schemes . . . 43
  - 2.3.1. Syntax . . . 43
  - 2.3.2. Semantics . . . 46
  - 2.3.3. Main algorithmic problems . . . 51
- 2.4. Equivalent models . . . 53
  - 2.4.1. λY-Calculus . . . 53
  - 2.4.2. Collapsible pushdown automata . . . 58
- 2.5. State of the art . . . 65
- 2.6. Scheme transformations . . . 67

3. Alternative Semantics for Higher Order Recursion Schemes **71**
- 3.1. Introduction . . . 71
- 3.2. Morphisms . . . 72
  - 3.2.1. Presentation and properties . . . 72
  - 3.2.2. Embedding a morphism into a scheme . . . 77

3.3. Extending scheme evaluation . . . 84
3.3.1. Continuous mappings . . . 85
3.3.2. Domain dependent value of a scheme . . . 88
3.3.3. The domain of morphisms . . . 91
3.3.4. The device morphism . . . 93
3.3.5. The device morphism computes the domain dependent value . . . 95
3.3.6. Applications . . . 102
3.4. Simulation of the IO evaluation policy . . . 104
3.4.1. Recognising IO-unproductive terms . . . 104
3.4.2. The IO-simulation . . . 111
3.5. Simulation of the OI evaluation policy . . . 116

4. Reflection and Selection **125**
4.1. Introduction . . . 125
4.2. Generalising the notion of runs to higher order terms . . . 127
4.2.1. Parity automata revisited . . . 127
4.2.2. An example of transitions on nonterminal . . . 130
4.2.3. Runs over terms . . . 135
4.3. Reflection . . . 139
4.3.1. Automata reflection . . . 139
4.3.2. MSO reflection . . . 141
4.4. Kobayashi-Ong Result . . . 147
4.4.1. Occurrences . . . 147
4.4.2. Parity games . . . 152
4.4.3. Adding colours to annotations . . . 154
4.4.4. Kobayashi-Ong game . . . 157
4.4.5. Solving problem 8 . . . 165
4.5. Selection . . . 167
4.5.1. Constructing the decorated ARS . . . 167
4.5.2. A winning condition for derivations . . . 171
4.5.3. Solving the selection problem . . . 180

5. Conclusion **185**
5.1. Contributions . . . 185
5.2. Perspectives . . . 187

A. Appendix **197**
A.1. Proof of Theorem **3.4.6** . . . 197

1. Introduction

1.1. A brief overview

1.1.1. Higher-order recursion schemes

The objects of this thesis are higher-order recursion schemes (HORS), a model for higher-order functional programs [Dam77a]. They were introduced in the mid 1970s to give a semantics to programs where the arguments of functions can be functions themselves, as in the programming language ALGOL 68 [Wij69, Lin93]. In the early 2000s, a new focus on recursion schemes appeared, in the field of infinite structure model-checking [Hun99, KNU01]. Indeed one can consider a higher-order recursion scheme as a finite generator of a possibly infinite tree, corresponding to the recursive-call tree of the program the scheme represents. The main problem was the decidability of the monadic second-order logic (MSO) model-checking over such trees, that Ong solved positively in 2006 [Ong06].

A scheme models the syntax of a recursive program; it is a set of pairs associating function symbols with their definitions, as examplified in Figure 1.1. It contains two kinds of symbols. The *non-terminal symbols* are the ones defined in the scheme (here *Start*, *Incr*, *Calc*, *Double* and *Fix*), and they correspond to the user-defined functions. The *terminal symbols* are the others (here $+$, $/$, 1 and 2), and they correspond to the built-in functions. In order to see a scheme as a generator of an infinite tree the pairs are considered as rewrite rules of a term rewriting system, and once we have fixed a start symbol (here *Start*), we define the *value tree* of the scheme as the tree obtained by applying the rewriting rules starting from the symbol *Start*. We present in Figure 1.2 a derivation of the scheme of Figure 1.1 (we represent in red the function we rewrite at each step), and its value tree is partially depicted in Figure 1.3. Although this is a simple example, it gives the intuition on how to define the value tree of a given scheme. Here the semantics of a scheme is a term rewriting system that produces a tree, however as we will see in Chapter 3, different semantics can be defined.

Historically, Nivat defined the notion of recursive program scheme [Niv72, Niv75] as

$$
\begin{array}{lcl}
\mathit{Start} & \to & \mathit{Double}(\mathit{Incr}, \mathit{Fix}(\mathit{Calc})) \\
\mathit{Incr}(x) & \to & 1 + x \\
\mathit{Calc}(x) & \to & 1 + x/2 \\
\mathit{Double}(\varphi, x) & \to & \varphi\big(\varphi(x)\big) \\
\mathit{Fix}(\varphi) & \to & \varphi\big(\mathit{Fix}(\varphi)\big)
\end{array}
$$

Figure 1.1.: A higher-order recursion scheme

$$\begin{aligned}
\mathit{Start} \;\to\; \mathit{Double}(\mathit{Incr}, \mathit{Fix}(\mathit{Calc})) \;\to\; & \\
\mathit{Incr}\big(\mathit{Incr}(\mathit{Fix}(\mathit{Calc}))\big) \;\to\; & 1 + \mathit{Incr}(\mathit{Fix}(\mathit{Calc})) \;\to \\
1 + 1 + \mathit{Fix}(\mathit{Calc}) \;\to\; & 1 + 1 + \mathit{Calc}(\mathit{Fix}(\mathit{Calc}))) \;\to \\
& 1 + 1 + 1 + \mathit{Fix}(\mathit{Calc})/2 \;\to\; \cdots
\end{aligned}$$

Figure 1.2.: A derivation of the scheme

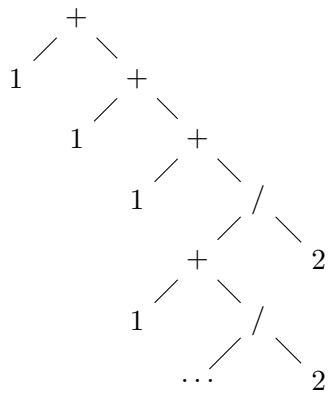


Figure 1.3.: The value tree of the scheme

a theoretical description of recursive functional program, as defined in LISP [McC62]. It was introduced to study semantics for recursive programs [Cou78a, Cou78b, CN78]. Functions defined in those schemes could not have functions as arguments. Therefore following the work of Indermark [Ind76], and motivated by programming languages like ALGOL 68, Damm introduced the notion of higher-order recursion scheme [Dam77a, Dam77b].

Another topic related to higher-order recursion schemes is the one of evaluation policies (call-by-name, call-by-value, etc.), as studied by Plotkin on the λ-calculus [Plo75]. Damm explored the OI and IO evaluation policies on safe higher-order recursion grammars [Dam82] (higher-order recursion grammars are a non-deterministic version of higher-order recursion schemes). Following a series of studies on the unsafe case [dG01, Mus01, AdMO05], Kobele and Salvati recently presented general results about this topic [KS13].

In this thesis, we are mainly interested about problems related to HORS model-checking. As we see in the next section, this topic rose in the early 2000s, starting with the works of Knapik Niwiński and Urzyczyn [KNU02].

1.1.2. Model-checking

Verification is the field of computer science dedicated to study the following very general question. Given an object (a machine, a program, an electrical circuit, etc.), and a property (a specification, a security constraint, an unwanted phenomenon), does the object satisfy the property? Although this question can never be solved in its full generality, the computer science approach consists in using modelisation to give a formal approximation of both the object and the property. Then, one may be able to decide whether the model of the object satisfies the model of the property; this solves the verification problem provided that the actual object and the actual property match their models. The model-checking problem [CE81, QS82] asks whether one can automatically verify if an object satisfies a property. More precisely given a class of objects $\mathcal{O}$ and a class of properties $\mathcal{P}$, the $(\mathcal{O}, \mathcal{P})$-model-checking problem asks if there exists an algorithm that takes as input a pair $(o, p) \in \mathcal{O} \times \mathcal{P}$ and outputs the truth value of "o satisfies p" (written $o \models p$). It is clear that there exists some pair $(\mathcal{O}, \mathcal{P})$ where the problem is undecidable (*e.g.* when $\mathcal{O}$ is the set of Turing machines and $\mathcal{P}$ is the set of non-trivial properties, as defined in Rice's theorem), and in general, model-checking results are of the form "the $(\mathcal{O}, \mathcal{P})$-model-checking problem is undecidable/decidable and complete with respect to the complexity class $\mathcal{C}$" for some $\mathcal{O}$, $\mathcal{P}$ and $\mathcal{C}$.

Most of the time, an object in $\mathcal{O}$ (generally called a model) is a relational structure, *i.e.* it is given by a set, and by a set of relations over that set. For example given an alphabet Σ, a labeled tree on Σ, as depicted in Figure 1.3 is given by a set whose elements are called *nodes*, such that

- with each node is associated a symbol (therefore the symbols are represented by unary relations),
- each node has finitely many *sons* (this is described by a binary relation),

- one node is distinguished as the *root* of the tree (represented by a unary relation),
- Every node is the son of exactly one other node except for the root that has no parent.

The properties are in general formulas of a logic, for example the monadic second-order logic (MSO) that can quantify over nodes and sets of nodes, that can test if some given nodes are in relation in the object, and that have the usual boolean operations.

At first, the models considered were finite,*e.g.* circuits, control flow graphs,etc. (see [CGP99] for an overview of finite model-checking). Then, from the mid 1990s, many articles studying the model-checking problem for classes of infinite objects flourished in the verification community [Wal96, BEM97]. Note that for infinite objects, the first input of the model-checking problem cannot be the object itself, but it has to be a finite structure describing the infinite object (*e.g.* regular expressions that describe languages, infinite graphs [Car06]). This difference with finite model-checking, raises a difficulty. For finite objects, the model-checking problem is very often decidable, as a "*try everything possible*"-algorithm usually works. On infinite objects, this is not the case anymore and before regarding the complexity of the problem, one must prove that it is decidable.

This thesis lies in the domain of infinite structures verification, where the models are infinite labelled ranked trees generated by higher-order recursion schemes, and the properties are described by MSO formula or infinite tree automata.

In the early 2000s, recursion schemes were reintroduced as generators of infinite trees as described earlier. Knapic et al. have proven the decidability of the model-checking of the MSO logic on infinite trees generated by *safe* HORS [KNU01, KNU02]. The safety restriction on HORS, first introduced by Damm [Dam82], is a syntactical constraint on the schemes; Parys has shown that safe schemes are a strict restriction of unsafe ones [Par12]. Ong has later proven the decidability of the model-checking of the MSO logic on infinite trees generated by any HORS [Ong06].

1.1.3. Global model-checking, witness generation

During the developments of the first model-checking programs [CES86, BCM+92], some features were implemented to enrich the software. We are interested in two of them, global model-checking and witness generation. A model-checking algorithm gives a yes or no answer to a precise question. For example on state systems, it can decide whether or not a given state s satisfies a state property $p(s)$. A *global model-checking algorithm* takes as input the structure and the state property p and outputs the set of states s such that $p(s)$ holds. Assume now that the model you are studying is a game (board game or other), and you ask the model-checker whether you have a winning strategy in that game. Even though the answer is "yes", you may want to know how to actually win the game. A *witness generation algorithm*, would decide if you have a winning strategy, and if it is the case, it would outputs a winning strategy. Witness generation can be useful for error detection: if a model-checker detects a security flaw in an encryption program,

a witness generation algorithm can highlight one of the flaws, allowing you to correct the program accordingly.

Although these features are usually easy to implement for a finite structure model-checker,for infinite structures the problem becomes more difficult. These are the topics we investigate in this thesis.

We explore the questions of global model-checking and witness generation regarding the model-checking of the MSO logic for infinite trees generated by higher-order recursion schemes. As in [BCOS10, CS12] we follow an endogenous approach *i.e.* given a scheme, we seek to produce another one whose value tree is the same as the original one, enriched with the output of the global model-checking/witness generation (we formalise this idea in Chapter 2). The originality of our approach compared to [BCOS10, CS12], is that we proceeds by using *shape-preserving scheme transformations*, *i.e.* the output scheme is obtained by enriching the first one and, as a consequence, it preserves the structure of the original scheme. We apply the techniques we introduced to study evaluation policies, by proving a simulation theorem between the OI and the IO evaluation policies.

We start with a short history of the recent developments of model-checking for HORS, as long as global model-checking and witness generation, and then we discuss our approach (a more precise state of the art, along with the statement of our contribution, is given at the end of Chapter 2, once the formal definitions have been presented). Finally, we give a brief summary of the content of this thesis.

1.2. Model-checking results

All model-checking results we present here are subresults, or equivalent results, of Ong's theorem establishing the decidability of the MSO model-checking on infinite trees generated by HORS [Ong06]. Historically, one can consider Rabin's Theorem, the decidability of the model-checking of MSO over the full, unlabeled, binary tree [Rab69], as the first step toward the study of model-checking on finitely described infinite structures. A second milestone could be the famous Muller and Schupp result [MS85], stating the decidability of the model-checking of MSO on infinite graphs generated by pushdown structures.

After presenting the model-checking results, we talk about global model-checking and then witness generation. In particular, we introduce the endogenous approach of [BCOS10, CS12].

1.2.1. The model-checking problem

From the mid 1990s, a series of articles nourished the domain of model-checking for finitely described infinite trees and graphs. They studied infinite structures generated by pushdown structure, iteration of operations, unfolding, etc. (see for example [Cou95, Wal96, Cau96, CW98, Wal01, Cau02]). Among structures less related to our problem but still interesting for model-checking one can quote timed-automata [ACD90], Petri nets [Esp94], etc. In the beginning of the 2000s, different equivalent results showed

that the infinite trees defined in these papers, can be generated by particular classes of higher-order recursion schemes. This motivated the study of model-checking on HORS.

The order of an HORS is the greatest order of its functions, defined in the usual way: an order-0 function is a function that has no argument (a constant), an order-1 function is a function whose arguments are order-0 functions, and more generally an order-i function is a function whose arguments have order less than i. As we said earlier, safety is a syntactical constraint of schemes, it was first introduced by Damm, and then a new formulation appeared in [KNU01]. Knapik Niwiński and Urzyczyn showed the decidability of MSO on the value tree of safe order-2 recursion schemes [KNU01]. Then, they proved the decidability of the whole class of safe recursion schemes [KNU02]. This result proceeds in two steps.

- First they show that the trees generated by safe order-n schemes are exactly the trees generated by order-n deterministic pushdown automata. Such an automaton is a stack automaton that handles a stack of order-n (order-1 stacks are the usual stacks of symbols, and order-$(n+1)$ stacks are the stacks of order-n stacks).

- Then, they proved the decidability of the MSO model-checking problem on trees generated by such automata.

This result asserted the safety constraint, and so did Caucal's result [Cau02]. He defined a hierarchy of trees, showed the decidability of MSO on every class of trees and establishing the equivalence of the hierarchy of trees generated by safe higher-order recursion scheme and his hierarchy. As remarked by Cachat and Walukiewicz [CW07], the n-EXPTIME completeness of this model-checking problem follows from the work of Engelfriet [Eng91] on iterated stack automata.

In 2005, the MSO model-checking of order-2 recursion scheme (whether safe or not), has been proven to be decidable. This result has been independently discovered in [KNUW05] and in [AdMO05]. In both paper, the proof consists in establishing an equivalence between order-2 recursion scheme and an extension of order-2 pushdown automata (called panic automata in [KNUW05], automata with links in [AdMO05]), and then proving the decidability of the MSO model-checking for the class of automata introduced. This result is extensively presented in [dM06].

Finally in 2006, Ong managed to prove the decidability of the MSO model-checking of any HORS. Unlike what was predicted in [KNUW05] and [AdMO05], the proof is not obtained by generalising to any order the automata introduced in these papers, and establishing an equivalence with HORS. Instead, the proof is obtained by analysing how the scheme produces its value tree, and defining a *computational tree* that describes this process; then by establishing a correspondence between paths in the value tree and paths in the computational tree; finally by defining a (big but finite) parity game and establishing another correspondence between paths in the computational tree and strategies in the game. Therefore the model-checking problem is reduced to the problem of finding a winning strategy in a finite parity game, which is solved [EJ91]. The problem is proven to be n-EXPTIME complete.

This result has been a major breakthrough in the domain. Although, as the author explains [KO09], "Ong's algorithm for verifying higher-order recursion schemes is rather complex and probably hard to understand: The algorithm reduces the model-checking problem to a parity game over variable profiles, and its correctness proof relies on game semantics."

In 2008, Hague, Murawski, Ong and Serre gave a new proof of this result introducing an extension of higher-order pushdown automata: collapsible pushdown deterministic automata (CPDA) [HMOS08]. Once again, they state the equivalence between HORS and CPDA, and then showed the decidability of the MSO model-checking problem on trees generated by CPDA.

It is interesting to note that in all the papers where a scheme is transformed into an automaton, the automaton acts as an interpreter of the scheme. Its stack contains terms of the HORS, and the automaton simulates the rewriting of those terms. Following that idea, Salvati and Walukiewicz gave in 2011 another proof of Ong's result [SW11]. First they recall the known equivalence between trees generated by HORS and so-called Böhm tree generated by terms of the *λY-calculus*. The λY-calculus is a typed λ-calculus enriched with a fixpoint operator ; furthermore, as an automaton, a *Krivine machine* on a term acts as an interpreter of the term. They later gave a clever transformation between schemes and λ-terms [SW12]. Then they studied the behaviour of a Krivine machine over such terms, and extracted a finite game from the set of configurations of the machine.

In 2006, Aehlig studied a subclass of properties definable in MSO, the properties definable by a *trivial acceptance condition (TAC) automaton* (the definition of a TAC automaton is given in Chapter 2). He has shown the decidability of the TAC automaton model-checking of any HORS [Aeh06, Aeh07]. The originality of his work lies in the fact that he does not use CPDA nor Krivine Machine, nor does he use Ong's techniques and game semantics; instead he defines a type system directly on the terms of the scheme and shows that combining this type system with a fixpoint procedure, one can decide whether the value tree of the scheme is accepted by the TAC automaton or not. In 2009, Kobayashi presented a similar construction in order to develop an actual model-checker for HORS [Kob09, Kob11]. The same year, Kobayashi and Ong extended this idea to prove Ong's result, defining a type system and then a parity game based on this type system, reducing the MSO model-checking problem to the problem of solving a parity game [KO09].

Our work in this thesis is closely related to [Kob09] and [KO09]. Indeed we use some of their constructions, and part of their proofs as building blocks for our results. We justify this choice later on.

1.2.2. Global model-checking, the endogenous approach

As we said earlier, global model-checking was introduced from the beginning of model-checking theory. In particular, it is often used as a tool to design other algorithms (*e.g.* in [LBBO01, CKV06]). In 2004, Schuele and Schneider compared standard model-checking and global model-checking for infinite state systems [SS04]. The same year,

Piterman and Vardi studied the global model-checking problem of two-way automata on regular trees and prefix-recognisable graphs [PV04].

It is important to observe that on infinite structure, the global model-checking problem has one more parameter than the model-checking one. Indeed theoretically a global model-checking algorithm outputs a set of states/nodes/vertices of a given model where a given property holds. If the model is infinite then the output of the algorithm may be infinite too. Therefore one must precise the output type of the algorithm, *i.e.* the set of finite objects that describe infinite sets of states/nodes/vertices of the model.

Carayol et al. have studied games played over the transition graphs of higher-order pushdown automata [CHM$^+$08]. They have shown that when considering higher-order stacks as simple well-bracketed words, the winning region of such game can be described by a finite word automaton (that is effectively constructed in the article). A consequence of this result, due to the equivalence between safe HORS and higher-order pushdown automata, is that the global model-checking of μ-calculus on infinite trees generated by safe schemes is computable. More precisely, they exhibit an algorithm that takes as inputs a scheme and a μ-calculus formula, and outputs a higher-order deterministic pushdown automaton on words that recognises the set of nodes of the tree where the formula holds.

Broadbent and Ong have presented a global model-checking construction for the μ-calculus model-checking of any HORS (safe or unsafe) [BO09]. They give a construction that outputs a nondeterministic CPDA on words that recognises the set of nodes of the tree where the formula holds.

Given an alphabet Σ, a tree T labelled by Σ, and a set S of nodes of the tree, one can encode the pair (S, T) as a tree T_S, obtained from T by marking with an additional label the nodes in S. From this observation, Broadbent, Carayol, Ong and Serre approached the global-model-checking problem with an endogenous point of view [BCOS10]. Given a scheme whose value tree is denoted T, and a node property p (described as an MSO formula), if S is the set of nodes where p holds, then the global-model-checking problem asks to output a generator g of S (*i.e.* a finite object that describes S). Their idea was to output another scheme whose value tree is T_S. It is indeed a generator of S and furthermore its value tree is the same as the one of the initial scheme, enriched with markings on the nodes of S. They refer to this problem as the *MSO-reflection problem*.

They presented an n-EXPTIME algorithm to solve the MSO-reflection problem. The algorithm proceeds by turning the input scheme into a CPDA, constructing another CPDA whose value tree is marked as requested in the problem, and then turning this CPDA into another scheme with the same value tree. This endogenous approach present a strong interest when the schemes are derived from a program. Indeed, the output scheme is an equivalent program, that "knows" at every moment whether the property hold or not. This may then be used for program synthesis, for example by creating a program that acts as another one, but such that some forbidden behaviour is blocked (we develop that idea in Chapter 3).

Following this idea, Carayol and Serre used this endogenous approach to perform witness generation [CS12]. Take a scheme whose value tree T satisfies a property of the form "there exists a set of nodes S in T such that…". A witness of this property would

be a set S that matches the requirement. While the witness generation problem asks to output a finite generator of such S, the endogenous equivalent of this problem, the *MSO-selection problem* requires to output another scheme whose value tree is T_S for some witness S. Again they presented an n-EXPTIME algorithm to solve the MSO-selection problem. This result subsumes the previous one as the reflection problem can be stated as a selection problem.

These results, along with the softwares developed to model-check HORS: TRecS [Kob11] and its sequels [SUK13], TravMC [NOR12], C-SHORe [BCHS13], establish a full framework for verification of higher-order recursion schemes, of functional program, and constitute a great achievement in infinite structures model-checking.

1.3. About scheme transformations

The idea of the endogenous approach is to put the set we generate inside the initial scheme. In [BCOS10, CS12], the scheme is transformed into a CPDA, then this CPDA is transformed into another CPDA which is transformed into a scheme again. As an illustration, take a Caml program P_1, compile it, modify the obtained code without changing the result of the program, and then transform this code into a new Caml program P_2. By the requirement that the result of the transformed assembler program is the same as the initial one, we know that P_1 computes the same value as P_2. However, can we retrieve P_1 from P_2? Or more generally, is there a correspondence between the structure of P_1 and of P_2? A natural question about the reflection and selection, is to know whether they can be obtained without using CPDA.

One of the commonly used formulation of the synthesis problem is the following. Given an algorithmic problem *prob* (*i.e.* a set of *inputs* $\mathcal{I}$, a set of *outputs* $\mathcal{O}$, and a relation $R \subseteq \mathcal{I} \times \mathcal{O}$), and a property π on algorithms (such as "the algorithm must return a value within an hour", or, for practical aspect of synthesis, constraints on programs such as "the program must not send packages through the network more that ten times per second"), does there exists an algorithm $\mathcal{A}$ such that for all $i \in \mathcal{I}$, $(i, \mathcal{A}(i)) \in R$, and $\mathcal{A}$ satisfies π? This problem is of course uncomputable in its full generality, but some cases have been studied and solved (see [Tho09] for an overview about theoretical aspects of synthesis). We focus on the case where the problem is not to find an algorithm for *prob*, but to find one that satisfies π. That is, given an algorithm $\mathcal{A}_0$ that solves problem *prob* but that does not satisfy π; can we construct an algorithm $\mathcal{A}$ that computes the same function as $\mathcal{A}_0$ while satisfying π?

If we shift from theory to practice this problem comes as follows. I designed a program according to a scope statement, but then I realised that it may not comply with all the specifications given. Can I transform this program such that it computes the same value as before and such that it satisfies all the specifications? Furthermore, even though the program is transformed automatically, I would like that the new program consists in a *simple* correction of the previous one, that they look alike or at least that the new program is humanly readable!

Going back to the theory of higher-order recursion scheme, it means that we would

like, when performing an endogenous algorithm (*i.e.* an algorithm that takes a scheme as input and outputs another scheme) such as a reflection or a selection algorithm, that the new scheme has the same structure as the previous one, we would like to be able to revert the process, *i.e.* to recreate the initial scheme from the transformed one. The continuation passing style (CPS) transformation [SS75] is a good example of such transformation.

In this thesis we introduce a notion of shape-preserving scheme transformation that satisfies the informal criteria stated above. Then all along this thesis, we use such transformations to solve different problems related to higher-order recursion schemes. In particular we present two algorithms of simulation of evaluation policies, an MSO-reflection algorithm and an MSO-selection algorithm. All of them involve shape-preserving scheme transformations. The MSO-reflection and selection algorithms are based on Kobayashi and Ong model-checking algorithm, because it does not involve intermediate constructions such as CPDA, Krivine machines or computational trees as in [Ong06]. Therefore it is, in our opinion, more suited to define shape-preserving transformations.

1.4. Structure of the thesis

In Chapter 2, we introduce the basic concepts that will be used throughout this thesis. We start with some generalities, and then introduce the notion of abstract reduction system and term rewriting system that lead to the notion of higher-order recursion schemes. We also introduce the trees we study here (infinite labelled ranked trees), the monadic second-order logic and parity automata. We then present, for the sake of comparison, the collapsible pushdown automata and the λY-calculus, two models equivalent to HORS. We conclude with a state of the art in which we present our contribution and with the description of the shape-preserving transformations.

Chapter 3 is about studying the different semantics of higher-order recursion schemes. Although we do not present new results in the domain of semantics, we extensively use some semantics concepts to solve other problems. We first present a scheme transformation that embeds a finite semantics into a scheme. Then we explain how from an interpretation on the alphabet of the scheme, one can define a semantics of any terms of the scheme. This construction is a standard semantics construction, obtained by computing the fixpoint of a semantics transformation. It consists mainly of a reformulation of the result of [Kob09], obtained by generalising the domain of interpretation. As in [Kob09], this procedure induces the decidability of the TAC-automata model-checking problem. Combined with the embedding construction, it allows to solve the TAC-automata reflection. Then we present the main contribution of this chapter which is a simulation theorem for evaluation policies [Had12]. We construct an algorithm that takes as input a scheme, and two evaluation policies π_1 and π_2, and outputs another scheme such that the value tree of the first scheme under policy π_1 is equal to the value tree of the second scheme under policy π_2. This simulation uses semantics techniques similar to the one introduced in the first part of this chapter.

Chapter 4 presents the algorithms for the MSO-reflection and MSO-selection. These

results are technically involved but we try to give as many intuition as possible. Due to the equivalence between MSO logic and parity tree automaton, we approach these problems from the automata point of view. First we extend the definition of an automaton in order to define runs on terms appearing in the scheme. Then we show that we are able to define a semantics describing the behaviour of the automaton, and then we solve the reflection problem, using the embedding construction of the previous chapter. Finally we solve the selection problem, based on [KO09] results. We describe the parity game to which the model-checking problem is reduced. Then we show that if there is a winning strategy in this game, one can embed this strategy in the scheme, is order to perform selection.

We conclude in Chapter 5. We recall the main contributions, and give several open questions and thoughts about possible directions in which this work can be extended.

2. Preliminaries

2.1. Introduction 23
2.2. General background 27
2.2.1. Words 27
2.2.2. Relations, partial orders 28
2.2.3. Abstract reduction systems 29
2.2.4. Trees 31
2.2.5. Logic and automata 33
2.2.6. Term rewriting systems 39
2.3. Higher order recursion schemes 43
2.3.1. Syntax 43
2.3.2. Semantics 46
2.3.3. Main algorithmic problems 51
2.4. Equivalent models 53
2.4.1. λY-Calculus 53
2.4.2. Collapsible pushdown automata 58
2.5. State of the art 65
2.6. Scheme transformations 67

2.1. Introduction

Since our work deals with higher order recursion schemes (HORS), we introduce here the basic formalisms required to describe both this model and the related decision and synthesis problems. One way to address these problems consists in translating HORS into an equivalent model and designing appropriate procedures for this new model. Thus we also present two equivalent models: λY-calculus and collapsible pushdown automata (CPDA). Finally, in order to compare our work to the state of the art, we summarise the main contributions related to HORS, and briefly introduce our contributions that will be developed in the subsequent chapters.

General background. Reduction systems [BN98] are usually introduced in order to give semantics for dynamical systems. When such dynamical systems correspond to (functional) programs, a particularly relevant property of reduction systems is confluence which ensure that evaluation choices do not impact the computed value. We introduce finite and infinite trees associated with the semantics of a reduction system and we show that there exists a *value tree* corresponding to the maximal behaviour of a confluent reduction system. Properties of trees may be specified in different ways. Here we only consider two equivalent specifications: monadic second order logic (MSO), and parity tree automata [Rab69]. Among such reduction systems, we are interested in term

rewriting systems (TRS), where the dynamic is given by a set of rules that inside a model transform a term into another.

Higher Order Recursion Schemes. The specific features of HORS with respect to term rewriting systems are the following ones: it handles typed terms, and the application of rules is deterministic. Types of an HORS corresponds to the standard notion of *simple types*: a type is either the ground type o, or a function type that maps a type to another $\tau \to \tau'$. Then the terms are built from an alphabet where characters are typed, and the terms must respect the type requirement. More precisely the alphabet is split into terminal and non-terminal symbols. With every non terminal symbol is associated a single rule that transforms a term whose head is this symbol, into another term. From the uniqueness of the rule, one can prove that an HORS is confluent, furthermore its value tree is only labelled by terminal symbols. Evaluation policies restrict which rules can be applied in a term; they have been introduced to mimic the choice of an interpreter for functional programs. The main evaluation policies are OI where the most external symbols are transformed first, and IO where the most internal symbols are transformed first.

Example 2.1.1. We define the HORS $\mathcal{G}$ on the alphabet $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, S, F, H, D\}$. The symbols $\mathtt{c}$ and S have arity 0, their type is written o. The symbols $\mathtt{a}$, $\mathtt{b}$, F and H expect one argument of type o, their type is written $o \to o$ (*i.e.* "with one argument of type o, I can form a term of type o"). The symbol D has arity 2, it expects a first argument of type $o \to o$ and a second argument of type o, its type is written $(o \to o) \to o \to o$. The terminal symbols are $\mathtt{a}$, $\mathtt{b}$ and $\mathtt{c}$, the non-terminal symbols are the other.

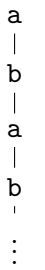
Given the variables x of type o, and φ of type $o \to o$ we define the following set of rewrite rules associated with the non-terminal symbols.

$$\begin{array}{lcl} S & \to & F\ \mathtt{c}, \\ F\ x & \to & \mathtt{a}\ (D\ H\ x), \\ H\ x & \to & \mathtt{b}\ (D\ F\ x), \\ D\ \varphi\ x & \to & \varphi\ (\varphi\ x). \end{array}$$

The initial non-terminal symbol is S. A derivation of the HORS is a sequence of terms obtained from S by rewritting the non-terminal symbols. For example here is the beggining of a derivation obtained with an *outermost-innermost (OI)* evaluation policy, *i.e.* we always rewrite the non-terminal symbols closest to the head of the term.

$$\begin{array}{l} S \to F\ \mathtt{c} \to \mathtt{a}\ (D\ H\ \mathtt{c}) \to \mathtt{a}\ (H\ (H\ \mathtt{c})) \to \mathtt{a}\ (\mathtt{b}\ (D\ F\ (H\ \mathtt{c})) \to \\ \qquad\qquad \mathtt{a}\ (\mathtt{b}\ (F\ (F\ (H\ \mathtt{c})))) \to \mathtt{a}\ (\mathtt{b}\ (\mathtt{a}\ (D\ H\ (F(H\ \mathtt{c}))))) \to \cdots \end{array}$$

The limit-tree of this derivation contains a unique branch alternating $\mathtt{a}$'s and $\mathtt{b}$'s, as described below.



Finally, we describe below the beginning of an *innermost-outermost (IO)* derivation, where the non-terminal symbols rewritten are the most internal ones.

$$S \to F\ \mathtt{c} \to \mathtt{a}\ (D\ H\ \mathtt{c}) \to \mathtt{a}\ (H\ (H\ \mathtt{c})) \to \mathtt{a}\ (H\ (\mathtt{b}\ (D\ F\ \mathtt{c}))) \to \\ \mathtt{a}\ (H\ (\mathtt{b}\ (F\ (F\ \mathtt{c})))) \to \mathtt{a}\ (H\ (\mathtt{b}\ (F\ (\mathtt{a}\ (D\ H\ \mathtt{c}))))) \to \cdots$$

Equivalent models. The λY-calculus is an extension of typed λ-calculus, where the extension is obtained by introduction of structured recursion represented by a recursive operator Y. Its syntax is very close to the one of HORS, the main difference being that the recursion is *anonymous* in λY-calculus. More precisely, terms do not contain explicit recursive symbols (the non-terminal ones), but instead use the operator Y to specify recursion.

Example 2.1.2. Terms of the λY-calculus are described using terminal symbols, variables, the symbol λ to denote a function, and the symbol Y to denote a recursion. For example, take the term $M = \big(\lambda\varphi.\ Y\ \lambda x.\ \mathtt{a}\ (\varphi\ x)\big)\ \mathtt{b}$. The term may be rewritten using two rules:

- the *β-reduction*, that states that a term $(\lambda x.\ N_1)\ N_2$ can be rewritten into $N_{1[x \mapsto N_2]}$, *i.e.* N_1 in which x is replaced by N_2,
- and the *δ-reduction*, that states that a term $Y\ N$ can be rewritten into $N\ (Y\ N)$.

We describe below the beginning of a possible derivation from M.

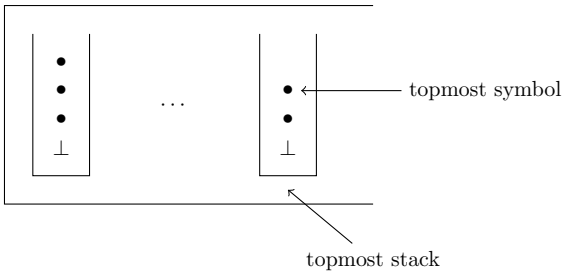
$$\big(\lambda\varphi.\ Y\ \lambda x.\ \mathtt{a}\ (\varphi\ x)\big)\ \mathtt{b} \to_\beta Y\ \lambda x.\ \mathtt{a}\ (\mathtt{b}\ x) \to_\delta (\lambda x.\ \mathtt{a}\ (\mathtt{b}\ x))\ (Y\ \lambda x.\ \mathtt{a}\ (\mathtt{b}\ x)) \to_\beta \\ \mathtt{a}\ (\mathtt{b}\ (Y\ \lambda x.\ \mathtt{a}\ (\mathtt{b}\ x))) \to_\delta \mathtt{a}\ (\mathtt{b}\ ((\lambda x.\ \mathtt{a}\ (\mathtt{b}\ x))\ (Y\ \lambda x.\ \mathtt{a}\ (\mathtt{b}\ x)))) \to_\beta \\ \mathtt{a}\ (\mathtt{b}\ (\mathtt{a}\ (\mathtt{b}\ (Y\ \lambda x.\ \mathtt{a}\ (\mathtt{b}\ x))))) \to_\delta \cdots$$

The value tree of this term is the same as the one of the HORS of example 2.1.1 it is the tree that contains a unique infinite branch alternating a's and b's.

On the other hand, CPDA are deterministic stack automata that produce trees. Stacks of a CPDA have an order: an order-1 stack is a stack of symbols, and an order-$n+1$ stack is a stack of order-n stacks. In addition, a symbol may be enlarged by a link that points into the stack. Accordingly to this elaborated structure, the operations on stacks include new features like copying a substack, collapsing a stack, etc.

Example 2.1.3. We define a simple order-2 CPDA whose states are q_0, q_1, q_2, whose stack symbols are: $\perp$ that represents the bottom of a stack, and $\bullet$, and whose terminal symbols are `a` and `b`, both of arity 1. It has order 2, which means that it handles a stack of stacks. Transitions associate with a state and a stack symbol a tuple that contains another state, a terminal symbol (or the symbol ε to denote a non-productive transition), and a stack operation. For the sake of simplicity, we do note use the *collapse operation*, more detailed examples can be found in section 2.4.2.

The stack of stacks handled by the automaton is graphically represented as follows.

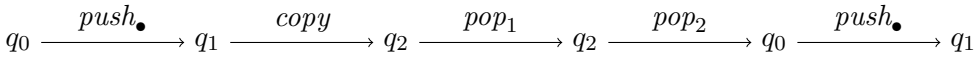


We give below the transitions of the CPDA. The operation $push_\bullet$ pushes the symbol $\bullet$ in the topmost stack, the operation *copy* copies the topmost stack, the operation pop_1 remove the topmost symbol of the topmost stack, and the operation pop_2 removes the topmost stack.

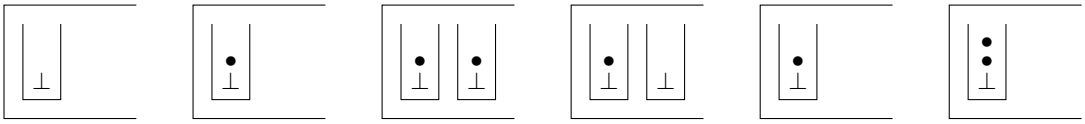
$$\begin{array}{lcl} q_0, \perp & \rightarrow & (q_1, \mathtt{b}, push_\bullet) \\ q_0, \bullet & \rightarrow & (q_1, \mathtt{b}, push_\bullet) \\ q_1, \bullet & \rightarrow & (q_2, \varepsilon, copy) \\ q_2, \bullet & \rightarrow & (q_2, \mathtt{a}, pop_1) \\ q_2, \perp & \rightarrow & (q_0, \varepsilon, pop_2) \end{array}$$

The automaton constructs step by step a branch of terminal symbols, by applying the transitions starting from state q_0 with an initial stack (described below). At each step it choses the transition according to the current state and to the topmost symbol of the topmost stack. We represent the (beginning of the) evolution of the state of the CPDA, of its stack, and of the tree it constructs. We use the symbol $\perp$ to denote the part of the tree under construction.

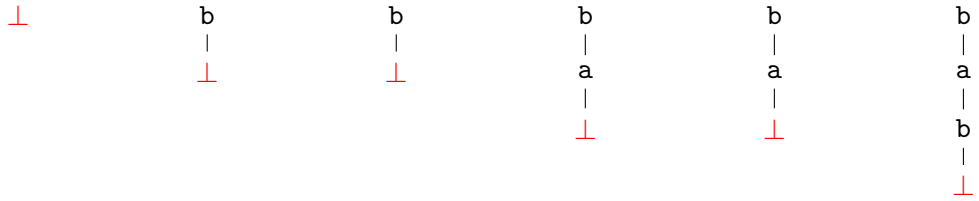
States/operations:



Stack:



Tree:



The tree constructed by the CPDA is the infinite branch whose labelling is the word $\mathtt{baba}^2\mathtt{ba}^3\mathtt{b}\cdots$.

State of the art. Finally, we give an overview of the results of the domain, including the present work in order to position our results with respect to the other contributions.

2.2. General background

2.2.1. Words

We call ***alphabet*** a (finite or infinite) set whose elements are seen as symbols, formally there is no differences between a set and an alphabet. We call finite words (resp. ω-words) of an alphabet Σ the finite sequences (resp. infinite countable sequences) of elements of Σ. We write Σ^* for the set of finite words, and the empty word is written ε. Given $u, v \in \Sigma^*$ we write $u \cdot v$ or simply uv the ***concatenation*** of u and v, and we say that u is a ***prefix*** of v if there exists $u' \in \Sigma^*$ such that $v = u \cdot u'$. A subset S of Σ^* is ***prefix-closed*** if for all $u \in S$, for all prefix $v \in \Sigma^*$ of u, $v \in S$. We write Σ^ω the set of infinite words of elements of Σ and given $u \in \Sigma^*$ and $v \in \Sigma^\omega$, we write $u \cdot v \in \Sigma^\omega$ the concatenation of u and v. We write Σ^∞ the union of Σ^* and Σ^ω.

A finite word automaton is a tuple $\langle \Sigma, Q, q_0, Q_F, \delta \rangle$ where Σ is a finite alphabet, Q is a finite set of ***states***, $q_0 \in Q$ is the ***initial state***, $Q_F \subseteq Q$ is the set of ***accepting states***, $\delta \subseteq Q \times \Sigma \times Q$ is the ***transition relation***. We write $q \xrightarrow{a} q' \in \delta$ instead of (q, a, q'). The ***language of the automaton*** is the set of finite words $u = a_1 \cdots a_k$ such that there exists a sequence of states $q_1, \ldots, q_k$ such that for all $i < k$, $q_i \xrightarrow{a_i} q_{i+1} \in \delta$ and $q_k \in Q_F$. For all automaton one can construct another automaton with the same language that is ***deterministic and complete*** *i.e.* for all $q \in Q$ and $a \in \Sigma$ there exists exactly one state q' such that $q \xrightarrow{a} q' \in \delta$.

2.2.2. Relations, partial orders

Let X, Y be two sets and $R \subseteq X \times Y$ be a relation between X and Y. We say that R is a ***partial function***, if for all $x \in X$ there exist at most one y such that $(x, y) \in R$. We say that it is a ***total function*** if for all $x \in X$ there exist exactly one y such that $(x, y) \in R$. In the following we may write xRy for $(x, y) \in R$, and $R(x) = y$ if R is a function.

Let X be a set and $R \subseteq X \times X$ be a relation over X. We say that R is:

- ***reflexive***, if for all $x \in X$, xRx,
- ***transitive***, if for all $x, y, z \in X$, if xRy and yRz, then xRz,
- ***symmetric***, if for all $x, y \in X$, if xRy then yRx,
- ***antisymmetric***, if for all $x, y \in X$, if xRy and yRx, then $x = y$.

The relation R is a ***(partial) order*** if it is reflexive, transitive and antisymmetric. It is an ***equivalence relation*** if it is reflexive, transitive, and symmetric. The ***reverse of*** R, is the relation R^{-1} defined by $xR^{-1}y$ if and only if yRx. Given two relations R and R' the ***composition*** of R and R', written $R \cdot R'$, is defined by $xR \cdot R'y$ if and only if there exists $z \in X$ such that xRz and $zR'y$. For all $n > 1$, the relation R^n is inductively defined by $R^1 = R$ and $R^{n+1} = R^n \cdot R$. The ***identity relation*** Id is defined by $(x, y) \in Id$ if and only if $x = y$. The ***closure by reflexivity*** of R is the relation $R \cup Id$. The ***closure by transitivity*** of R is the relation $R^+ = \bigcup_{n>1} R^n$, and the ***closure by reflexivity and transitivity*** of R is the relation $R^* = Id \cup R^+$. Remark that given a relation R, $R \cup Id$ is reflexive, R^+ is transitive, and R^* is reflexive and transitive.

Let $\leq$ be an order over a set X. Given a subset Y of X, and an element $x \in X$, then

- x is an ***upper bound*** (resp. ***lower bound***) of Y if for all $y \in Y$, $y \sqsubseteq x$ (resp. $x \sqsubseteq y$),
- x is a ***supremum*** (resp. ***infimum***) of Y if it is an upper bound (resp. lower bound) of X' and for all upper bound (resp. lower bound) $y \in X$ of Y, $x \sqsubseteq y$ (resp. $y \sqsubseteq x$),
- x is a ***maximum*** (resp. ***minimum***) of Y if it is a supremum (resp. infimum) and $x \in X'$.

Remark that any subset Y has at most one supremum (resp. infimum).

Given a set X, we write 2^X or $\mathcal{P}(X)$ the set of subsets of X.

Given an equivalence relation R, on X, the ***quotient of*** X ***by*** R, is the subset X/R of 2^X such that for all $Y \subseteq X$, $Y \in X/R$ if and only if for all $x, y \in Y$, $(x, y) \in R$ and for all z such that $(x, z) \in R$, $z \in Y$.

A ***graph*** is a tuple $G = \langle V, E \rangle$ where V is a finite set of ***vertices*** and $E \subseteq V \times V$ is called the set of ***edges*** of G. We say that G contains a ***cycle*** if there exists $v_1, \ldots, v_k \in V$ for some $k > 1$ such that for all $i < k$, $(v_i, v_{i+1}) \in E$, and such that $v_k = v_1$. A graph is ***acyclic*** if it contains no cycles.

2.2.3. Abstract reduction systems

The main topic of our work is the analysis of the evaluation of a higher order recursion scheme (HORS). Such an evaluation process can be viewed as a transition system also called in our context abstract reduction system.

Definition 2.2.1 (Abstract reduction system)**.**

An ***abstract reduction system (ARS)*** is a pair $\mathcal{S} = \langle X, \to \rangle$ where:

- X is a set,
- $\to \subseteq X \times X$, is the ***reduction relation*** of $\mathcal{S}$.

An ARS is ***finitely branching*** if for all $x \in X$, the set of ***successors*** of x, $\{y \mid x \to y\}$ is finite.

For example the set of all natural numbers along with the successor relation is an ARS $\langle \mathbb{N}, \to \rangle$ defined as $x \to y$ if $y = x + 1$. Recall that we write $\to^*$ the reflexive and transitive closure of $\to$. In this example $x \to^* y$ if and only if $x \leq y$.

In higher order recursion schemes, the set X will be the set of terms, and the scheme will give rules that explains how to transform a term into another yielding the reduction relation.

A *derivation* of an ARS is a finite or infinite sequence of states related by the reduction relation.

Definition 2.2.2 (Derivation)**.**

A ***derivation*** $x_1 \to x_2 \to \cdots$ is a possibly infinite sequence $x_1, x_2, \ldots$ of elements of X such that for all $i \geq 1$, $x_i \to x_{i+1}$.
A derivation $x_1 \to x_2 \to \cdots$ is ***maximal*** if either it is infinite, or it is finite and its last element has no successor.

An ARS is ***deterministic*** if for all $x \in X$, its set of successors is either a singleton or the empty set. While determinism is too strong of a requirement for ARS, a weaker but useful requirement is the confluence property.

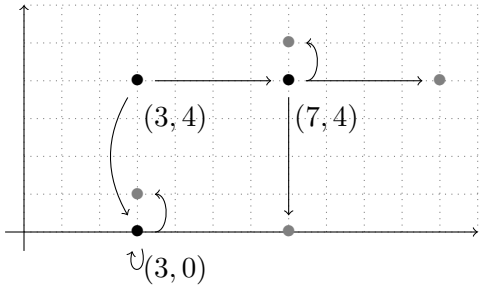


Figure 2.1.: The relation $\to$ described in example 2.2.1 is confluent but does not fulfill the diamond property.

Definition 2.2.3 (Confluence)**.**

We say that $\to$ is ***confluent*** if for all x, x_1, x_2 such that $x \to^* x_1$ and $x \to^* x_2$ there exists x' such that $x_1 \to^* x'$ and $x_2 \to^* x'$.

We now describe a sufficient condition (Proposition **2.2.1**) for an ARS to be confluent, that we will apply later on higher order recursion schemes.

Definition 2.2.4 (Diamond property)**.**

We say that a relation $\triangleright$ satisfies the ***diamond property*** if for all x, x_1, x_2 such that $x \triangleright x_1$ and $x \triangleright x_2$ there exists x' such that $x_1 \triangleright x'$ and $x_2 \triangleright x'$.

In particular, a relation is confluent if and only if its reflexive and transitive closure satisfies the diamond property. A simple induction allows to state that if a relation $\triangleright$ satisfies the diamond property then it is confluent.

Example 2.2.1. Note that the converse is not true in general, for example take the ARS $\langle \mathbb{N}^2, \to \rangle$ with $(x, y) \to (x + y, y)$, $(x, y) \to (x, 0)$ and $(x, y) \to (x, y + 1)$ for all $(x, y) \in \mathbb{N}^2$. One can easily show that $(x, y) \to^* (x', y')$ if and only if $x' \geq x$ therefore $\to$ is confluent. However $(3, 4) \to (7, 4)$, $(3, 4) \to (3, 0)$ but there does not exist (x', y') such that $(3, 0) \to (x', y')$ and $(7, 4) \to (x', y')$, as illustrated in Figure 2.1.

Proposition 2.2.1.

Given an ARS $\mathcal{S} = \langle X, \to \rangle$, if there exists a relation $\triangleright \subseteq X \times X$ which satisfies the diamond property and such that $\to \subseteq \triangleright \subseteq \to^*$ then $\mathcal{S}$ is confluent.

Example 2.2.1 continued. Let the relation $\triangleright$ on $\mathbb{N}^2$ be defined by $(x, y) \triangleright (x + k, y)$ for all $k \geq 0$, $(x, y) \triangleright (x, 0)$ and $(x, y) \triangleright (x, y + 1)$. Then we have $\to \subseteq \triangleright \subseteq \to^*$, and furthermore $\triangleright$ satisfies the diamond property.

2.2.4. Trees

In the context of HORS, a tree is built and updated along an evaluation of a scheme, corresponding at any time to a partial evaluation. Henceforth, the analysis of the behaviour of the associated reduction system requires to formally define trees and how reduction systems produce trees. We are mainly interested in finite or infinite labeled trees where the label of a node determines the number of its successors. Furthermore, the special character $\perp$ will be implicitly used to truncate trees.

Definition 2.2.5 (Ranked alphabet)**.**

A ***ranked alphabet*** Σ is a finite set of symbols such that each element has a finite arity.

By convention $\perp$ has arity 0. In the sequel when we introduce a ranked alphabet Σ we assume that it does not contains $\perp$, and we denote $\Sigma_\perp = \Sigma \uplus \{\perp\}$. We use the notation $\mathit{arity}(\alpha)$ do denote the arity of α.

Definition 2.2.6 (Tree)**.**

Let Σ be a ranked alphabet, and m the maximum arity over all the symbols, a ***tree*** T over Σ is a partial mapping $T : \{0, \ldots, m-1\}^* \to \Sigma_\perp$, which fulfills:

- the domain of T, denoted dom^T, is prefix-closed,
- for all $u \in \mathit{dom}^T$, $\{j \mid u \cdot j \in \mathit{dom}^t\} = \{0, \ldots, \mathit{arity}(T(u)) - 1\}$ (in particular, it is the empty set when $\mathit{arity}(T(u)) = 0$).

We denote by $\mathbf{Trees}(\Sigma)$ the set of trees over Σ.

Fix a tree T. Elements of dom^T are called ***nodes***. Given a node u the elements of $\{u \cdot j \in dom^T \mid j \leq m-1\}$ are called the ***successors*** of u. A ***branch*** is a potentially infinite sequence of nodes $u_0, u_1, \ldots$ such that $u_0 = \varepsilon$ and u_{i+1} is a successor of u_i for all i and which is finite if and only if the last node has no successor. Given a branch b we write $T(b)$ the possibly infinite word $T(u_0) \cdot T(u_1) \cdots \in \Sigma_\perp^\infty$. We say that T is finite if dom^T is finite. Given $u \in dom^T$, we define the subtree $T[u]$ as the tree such that $dom^{T[u]} = \{v \mid u \cdot v \in dom^T\}$ and $T[u](v) = T(u \cdot v)$ for all $v \in dom^{T[u]}$.

Definition 2.2.7 (Extended tree)**.**

Let T, T' be two trees, we say that T' ***extends*** T, written $T \sqsubseteq T'$ if $dom^T \subseteq dom^{T'}$ and for all $u \in dom^T$, either $T(u) = T'(u)$ or $T(u) = \perp$.

Observe that $\sqsubseteq$ is a partial order over trees. This order has a useful property formalised by the next proposition.

Proposition 2.2.2 (Limit tree)**.**

Given a possibly infinite sequence of trees $T_0 \sqsubseteq T_1 \sqsubseteq \cdots$, the set $\{T_i \mid i \geq 0\}$ has a supremum, called the ***limit tree*** of the sequence.

Proof. We define the domain of the tree T_∞ as $dom^{T_\infty} = \bigcup_n dom^{T_n}$. Observe that given $u \in dom^{T_\infty}$, there exist j and $a \in \Sigma_\perp$ such that for all $j' \geq j$, $u \in dom^{T_{j'}}$ and $T_{j'}(u) = a$. So we define $T_\infty(u) = a$. □

Actually, the partially ordered set of trees satisfies a stronger property, it is a *complete partial order*, *i.e.* all directed sets have a supremum.

Definition 2.2.8 (Directed set, CPO)**.**

Let $(X, \leq)$ be a partially ordered set. A subset Y of X is ***directed*** if for all $y_1, y_2 \in Y$, there exists $y \in Y$ such that $y_1 \leq y$ and $y_2 \leq y$.
Furthermore, $(X, \leq)$ is a ***complete partial order (CPO)*** if all directed subset Y of X has a supremum in X and X has a minimum.

For both theoretical and practical reasons, the labels may be enlarged with additional information (*e.g.* collected by some device exploring the tree). This requires to build new ranked alphabets: let Σ be a ranked alphabet, and X be a set, then $\Sigma_\perp \times X$ is a ranked alphabet, defined by $arity(a, x) = arity(a)$. In the sequel, a^x denotes (a, x).

Definition 2.2.9 (X-Annotation)**.**

Let X be a set, and T be a tree over Σ, and T' be a tree over $\Sigma_\perp \times X$. Then T' is an ***X-annotation*** of T, if $dom^T = dom^{T'}$, and for all $u \in dom^T$, $T'(u) = T(u)^x$ for some $x \in X$.
For any tree T' on the ranked alphabet $\Sigma_\perp \times X$, we define $Unlab(T')$ as the tree on Σ obtained by changing all labels a^x of T' into a, *i.e.* the tree obtained by removing the annotated part of the tree.

We consider abstract reduction systems as generators of trees. For that, with some elements of the set X we associate trees, and when an element $x \in X$ is transformed, the associated tree is extended.

Definition 2.2.10 (Decorated abstract reduction system)**.**

Let Σ be an ranked alphabet. A ***decorated abstract reduction system*** is a tuple $\mathcal{S} = \langle X, \to, f\rangle$ with $\langle X, \to\rangle$ an ARS, and f a partial mapping from X to $\mathbf{Trees}(\Sigma)$ such that:

for any $x, y \in X$, if $x \to y$ and $f(x)$ is defined then $f(y)$ is defined and $f(x) \sqsubseteq f(y)$.

Given $x_0 \in dom(f)$, we define the ***limit tree*** of a derivation $x_0 \to x_1 \to x_2 \to \cdots$, as the limit of $f(x_0) \sqsubseteq f(x_1) \sqsubseteq f(x_2) \sqsubseteq \cdots$.

If the ARS is deterministic, then from any $x \in dom(f)$ there exists a unique maximal derivation. Therefore with any $x \in dom(f)$ we can associate a unique limit tree. Fortunately this property is also fulfilled by confluent ARS.

Proposition 2.2.3 (Value tree of an element)**.**

Let $\mathcal{S} = \langle X, \to, f\rangle$ be a confluent decorated ARS, and $x \in dom(f)$. Define $\lim(x) \subseteq \mathbf{Trees}(\Sigma)$ as the set of trees T such that T is the limit tree of a derivation starting from x.
Then $\lim(x)$ has a maximum denoted $\|x\|$ called the ***value tree*** of x.

2.2.5. Logic and automata

As discussed above, we want to analyse the behaviour of a decorated reduction system. In order to formalise properties of trees, we introduce monadic second order (MSO)

logic and tree automata. MSO logic is well suited to describe tree properties: a sentence (*i.e.* a closed formula) reasons about a tree, a formula with one free variable expresses a property on nodes or on sets of nodes in the tree, depending of the order of the variable.

Definition 2.2.11 (MSO Formulas)**.**

Let Σ be a finite ranked alphabet whose maximum arity is m, and let $\mathcal{V}_1$ and $\mathcal{V}_2$ be respectively the infinite set of ***first order variables*** and ***second order variables***. The set of ***monadic second order formulas*** MSO[$\Sigma_\bot$] (or simply MSO when Σ is clear from the context) is inductively defined by:

$\texttt{true} \in \text{MSO}$, $\quad a(x) \in \text{MSO}$ for $a \in \Sigma_\bot$ and $x \in \mathcal{V}_1$,

$Son_i(x,y) \in \text{MSO}$ for $i \leq m$, $x, y \in \mathcal{V}_1$, $\quad (x \in X) \in \text{MSO}$ for $x \in \mathcal{V}_1$, $X \in \mathcal{V}_2$,

$\exists x.\varphi \in \text{MSO}$ for $x \in \mathcal{V}_1$, $\varphi \in \text{MSO}$, $\quad \exists X.\varphi \in \text{MSO}$ for $X \in \mathcal{V}_2$, $\varphi \in \text{MSO}$,

$\neg\varphi \in \text{MSO}$ for $\varphi \in \text{MSO}$, $\quad \varphi \vee \psi \in \text{MSO}$ for $\varphi, \psi \in \text{MSO}$.

In the following,we write $x, y, z, \ldots$ for first order variables and $X, Y, Z, \ldots$ for second order variables.

We add the other usual symbols as syntactic shorthand:

$$\forall x.\varphi \;:=\; \neg\exists x.(\neg\varphi), \qquad \forall X.\varphi \;:=\; \neg\exists X.(\neg\varphi),$$

$$\varphi \wedge \psi \;:=\; \neg((\neg\varphi) \vee (\neg\psi)), \qquad \varphi \Rightarrow \psi \;:=\; (\neg\varphi) \vee \psi,$$

$$\texttt{false} \;:=\; \neg\texttt{true}, \qquad \varphi \Leftrightarrow \psi \;:=\; \varphi \Rightarrow \psi \wedge \psi \Rightarrow \varphi,$$

$$x = y \;:=\; \forall X \; x \in X \Leftrightarrow y \in X.$$

Definition 2.2.12 (Free variables)**.**

Given a formula φ the set $FV(\varphi) \subseteq \mathcal{V}_1 \uplus \mathcal{V}_2$ of free variables of φ is inductively defined by:

$$FV(\texttt{true}) = \emptyset, \qquad FV(a(x)) = \{x\},$$

$$FV(Son_i(x,y)) = \{x,y\}, \qquad FV(x \in X) = \{x, X\},$$

$$FV(\exists x.\varphi) = FV(\varphi)\backslash\{x\}, \qquad FV(\exists X.\varphi) = FV(\varphi)\backslash\{X\},$$

$$FV(\neg\varphi) = FV(\varphi), \qquad FV(\varphi \vee \psi) = FV(\varphi) \cup FV(\psi).$$

Given a formula φ, let $FV_1(\varphi) = FV(\varphi) \cap \mathcal{V}_1$ and $FV_2(\varphi) = FV(\varphi) \cap \mathcal{V}_2$ be the sets of ***first order free variables*** and ***second order free variables*** rspectively.

Definition 2.2.13 (Semantics)**.**

Let $T : dom^T \to \Sigma$ be a tree on Σ, φ be an MSO formula, and $\rho : \mathcal{V}_1 \uplus \mathcal{V}_2 \to dom^T \cup \mathcal{P}(dom^T)$ be a partial mapping such that $FV(\varphi) \subseteq dom(\rho)$, $\rho(FV_1(\varphi)) \subseteq dom^T$, and $\rho(FV_2(\varphi)) \subseteq \mathcal{P}(dom^T)$.
We associate with the tuple (T, ρ, φ) a Boolean value $t \models_\rho \varphi$, inductively defined by:

- $T \models_\rho$ `true` is true,
- $T \models_\rho a(x)$ if and only if $T(\rho(x)) = a$,
- $T \models_\rho Son_i(x, y)$ if and only if $\rho(y) = \rho(x) \cdot i$,
- $T \models_\rho x \in X$ if and only if $\rho(x) \in \rho(X)$,
- $T \models_\rho \exists x.\varphi$ if and only if there exists $u \in dom^T$ such that $T \models_{\rho \cup \{x \mapsto u\}} \varphi$,
- $T \models_\rho \exists X.\varphi$ if and only if there exists $S \subseteq dom^T$ such that $t \models_{\rho \cup \{X \mapsto S\}} \varphi$,
- $T \models_\rho \neg\varphi$ if and only if $T \models_\rho \varphi$ is false,
- $T \models_\rho \varphi \vee \psi$ if and only if $T \models_\rho \varphi$ or $T \models_\rho \psi$,

where $\rho \cup \{x \mapsto u\}$ is the extension of ρ defined by $dom(\rho \cup \{x \mapsto u\}) = dom(\rho) \cup x$, $\rho \cup \{x \mapsto u\}(x) = u$ and for all $y \neq x \in dom(\rho)$, $\rho \cup \{y \mapsto u\} = \rho(y)$.

Example 2.2.2. Here are some examples of properties one can describe in MSO.

- A node v is a son of the node u in the tree t if $t \models_{\{x \mapsto u, y \mapsto v\}} Son(x, y) = \bigvee_i Son_i(x, y)$,
- u is the root of the tree t if $t \models_{\{x \mapsto u\}} root(x) = \neg\exists y\ Son\ (y, x)$,
- Given a node u and a set S, then the father of u is in S if $t \models_{\{z \mapsto u, X \mapsto S\}} \psi_1[z, X]$ with:
$$\psi_1[z, X] = \exists y\ Son(y, z) \wedge y \in X,$$
and u has exactly one son in S if $t \models_{\{z \mapsto u, X \mapsto S\}} \psi_2(z, X)$ with:
$$\psi_2[z, X] = \exists x\ Son(z, x) \wedge x \in X \wedge \forall y\ Son(z, y) \Rightarrow (y = x).$$
- Given two nodes u and v, then u is an ancestor in the tree if $t \models_{\{x \mapsto u, y \mapsto v\}}$ "$x \sqsubseteq y$" with:
$$\text{“}x \sqsubseteq y\text{”} = \exists X\ x \in X \wedge y \in X \wedge \forall z\ z \in X \Rightarrow (\psi_1[z, X] \Leftrightarrow \neg equals(z, x)) \wedge (\psi_2[z, X] \Leftrightarrow \neg equals(z, y)).$$

- Given a set S, S is an infinite branch if $t \vdash_{\{X \mapsto S\}} \mathit{Branch}(X)$ with:

$$\mathit{Branch}(X) = (\forall x\ \mathit{root}(x) \to x \in X) \wedge (\forall x\ x \in X \Rightarrow \exists y\ (y \in X \wedge \mathit{Son}(x, y))) \wedge (\forall x \forall y\ (x \in X \wedge y \in X) \Rightarrow (x \leq y \vee y \leq x)).$$

In the following, we will either precise explicitly the free variables of a formula when we refer to it, in the following syntax: φ is written $\varphi[x_1, \ldots, x_k, X_1, \ldots, X_\ell]$ if $FV(\varphi) = \{x_1, \ldots, x_k, X_1, \ldots, X_\ell\}$, or we will write $\varphi[\ldots]$ to denote a formula that may contain some free variables. In particular when we write φ, it means that $FV(\varphi) = \emptyset$.

Once defined a property with one first order free variable, one can be interested to know if it holds for a given node in a given tree. A more global information consists in determining the set of all nodes in the tree that fulfils this property. This leads to the notion of marking and reflection.

Definition 2.2.14 (Marking)**.**

Let T be a tree and $S \subseteq \mathit{dom}^T$ be a set of nodes. The ***S*-marking** of t is the $\{0,1\}$-annotation T_S of T defined by $T_S(u) = t(u)^1$ for all $u \in S$ and $T_S(u) = t(u)^0$ for all $u \notin S$

Definition 2.2.15 (Reflection)**.**

Let T be a tree and $\varphi[x]$ be a formula where x is a first order free variable, the $\varphi[x]$***-reflection*** on T is the S-marking of T where S is the set of nodes u such that $T \models_{\{x \mapsto u\}} \varphi[x]$. It is denoted $T_{\varphi[x]}$.

Using a second order free variable, a more elaborated information can be obtained. Such information is called a selector.

Definition 2.2.16 (Selector)**.**

Let T be a tree and $\varphi[X]$ be a formula where X is a second order free variable, such that $T \models \exists X \varphi[X]$[a]. A $\varphi[X]$***-selector*** is an S-marking T_S of T for some set S that fulfils $T \models_{\{X \mapsto S\}} \varphi[X]$.

[a] If this restriction is not satisfied, the definition is irrelevant.

Let $\varphi[x]$ be a formula where x is a first order variable. Let us define $\psi[X] = \forall x\ x \in X \Leftrightarrow \varphi[x]$. For all tree T, there exists a unique $\psi[X]$-selector on T, which is equal to $T_{\varphi[x]}$. This establishes that the notion of selector is more expressive than reflection.

Parity tree automata are another model to describe tree properties. An automaton can be viewed as a finite state machine running down over a tree from the root, labelling the tree with states. Then the tree is said to be accepted by the automaton if the labelling satisfies a given *acceptance condition*.

Definition 2.2.17 (Parity automaton)**.**

A ***non-deterministic (max) parity automaton*** is a tuple $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \Omega \rangle$ where:

- Σ a ranked alphabet,
- Q a finite ***set of states*** with $q_0 \in Q$ the ***initial state***,
- $\delta \subseteq \{q \xrightarrow{a} (q_1, \ldots, q_{arity(a)}) \mid a \in \Sigma_\perp, q, q_1, \ldots, q_{arity(a)} \in Q\}$ is a ***transition relation***,
- $\Omega : Q \to \mathbb{N}$ the ***colouring function***.

To simplify the notation if $a \in \Sigma$ has arity 0, we write $a(q)$ the transition $q \xrightarrow{a}$; in addition, parity automata will simply be called automata.

Definition 2.2.18 (Run)**.**

Let $\mathcal{A}$ be an automaton. Given an infinite tree T on Σ a ***run*** r of $\mathcal{A}$ on t is defined as a Q-annotation r of T, such that $r(\varepsilon) = T(\varepsilon)^{q_0}$ and for all $u \in dom^T$ then if $r(u) = a^q$ there exists $q \xrightarrow{a} q_1, \ldots, q_k \in \delta$ such that for all i, $r(u \cdot i) = T(u \cdot i)^{q_i}$.

Definition 2.2.19 (Acceptance condition)**.**

Let $\mathcal{A}$ be an automaton. Let T be a tree and r be a run over T. Then r is ***accepting*** if for every infinite branch b of r, if $r(b) = a_0^{q_0}, a_1^{q_1}, \ldots$, the greatest colour seen infinitely often in $\Omega(q_0), \Omega(q_1), \Omega(q_2), \ldots$ is even.
$\mathcal{A}$ ***accepts*** the tree T, written $t \models \mathcal{A}$, if there exists an accepting run r on T.

Remark that the only requirement for a finite branch b is that for the last node u of b, if $r(u) = a^q$ then $a(q) \in \delta$.

Given $q \in Q$, $\mathcal{A}_q$ is the automaton obtained from $\mathcal{A}$ by changing the initial state to q. We say that $\mathcal{A}$ accepts the tree T from state q, if $t \models \mathcal{A}_q$. Given an automaton $\mathcal{A}$ and a tree T, we define $\mathcal{A}(T) \subseteq Q$ as the set of states $q \in Q$ such that $\mathcal{A}$ accepts T from q.

Example 2.2.3. Let $\mathcal{A} = \langle \Sigma, Q, \delta, q_{\mathtt{b}}, \Omega \rangle$ be the automaton such that $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ with $\mathit{arity}(\mathtt{a}) = \mathit{arity}(\mathtt{b}) = 2$, $Q = \{q_{\mathtt{a}}, q_{\mathtt{b}}\}$, $\Omega(q_{\mathtt{a}}) = 2$, $\Omega(q_{\mathtt{b}}) = 1$, and δ contains the following transitions:

$$\begin{array}{llll} q_{\mathtt{a}} & \xrightarrow{\mathtt{a}} & (q_{\mathtt{a}}, q_{\mathtt{a}}), & q_{\mathtt{a}} \xrightarrow{\mathtt{b}} (q_{\mathtt{b}}, q_{\mathtt{b}}), \\ q_{\mathtt{b}} & \xrightarrow{\mathtt{a}} & (q_{\mathtt{a}}, q_{\mathtt{a}}), & q_{\mathtt{b}} \xrightarrow{\mathtt{b}} (q_{\mathtt{b}}, q_{\mathtt{b}}). \end{array}$$

This automaton accepts a tree if and only if all its branches contains infinitely many occurrences of $\mathtt{a}$. Indeed, if in a tree a branch contains infinitely many occurrences of $\mathtt{a}$ then the same branch on the run will contains infinitely many occurrences of $q_{\mathtt{a}}$ therefore the greatest colour seen infinitely often is 2, otherwise at some point the branch will only contains occurrences of $q_{\mathtt{b}}$ therefore the greatest colour seen infinitely often would be 1.

Definition 2.2.20 (TAC automaton)**.**

A ***trivial acceptance condition (TAC) automaton*** is a parity automaton $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \Omega \rangle$ such that all states have the colour 0. In that case we omit Ω. Therefore the automaton accepts a tree if and only if there exists a run of $\mathcal{A}$ on that tree, indeed, all runs are accepting.

TAC automata are useful to specify safety properties as illustrated by the next example.

Example 2.2.4. Let $\mathcal{A} = \langle \Sigma, \{q\}, \delta, q \rangle$ be the TAC automaton such that $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ with $\mathit{arity}(\mathtt{a}) = \mathit{arity}(\mathtt{b}) = \mathit{arity}(\mathtt{c}) = 2$ and $\delta = \{q \xrightarrow{\mathtt{a}} (q, q), q \xrightarrow{\mathtt{b}} (q, q)\}$. This automaton accepts the trees that do not contain any occurrence of $\mathtt{c}$.

From an expressiveness point of view, tree automata and MSO formulas are equivalent. However MSO formula are more natural to state properties, while automata are more handy for designing decision procedures and the complexity of standard problems is better on automata. The translation from formula to automata is non-elementary, while the other is linear.

Theorem 2.2.4 ([Rab69])**.**

For any automaton $\mathcal{A}$ there exists a formula $\varphi_{\mathcal{A}}$ such that for all tree T, $T \models \mathcal{A}$ if and only if $T \models \varphi_{\mathcal{A}}$.
For any formula φ there exists an automaton $\mathcal{A}_{\varphi}$ such that for all tree T, $T \models \varphi$ if and only if $T \models \mathcal{A}_{\varphi}$.
Furthermore, these two transformations are effective.

Example 2.2.5. The automaton of example 2.2.3 is equivalent to the following formula:

$$\forall X \; \mathit{Branch}(X) \Rightarrow \forall x \in X \; \exists y \in X \; x \sqsubseteq y \wedge \mathtt{a}(y).$$

Definition 2.2.21 ($\mathcal{A}$-Reflection)**.**

Given a tree T and an automaton $\mathcal{A}$, the ***$\mathcal{A}$-reflection*** on T is the tree $T_{\mathcal{A}}$ such that $T_{\mathcal{A}}$ is a 2^Q-annotation of T such that for all node u, the annotation on u is the set of states from which $\mathcal{A}$ accepts the subtree $T[u]$: $T_{\mathcal{A}}(u) = \big(T(u), \mathcal{A}(T[u])\big)$.

2.2.6. Term rewriting systems

The evaluation process of an HORS consists in rewriting terms of the scheme. Thus we present in a general way term rewriting systems and their properties. In a nutshell a term rewriting system is a set of rewrite rules that transform terms. A term consists of a symbol or a pair of terms.

Definition 2.2.22 (Terms)**.**

Let Γ be an alphabet, then the set of (untyped) ***terms*** on Γ, $\mathcal{T}(\Gamma)$ is inductively defined by:

- $\Gamma \subset \mathcal{T}(\Gamma)$,
- $t \; t' \in \mathcal{T}(\Gamma)$ where $t, t' \in \mathcal{T}(\Gamma)$.

We say that the term $t \; t'$ it is the term t' ***applied*** to the term t. We consider that this notation is ***associative to the left*** *i.e.* one writes $t \; t' \; t''$ for the term $(t \; t') \; t''$. Observe that any term can then be uniquely written $\alpha \; t_1 \ldots t_k$ with $\alpha \in \Gamma$ and for all i, $t_i \in \mathcal{T}(\Gamma)$. We say that α is the ***head*** of the term $\alpha \; t_1 \ldots t_k$.

Definition 2.2.23 (Rewrite rule)**.**

Let Γ be an alphabet and $\mathcal{V}$ a set of ***variables*** disjoint from Γ. A ***rewrite rule*** on $(\Gamma, \mathcal{V})$ is a pair of terms $t \to e$ in $\mathcal{T}(\Gamma \uplus \{x_1, \ldots, x_k\})$ such that:

$$x_1, \ldots, x_k \in \mathcal{V} \text{ and } t = \alpha\ x_1 \ldots x_k \text{ for some } \alpha \in \Gamma.$$

We will write such rewrite rule $\alpha\ x_1 \ldots x_k \to e$, we say that this rewrite rule is ***associated with*** α, and we refer to e as the ***production*** of the rule. Given a rewrite rule $r = \alpha\ x_1 \ldots x_k \to e$ we say that a term $s \in \mathcal{T}(\Gamma)$ is an r***-redex*** if $s = \alpha\ t_1 \ldots t_k$ for some $t_1, \ldots, t_k$. The effect of rule r on the redex s, is based on the substitution of x_i by t_i in e for all i.

For now, we do not formalise what is an occurrence of a symbol in a term (we study occurrences in Chapter 4). However we still may refer to the number of occurrence of a term in another one.

Definition 2.2.24 (Number of occurrence, subterm)**.**

Let $t, t' \in \mathcal{T}(\Gamma)$ two terms. We define inductively $\sharp_{t'}(t)$ the ***number of occurrences*** of t' in t as, if $t = t'$, $\sharp_{t'}(t) = 1$, for $\alpha \in \Gamma$ such that $t' \neq \alpha$, $\sharp_{t'}(\alpha) = 0$ for $t_1, t_2 \in \mathcal{T}(\Gamma)$, if $t' \neq t_1 t_2$, then $\sharp_{t'}(t_1 t_2) = \sharp_{t'}(t_1) + \sharp_{t'}(t_2)$. If $\sharp_{t'}(t) \geq 0$, we shall say that t' is a ***subterm*** of t.

Now we can define the substitution of some term by another, and introduce the notion of context.

Definition 2.2.25 (Substitution, context)**.**

Let Γ be an alphabet and t, s, s' be some terms on Γ. the ***substitution*** of s by s' in the term t, written $t_{[s \mapsto s']}$, is defined by induction:

- $s_{[s \mapsto s']} = s'$,
- $\alpha_{[s \mapsto s']} = \alpha$ for $\alpha \in \Gamma$ and $s \neq \alpha$,
- $(t_1\ t_2)_{[s \mapsto s']} = t_{1[s \mapsto s']}\ t_{2[s \mapsto s']}$ if $s \neq (t_1\ t_2)$.

Let $\bullet \notin \Gamma$. A ***(one hole) context*** is a term $C[\bullet] \in \mathcal{T}(\Gamma \uplus \{\bullet\})$ containing exactly one occurrence of $\bullet$; given a term t, $C[t]$ is defined by $C[\bullet]_{[\bullet \mapsto t]}$.

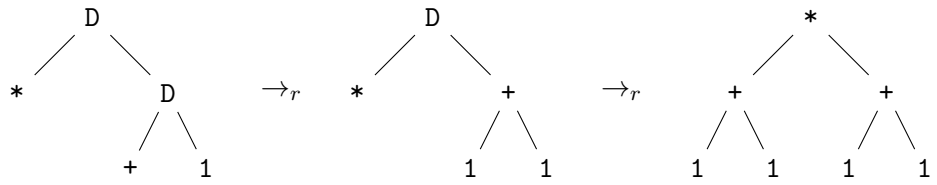


Figure 2.2.: Application of a rewrite rule (Example 2.2.6).

Let Γ be an alphabet and $\mathcal{V}$ a set of variables. Given two terms t, t' in $\mathcal{T}(\Gamma)$ and a rewriting rule r on Γ and $\mathcal{V}$, we say that t ***rewrites*** into t' according to r, written $t \to_r t'$ if there exists a context $C[\bullet]$ and an r-redex $\alpha\ t_1 \dots t_k$ such that $t = C[\alpha\ t_1 \dots t_k]$ and $t' = C[e_{[x_1 \mapsto t_1] \dots [x_k \mapsto t_k]}]$. Note that for any permutation π on $\{1, \dots, k\}$, we have

$$e_{[x_{\pi(1)} \mapsto t_{\pi(1)}] \dots [x_{\pi(k)} \mapsto t_{\pi(k)}]} = e_{[x_1 \mapsto t_1] \dots [x_k \mapsto t_k]}.$$

We write this unique term $e_{[\forall i\ x_i \mapsto t_i]}$.

Example 2.2.6. Let $\Gamma = \{\mathtt{1}, \mathtt{+}, \mathtt{*}, \mathtt{D}\}$ and $r = D\ \varphi\ x\ \to\ \varphi\ x\ x$, then the term `D * (D + 1)` rewrites into `D * (+ 1 1)` that rewrites into `* (+ 1 1) (+ 1 1)`, as illustrated on Figure 2.2.

We are now in position to define term rewriting systems.

Definition 2.2.26 (Term rewriting system)**.**

A ***term rewriting system (TRS)*** is a tuple $\mathcal{S} = \langle \Gamma, \mathcal{V}, \mathcal{R} \rangle$ where Γ is an alphabet called the ***signature*** of $\mathcal{S}$, $\mathcal{V}$ is a set of ***variables***, and $\mathcal{R}$ is a set of rewrite rules on $(\Gamma, \mathcal{V})$.

Definition 2.2.27 (ARS of a TRS)**.**

Let $\mathcal{S} = \langle \Gamma, \mathcal{V}, \mathcal{R} \rangle$ be a TRS. The ARS $\langle \mathcal{T}(\Gamma), \to_{\mathcal{S}} \rangle$, is defined by : $t \to_{\mathcal{S}} t'$ if there exists $r \in \mathcal{R}$ such that $t \to_r t'$.

As usual we write $\to_{\mathcal{S}}^*$ for the reflexive and transitive closure of $\to_{\mathcal{S}}$. We say that a term is a ***redex*** if it is an r-redex for some $r \in \mathcal{R}$. A term rewriting system is ***deterministic*** if for all $\alpha \in \Gamma$ there exists at most one rule associated to α in $\mathcal{R}$.

Once a TRS is fixed, we introduce two other relations $\rightsquigarrow$ and $\Rightarrow$ that will be very handy. Intuitively, if one considers that $\to$ means *rewrite one redex*, $\rightsquigarrow$ means *rewrite some redexes* and $\Rightarrow$ means *rewrite all redexes*.

Definition 2.2.28 (The relation $\rightsquigarrow$)**.**

Let $\mathcal{S} = \langle \Gamma, \mathcal{V}, \mathcal{R} \rangle$ be a TRS. The relation $\rightsquigarrow \subseteq \mathcal{T}(\Gamma) \times \mathcal{T}(\Gamma)$ is inductively defined by:
for all $\alpha \in \Gamma$ and $t_1, \ldots, t_k, t'_1, \ldots, t'_k$ such that for all i, $t_i \rightsquigarrow t'_i$ we have

- if there exists $\alpha\ x_1 \ldots x_k \to e \in \mathcal{R}$, $\alpha\ t_1 \ldots t_k \rightsquigarrow e_{\left[\forall i\ x_i \mapsto t'_i\right]}$,
- $\alpha\ t_1 \ldots t_k \rightsquigarrow \alpha\ t'_1 \ldots t'_k$ (consequently $t \rightsquigarrow t$ for all term t).

Definition 2.2.29 (Parallel rewriting)**.**

Let $\mathcal{S} = \langle \Gamma, \mathcal{V}, \mathcal{R} \rangle$ be a TRS. The relation $\Rightarrow \subseteq \mathcal{T}(\Gamma) \times \mathcal{T}(\Gamma)$ is inductively defined by:
For all $\alpha \in \Gamma$ and $t_1, \ldots, t_k, t'_1, \ldots, t'_k$ such that for all i, $t_i \Rightarrow t'_i$ we have

- if there exists $\alpha\ x_1 \ldots x_k \to e \in \mathcal{R}$, $\alpha\ t_1 \ldots t_k \Rightarrow e_{\left[\forall i\ x_i \mapsto t'_i\right]}$,
- otherwise $\alpha\ t_1 \ldots t_k \Rightarrow \alpha\ t'_1 \ldots t'_k$.

First, remark that if $t \Rightarrow t'$ then $t \rightsquigarrow t'$. Second, if the TRS is deterministic then for all t there exists a unique t' such that $t \Rightarrow t'$; we write $t^\star$ this term.

Theorem 2.2.5 (Confluence)**.**

Given a deterministic TRS, $\mathcal{S} = \langle \Gamma, \mathcal{V}, \mathcal{R} \rangle$, its associated ARS $\langle \mathcal{T}(\Gamma), \to_{\mathcal{S}} \rangle$ is confluent.

Sketch of proof. Let t, t' be two terms. If $t \rightsquigarrow t'$ then $t' \rightsquigarrow t^\star$. In particular, in deterministic TRS the relation $\rightsquigarrow$ fulfills the diamond property. On the other hand if $t \rightsquigarrow t'$ then $t \to^* t'$ and if $t \to t'$ then $t \rightsquigarrow t'$. Therefore, using Proposition **2.2.1** one concludes. □

In deterministic TRS, given a term t, we define the ***parallel derivation*** as the sequence $t \Rightarrow t_1 \Rightarrow t_2 \Rightarrow \cdots$ It is interesting for three reasons, first it is unique, then it induces a derivation is the usual sense as $t \to^* t_1 \to^* t_2 \to^* \cdots$, and finally for all derivation $t \to t'_1 \to t'_2 \to \cdots$ we have for all i, $t'_i \to^* t_i$.

2.3. Higher order recursion schemes

A higher order recursion scheme is a particular case of a deterministic TRS whose main feature is the presence of types for terms that must be preserved by the rewrite rules.

2.3.1. Syntax

The notion of simple types is an abstraction of the types of elements that appear in programming languages: base types (integer, boolean float, etc.) or mappings whose parameters and return value are typed. When abstracting this notion, we consider a single base type (called the ground type) and functional types.

Definition 2.3.1 (Types)**.**

We fix two symbols o and $\to$. The set Types of all ***(simple) types*** is defined by (finite) induction as:

- $o \in$ Types and it is called the ***ground type***,
- for all $\tau_1, \tau_2 \in$ Types, $(\tau_1 \to \tau_2) \in$ Types.

We consider that the writing $\tau_1 \to \tau_2$ is ***associative to the right*** *i.e.* for all τ_1, τ_2, τ_3, $\tau_1 \to \tau_2 \to \tau_3$ denotes $\tau_1 \to (\tau_2 \to \tau_3)$. Remark that any type can be uniquely written $\tau_1 \to \cdots \to \tau_k \to o$. For example $o \to ((o \to o) \to o)$ and $o \to (o \to o) \to o$ denote the same type. For all $\ell \geq 0$, we define $\tau^\ell \to \tau'$ as τ' if $\ell = 0$ and otherwise $\underbrace{\tau \to \cdots \to \tau}_{\ell \text{ times}} \to \tau'$.

Given a type $\tau = \tau_1 \to \cdots \to \tau_k \to o$ we say that k is the ***arity*** of τ, and we define inductively the ***order*** of τ as $\mathit{order}(o) = 0$ and for $k > 0$ and $\tau_1, \ldots, \tau_k \in$ Types, $\mathit{order}(\tau_1 \to \cdots \to \tau_k \to o) = 1 + \max_{i \leq k} \mathit{order}(\tau_i)$. For example all types τ of order 0 or 1 can be written $o^k \to o$ where $0 \leq k$ is the arity of τ.

Typed terms will be terms over a typed alphabet equipped with a type that can be deduced from their structure and the types associated with the symbols.

Definition 2.3.2 (Typed set)**.**

A ***typed set*** is a pair $\langle \Gamma, : \rangle$, where:

- Γ a set,
- $:$ is a (total) functional relation from Γ to Types, called the ***typing relation***.

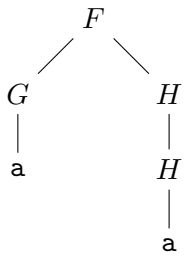


Figure 2.3.: The term $F\ (G\ \mathtt{a})\ (H\ (H\ \mathtt{a}))$ represented as its ***syntactical tree***.

In general, with one set Γ we will associate a unique typing relation, therefore we simply denote by Γ the typed set $\langle \Gamma, : \rangle$. For all $\alpha : \tau \in \Gamma$, we define the order and the arity of α as the order and the arity of τ. Given a type τ, we write $\Gamma^\tau = \{\alpha \in \Gamma \mid \alpha : \tau\}$.

Compared to Definition **2.2.22**, here the construction of terms must be performed according to the types.

Definition 2.3.3 (Typed Terms)**.**

Given a typed set Γ, the typed set $\mathcal{T}(\Gamma)$ of ***typed terms*** on Γ is inductively defined by:

- for all $\alpha : \tau \in \Gamma$ we have $\alpha : \tau \in \mathcal{T}(\Gamma)$,
- for all $t : \tau_1 \to \tau_2 \in \mathcal{T}(\Gamma)$ and $t' : \tau_1 \in \mathcal{T}(\Gamma)$, $t\ t' : \tau_2 \in \mathcal{T}(\Gamma)$.

The abbreviations and notations introduced for general terms also apply in this context. Thus any term of type τ can be uniquely written $\alpha\ t_1 \dots t_k$ for some $\alpha : \tau_1 \to \cdots \to \tau_k \to \tau \in \Gamma$ and some $t_i : \tau_i \in \mathcal{T}(\Gamma)$ with $1 \leq i \leq k$.

Example 2.3.1. Let $\Gamma = \{F : (o \to o) \to o \to o\ ,\ G : o \to o \to o\ ,\ H : (o \to o)\ ,\ \mathtt{a} : o\}$: $(F\ H)$ and $(G\ \mathtt{a})$ are terms of type $o \to o$; $F(G\ \mathtt{a})\ (H\ (H\ \mathtt{a}))$ is a term of type o; $(F\ \mathtt{a})$ is not a term since F is expecting a first argument of type $o \to o$ while $\mathtt{a}$ is of type o. We may give a graphical representation of the terms as depicted on Figure 2.3.

Here a rewrite rule must preserve the typing relation, during its application. More specifically we require that the term and its associated production are of ground type.

Definition 2.3.4 (Rewrite rule)**.**

Given two disjoint typed sets Γ and $\mathcal{V}$, a (typed) ***rewrite rule*** on $(\Gamma, \mathcal{V})$ is a pair of terms $t \to e$ in $\mathcal{T}(\Gamma \uplus \{x_1, \ldots, x_k\})^2$ such that:

- $x_1, \ldots, x_k \in \mathcal{V}$,
- t and e have type o,
- $t = \alpha\ x_1 \ldots x_k$ for some $\alpha \in \Gamma$, (in particular $k = \mathit{arity}(\alpha)$).

Example 2.2.6 continued. If we type the alphabet Γ as $\Gamma = \{\mathtt{1} : o, \mathtt{+}, \mathtt{*} : o^2 \to o, \mathtt{D} : (o^2 \to o) \to o)\}$, the rewrite rule $r = D\ \varphi\ x\ \to\ \varphi\ x\ x$ is a valid typed rewrite rule.

We are now in position to introduce higher order recursion schemes [Dam77a][1]. By analogy with programming languages, there are two kinds of symbols in HORS: terminals symbols which correspond to built-in functions or operators of the programming language, and non-terminal symbols which correspond to user-defined functions. In addition variables of HORS are used to specify parameters of the user-defined functions. Finally, there is one rewrite rule per non-terminal symbol which corresponds to the "body" of the function denoted by the symbol. In HORS, built-in functions are supposed to be typed with order 0 or 1, meaning that they correspond either to a constant (type o) or to a function manipulating ground typed arguments (type $o^k \to o$ for some k).

Observe that there is a one-to-one correspondance from typed alphabets whose characters have order 0 or 1, to ranked alphabet by considering that a symbol $a : o^k \to o$ is a ranked symbol of arity k.

[1] Damm only defined safe terms then.

Definition 2.3.5 (Higher order recursion scheme)**.**

A ***higher order recursion scheme (HORS)*** $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ is a tuple such that:

- $\mathcal{V}$ is a finite typed set whose elements are called the ***variables*** of the scheme;
- Σ is a finite typed alphabet whose elements are called the ***terminals*** of the scheme;
- $\mathcal{N}$ is a finite typed set whose elements are called the ***non-terminals*** of the scheme;
- $\mathcal{R}$ is a set of rewrite rules on $\Sigma \uplus \mathcal{N}$ and $\mathcal{V}$, with each non-terminal is associated exactly one rewrite rule, and no rule is associated with any terminal;
- $S : o \in \mathcal{N}$ is the ***initial non-terminal***.

We write $\to_{\mathcal{G}}$ the rewrite relation associated to $\mathcal{R}$ (or simply $\to$ when $\mathcal{G}$ is clearly established by the context). The implicit TRS $\langle \mathcal{T}(\Sigma \uplus \mathcal{N}), \mathcal{R} \rangle$ associated with a scheme $\mathcal{G}$ is deterministic, therefore its corresponding ARS is confluent.

In the following we may refer to the terms of $\mathcal{T}(\Sigma \uplus \mathcal{N})$ as the terms of the scheme $\mathcal{G}$.

Example 2.3.2. (This example is adapted from [BCHS13]) Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ be the scheme such that: $\mathcal{V} = \{x : o, \varphi : o \to o\}$, $\Sigma = \{\mathtt{or_s} : o^2 \to o, \mathtt{or_u} : o^2 \to o, \mathtt{upgrade} : o \to o, \mathtt{playWith} : o \to o, \mathtt{toy} : o, \mathtt{null} : o\}$. The set of non-terminals is $\mathcal{N} = \{Play : o \to o, Up : (o \to o) \to o \to o\}$, and $\mathcal{R}$ contains the following rewrite rules:

$$\begin{array}{lcl} S & \to & Play\ \mathtt{toy} \\ Play\ x & \to & \mathtt{or_u}\ (\mathtt{playWith}\ x)\ (Up\ Play\ x) \\ Up\ \varphi\ x & \to & \mathtt{or_s}\ (\varphi\ \mathtt{null})\ (\varphi\ (\mathtt{upgrade}\ x)). \end{array}$$

This scheme describes a program where the user wants to play with a toy. The program starts by calling the function $Play$ on `toy`. In $Play$ the user has the choice either to actually play with the toy or to upgrade it. If he asks to upgrade the toy then the system is asked if there exists an upgrade for the toy. If there exists one then the program goes back on the $Play$ function on the upgraded toy, if there does not exists an upgrade, then the Up function launches $Play$ on `null`. The choice of the user and of the system are represented by the terminals $\mathtt{or_u}$ and $\mathtt{or_s}$. Notice that in this example the tree structure is needed since this program does not have a unique behaviour.

2.3.2. Semantics

We provide a decorated ARS as semantics of a HORS. The rewriting system of the HORS already induces an ARS, so it remains to define a mapping from typed terms of the HORS to trees over Σ which fulfils the requirement for a decorated ARS (Equation **2.2.10**).

A first idea could be to consider the syntactical tree of a term as its mapping. However non-terminal symbols occurring in the term correspond to parts of the HORS that have not been evaluated yet. Thus the correct mapping is obtained by replacing each subtree whose root is labelled by a non-terminal symbol by a leaf labelled by $\bot$. In order to avoid that missing arguments in the term modify the arity of a symbol, the domain of this partial mapping is the set of ground typed terms. In the sequel we denote a finite tree by its prefix representation, which exactly corresponds to a ground typed term of $\mathcal{T}(\Sigma \uplus \{\bot\})$.

Definition 2.3.6 (The $\bot$-transformation)**.**

Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ be a HORS and let $t = \alpha\ t_1 \dots t_k : o \in \mathcal{T}(\Sigma \uplus \mathcal{N})$. Then the tree $t^\bot$ is inductively defined by:

- $(\alpha\ t_1\ \dots\ t_k)^\bot\ =\ \alpha\ t_1^\bot\ \dots\ t_k^\bot$ if $\alpha : o^k \to o \in \Sigma$,
- $(\alpha\ t_1\ \dots\ t_k)^\bot\ = \bot$ otherwise (in this case α is a non-terminal).

We set the partial mapping f from ground typed terms to trees by $f(t) = t^\bot$. Given two terms $t, t' : o$ such that $t \to t'$ one can show by induction on the structure of t that $t^\bot \sqsubseteq t'^\bot$. Therefore a scheme $\mathcal{G}$ is a decorated ARS $\langle \mathcal{T}(\Sigma \uplus \mathcal{N}), \to_\mathcal{G}, f \rangle$, and since it is a deterministic term rewriting system, it is confluent. Thus given a ground term $t : o$, the value tree of t in the decorated ARS is called the ***value tree*** of t in the scheme $\mathcal{G}$ and is written $\|t\|_\mathcal{G}$. We define the value tree of the scheme, written $\|\mathcal{G}\|$, as the value tree of S, $\|\mathcal{G}\| = \|S\|_\mathcal{G}$.

Example 2.3.2 continued. An example of derivation is depicted in Figure 2.4, the value tree of the scheme is depicted in Figure 2.5 (we use `pW` and `up` as shorthands for `playWith` and `upgrade`). The leftmost branch describes an execution of the program where the user never choses to upgrade the toy. The rightmost branch on the other hand describes an execution where the user always want to upgrade the toy, and the system always grant him with an upgrade. The second leftmost branch describes an execution where the user wanted an upgrade, and the system responded `null`, then the user decides to play with `null`.

Given a ground typed term $t : o$, the scheme $\mathcal{G}_t$ is obtained from the $\mathcal{G}$ by making it start from the term t instead of the non terminal S.

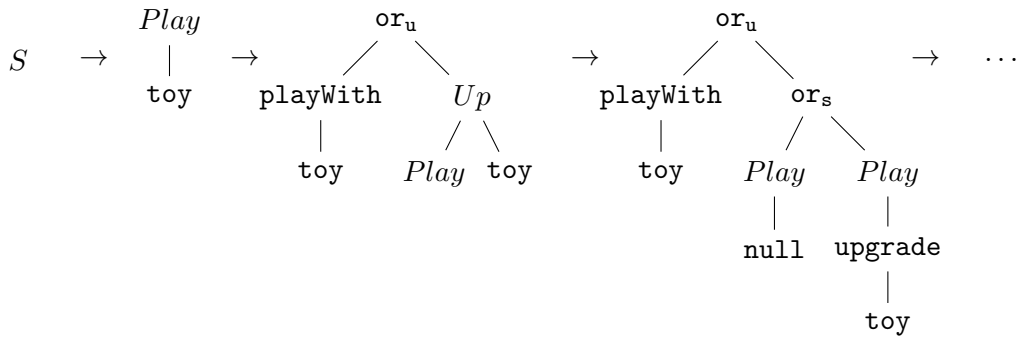


Figure 2.4.: An example of derivation of the scheme of example 2.3.2.

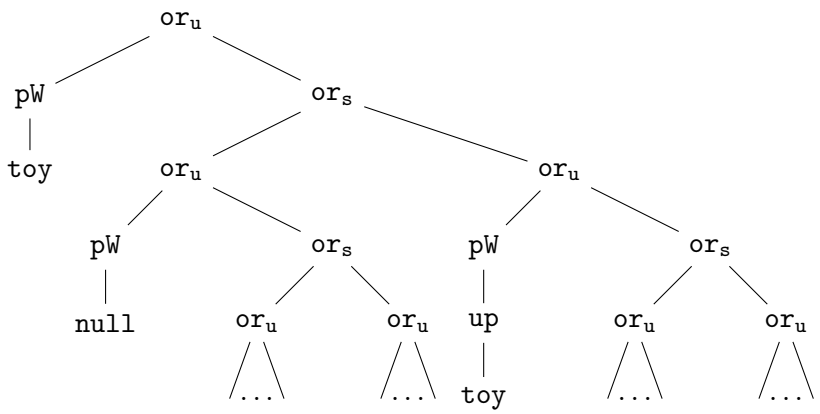


Figure 2.5.: The value tree of the scheme of example 2.3.2.

Definition 2.3.7 (The scheme $\mathcal{G}_t$)**.**

Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be an HORS, and $t : o$ be a term in $\mathcal{T}(\Sigma \uplus \mathcal{N})$. The scheme $\mathcal{G}_t = \langle \mathcal{V}, \Sigma, \mathcal{N}', I, \mathcal{R}' \rangle$ is defined by:

- $\mathcal{N}' = \mathcal{N} \uplus \{I : o\}$
- $\mathcal{R}' = \mathcal{R} \uplus \{I \to t\}$.

Note that $\|t\|_{\mathcal{G}} = \|I\|_{\mathcal{G}_t} = \|\mathcal{G}_t\|$.

We now introduce evaluation policies for HORS. In order to get some intuition about such policies let us discuss the value of this pseudo C program.

```
int i = 1;
int f(int x) {
    return i + x;
}
print(f(i++));
```

Assume that the parameter of function `f` is called by value. Then its argument is 1 and `i` becomes 2. Afterwards executing the body of `f` returns 3. Assume now that the parameter is called by name then the body of `f` becomes return `i + i++`, and the return value is 2. Such a problem could not occur in HORS due to confluence. However depending on evaluation policies the value tree could be more or less defined. We now formalise evaluation policies in HORS and describe the two main possible policies OI (outermost-innermost) and IO (innermost-outermost).

Definition 2.3.8 (Evaluation policy)**.**

Given an HORS $\mathcal{G}$, an ***evaluation policy*** is a binary relation $\triangleright$ included in $\to_{\mathcal{G}}$. A $\triangleright$***-derivation*** is defined as a derivation $x_0 \to x_1 \to x_2 \to \cdots$ such that for all i, $x_i \triangleright x_{i+1}$.

Of course, even if $\to_{\mathcal{G}}$ is confluent, $\triangleright$ may not necessarily be confluent. Here, the only evaluation policies we are interested in should be confluent. Given a confluent evaluation policy $\triangleright$ we define the $\triangleright$-value tree of the scheme as $\|\mathcal{G}\|_{\triangleright} = \|S\|_{\mathcal{G},\triangleright}$. Observe that we always have $\|\mathcal{G}\|_{\triangleright} \sqsubseteq \|\mathcal{G}\|$.

The two evaluations policies we introduce are described by putting some constraints on redexes allowed to be rewritten. The OI policy allows only the redex that are not

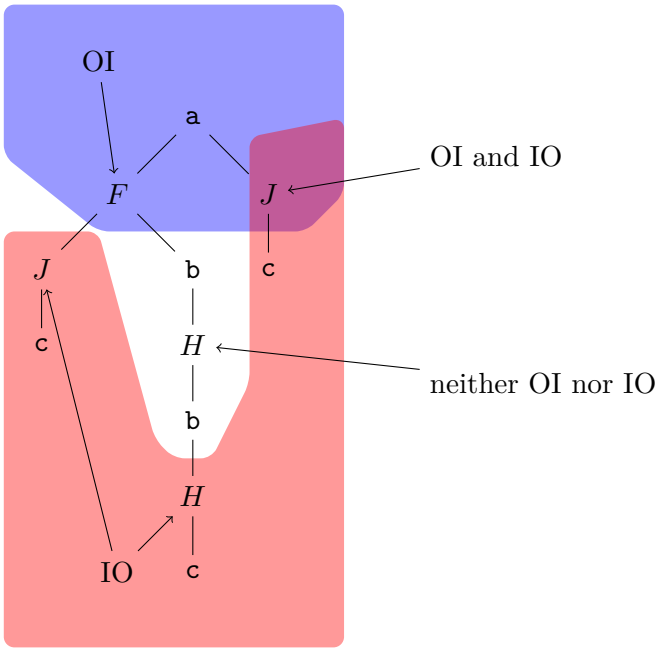


Figure 2.6.: An illustration of OI and IO

contained in any redex, *i.e.* the redex that are the closest to the root. On the other hand, the IO policy allows only the redex that do not contain any redex as strict subterms. They are described in Figure 2.6. Remark that some redex can be allowed by both evaluation policies while other redex cannot be allowed by any.

Definition 2.3.9 (OI).

The subset of OI-contexts is inductively defined by:

- $\bullet$ is an OI-context,
- $\alpha\ t_1 \dots t_k$ is an OI-context if (1) it is a context, (2) it is not a redex, and (3) some t_i is a OI context.

Let $F\ x_1 \dots x_k \to e \in \mathcal{R}$ and consider the rewriting $t \to t'$ with $t = C[F\ t_1 \dots t_k]$ and $t' = C[e_{[\forall i\ x_i \mapsto t_i]}]$. We say that $t \to t'$ is OI, if $C[\bullet]$ is OI and we write $\to_{\text{OI}}$ this evaluation policy.

The next proposition whose second assertion is a consequence of a well known λ-calculus theorem, called the ***standardisation theorem*** (see for example [Bar85, Dam82]), emphasises the interest of OI policy.

Proposition 2.3.1.

Given a scheme $\mathcal{G}$, the ARS $\langle \mathcal{T}(\Sigma \uplus \mathcal{N}), \to_{\mathrm{OI}} \rangle$ is confluent. Furthermore $\|\mathcal{G}\|_{\to_{\mathrm{OI}}} = \|\mathcal{G}\|$.

Definition 2.3.10 (IO)**.**

We say that a term $\alpha\ t_1 \dots t_k$ is IO if for all i, there is no occurrence of any redex in t_i.
Let $F\ x_1 \dots x_k \to e \in \mathcal{R}$ and the rewriting $t \to t'$ with $t = C[F\ t_1 \dots t_k]$ and $t' = C[e_{[\forall i\ x_i \mapsto t_i]}]$. We say that $t \to t'$ is IO, if $F\ t_1 \dots t_k$ is IO and we write $\to_{\mathrm{IO}}$ this evaluation policy.

Proposition 2.3.2.

Given a scheme $\mathcal{G}$, the ARS $\langle \mathcal{T}(\Sigma \uplus \mathcal{N}), \to_{\mathrm{IO}} \rangle$ is confluent. We denote $\|\mathcal{G}\|_{\mathrm{IO}}$ the tree $\|\mathcal{G}\|_{\to_{\mathrm{IO}}}$.

In general $\|\mathcal{G}\|_{\mathrm{IO}}$ is not equal to $\|\mathcal{G}\|$. Take the following dummy scheme: $\mathcal{G} = \langle \{\mathtt{a} : o\}, \{S : o, F : o \to o, H : o\}, \mathcal{R}, S \rangle$ such that $\mathcal{R}$ contains the following rules:

$$S \ \to \ F\ H \qquad F\ x \ \to \ \mathtt{a} \qquad H \ \to \ H.$$

Since $S \to F\ H \to \mathtt{a}$, $\|\mathcal{G}\| = \mathtt{a}$. However the only possible IO derivation is $S \to_{\mathrm{IO}} F\ H \to_{\mathrm{IO}} F\ H \to_{\mathrm{IO}} \cdots$, which leads to $\|\mathcal{G}\|_{\mathrm{IO}} = \bot$.

2.3.3. Main algorithmic problems

In this section we present algorithmic problems, *i.e.* problems described as an input and an output, and *solving* one of these problems consists in constructing an algorithm whose input and output correspond to the one specified in the problem.

An HORS is a program where the built-in functions are uninterpreted. Therefore, as seen in example 2.3.2 the value tree of a scheme includes all the possible executions of such a program, whatever the interpretation. Thus establishing properties of the value tree gives some useful information about the corresponding program. This leads to the model checking problem.

Problem 1 (Model checking). *The* MSO *(resp. automaton)* ***model checking problem*** *is stated as follows.*
Input: *A scheme* $\mathcal{G}$*, an* MSO *formula* φ *(resp. an automaton* $\mathcal{A}$*).*
Output: *The truth value of* $\|\mathcal{G}\| \models \varphi$ *(resp.* $\|\mathcal{G}\| \models \mathcal{A}$*).*

Given a property on nodes, for example a formula $\varphi[x]$, when wanting to know which are the nodes of a tree satisfying $\varphi[x]$, one cannot evaluate one node after another since there are infinitely many of them. Therefore the global model checking problem asks given a property on nodes, to output a generator of the set of nodes satisfying the property.

Problem 2 (Global model checking). *The* MSO *(resp. automata)* ***global model checking*** *problem (also called the* ***reflection problem****) is stated as follows.*
Input: *A scheme* $\mathcal{G}$*, of* $\|\mathcal{G}\|$*, an* MSO *formula* $\varphi[x]$ *(resp. an automaton* $\mathcal{A}$*).*
Output: *A scheme* $\mathcal{G}'$ *such that* $\|\mathcal{G}'\|$ *is the reflection of* φ *(resp. of* $\mathcal{A}$*) on* $\|\mathcal{G}\|$*.*

Finally, given a tree and a formula, sometimes knowing that the tree satisfies the formula is not enough, one want a proof or a witness of this. For example if a tree satisfies the property "there exists a branch entirely labelled by the letter `a`", one would like to exhibit such a branch. This problem is called the selection.

Problem 3 (Selection). *The* MSO *(resp. automaton)* ***selection problem*** *as follows.*
Input: *A scheme* $\mathcal{G}$*, of* $\|\mathcal{G}\|$*, an* MSO *formula* $\varphi[X]$ *(resp. an automaton* $\mathcal{A}$*) such that* $\|\mathcal{G}\| \models \varphi[X]$ *(resp.* $\|\mathcal{G}\| \models \mathcal{A}$*).*
Output: *A scheme* $\mathcal{G}'$ *such that* $\|\mathcal{G}'\|$ *is a selector of* φ *(resp. an accepting run of* $\mathcal{A}$*) on* $\|\mathcal{G}\|$*.*

Remark Remark that these problems have two versions, the MSO problem and the Automata problem. Due to the equivalence results between MSO and the tree automata, both model checking problems trivially are equivalents, as for the selection problems. However automata reflection problem is simpler that MSO reflection problem, therefore Section 4.3.2 explain how we can solve the MSO reflection problem from the automata reflection problem.

The next problems are more related to the behaviour of the scheme rather than to the value tree. The first two concern the simulation of one evaluation policy by another while the third one asks if it is possible to enhance the scheme in a certain way defined below.

Problem 4 (OI-**Simulation).** *The* OI-***simulation problem*** *is stated as follows.*
Input: *A scheme* $\mathcal{G}$*.*
Output: *A scheme* $\mathcal{G}'$ *such that* $\|\mathcal{G}'\|_{\mathrm{IO}} = \|\mathcal{G}\|$*.*

Problem 5 (IO-**Simulation).** *The* IO-***simulation problem*** *is stated as follows.*
Input: *A scheme* $\mathcal{G}$*.*
Output: *A scheme* $\mathcal{G}'$ *such that* $\|\mathcal{G}'\| = \|\mathcal{G}\|_{\mathrm{IO}}$*.*

Given a tree t on the ranked alphabet Σ, and two symbols a and b of same arity, we note $t_{[a\mapsto b]}$ the tree obtained by replacing all occurrences of a by b in t. Notice that if $b \neq a$, there is no node labelled by a in $t_{[a\mapsto b]}$.

A term $t : o$ is ***unproductive*** if $\|t\| = \bot$. Syntactically detecting unproductive terms and replacing them by a new symbol `bot` would make the evaluation more efficient as it avoids useless derivations. This motivates the following problem.

Problem 6 ($\bot$-Elimination). *The* ***$\bot$-elimination problem*** *is stated as follows.*
Input: *A scheme $\mathcal{G}$ on the signature Σ,*
Output: *A scheme $\mathcal{G}'$ on the signature $\Sigma \uplus \{\texttt{bot} : o\}$ such that $\|\mathcal{G}'\| = \|\mathcal{G}\|_{[\bot\mapsto\texttt{bot}]}$.*

Remark 2.3.1. The main idea about the $\bot$-elimination problem, is to be able to spot directly when a redex would lead to $\bot$, and to rewrite it immediately, instead of letting the redex be uselessly rewritten. The fact of using the symbol `bot` instead of $\bot$ is simply to avoid confusion between the symbols produced by the scheme and the special symbol.

In particular, once we solve the $\bot$-elimination problem, we use this algorithm (or a similar transformation) to solve other problems. Therefore when we say for example that we produce a new scheme whose value tree is the reflection of the value tree of the input scheme, this is true up to the substitution of the symbol `bot` by $\bot$ in the resulting scheme.

2.4. Equivalent models

2.4.1. λY-Calculus

The λY-calculus is an extension of the typed λ-calculus with a ***fixpoint operator*** Y that allows to have infinite derivations, while due to strong normalisation, any term of the typed λ-calculus has only finite derivations. Adding the fixpoint operator Y to the typed λ-calculus has first been studied in the thesis of R. Platek in the 60's [Pla66].

Definition 2.4.1 (λY-terms)**.**

We fix the symbols λ and Y, and a typed set $\mathcal{V}$ whose elements are called ***variables***, such that for all type τ, $\mathcal{V}^\tau$ is infinite. Given a ranked alphabet Σ the typed set $\Lambda(\Sigma)$ of λY***-terms*** over Σ is inductively defined:

- $Y^\tau : (\tau \to \tau) \to \tau \in \Lambda(\Sigma)$ for all type τ,
- $x : \tau \in \Lambda(\Sigma)$ for all $x : \tau \in \mathcal{V}$,
- $a : o^k \to o \in \Lambda(\Sigma)$ for all $a : o^k \to o \in \Sigma$,
- $\lambda x.N : \tau \to \tau' \in \Lambda(\Sigma)$ for all $x : \tau \in \mathcal{V}$ and $N : \tau' \in \Lambda(\Sigma)$,
- $M\ N : \tau' \in \Lambda(\Sigma)$, for all $M : \tau \to \tau' \in \Lambda(\Sigma)$ and $N : \tau \in \Lambda(\Sigma)$.

Remark that any λY-term can be uniquely written $N\ N_1 \dots N_k$ with $N = a \in \Sigma$, $N = x \in \mathcal{V}$, $N = Y^\tau$ for some τ or $N = \lambda x.N'$ for some $N' \in \Lambda(\Sigma)$.

The λY-term $\lambda x.M$ can be seen as the mapping that takes an argument x and when applied to a term N, returns M where x is replaced by N. Therefore we define a rewrite relation, called the beta reduction as $(\lambda x.M)\ N \to_\beta M_{[x \mapsto N]}$ where $M_{[x \mapsto N]}$ is the substitution of all occurrences of x in M by N. However this substitution must be defined more carefully than we did previously for HORS to avoid unwanted reductions.

Definition 2.4.2 (Free variables, bounded variables)**.**

Given a term M, the sets $FV(M) \subset \mathcal{V}$ and $BV(M) \subset \mathcal{V}$ of ***free variables*** and ***bounded variables*** of M are inductively defined by:

- $FV(Y^\tau) = \emptyset$, $BV(Y^\tau) = \emptyset$,
- $FV(x) = \{x\}$, $BV(x) = \emptyset$ for $x \in \mathcal{V}$,
- $FV(a) = \emptyset$, $BV(a) = \emptyset$ for $a \in \Sigma$,
- $FV(\lambda x.N) = FV(N) \backslash \{x\}$, $BV(\lambda x.N) = BV(N) \cup \{x\}$
- $FV(M\ N) = FV(M) \cup FV(N)$, $BV(M\ N) = BV(M) \cup BV(N)$.

Substitution is a syntactical rewriting of a variable x by an expression N in a term M. However in order to get the intended interpretation of substitution, two restrictions are required: (1) x should not be bounded in M since such occurrences of x should be

unchanged and (2) the free variables of N must remain free in the substitution, *i.e.* they should not be bounded in M.

Definition 2.4.3 (Substitution)**.**

Given $x : \tau \in \mathcal{V}$, $M : \tau', N : \tau \in \Lambda(\Sigma)$, such that $(FV(N) \cup \{x\}) \cap BV(M) = \emptyset$, the term $M_{[x \mapsto N]}$ is defined by induction on M by:

$$Y^\tau{}_{[x \mapsto N]} = Y^\tau, \qquad a_{[x \mapsto N]} = a \text{ for } a \in \Sigma,$$
$$x_{[x \mapsto N]} = N, \qquad y_{[x \mapsto N]} = y \text{ if } y \in \mathcal{V} \text{ and } y \neq x,$$
$$(\lambda y.M')_{[x \mapsto N]} = \lambda y.M'_{[x \mapsto N]}, \qquad (M_1\ M_2)_{[x \mapsto N]} = M_{1[x \mapsto N]}\ M_{2[x \mapsto N]},$$

Most of the transformation between terms should be applied to subterms. This justifies the next definition.

Definition 2.4.4 (Closure by context)**.**

A relation $\triangleright$ over λY-terms is closed by context if given $M_1, M_2, N_1, N_2 \in \Lambda(\Sigma)$,

- if $M_1 \triangleright M_2$ then $\lambda x.M_1 \triangleright \lambda x.M_2$,
- if $M_1 \triangleright M_2$ and $N_1 = N_2$ or $N_1 \triangleright N_2$ and $M_1 = M_2$ then $M_1\ N_1 \triangleright M_2\ N_2$.

The β-reduction consists in applying a "function" to an "argument" inside a λY-term.

Definition 2.4.5 (β-Reduction)**.**

The relation $\rightarrow_\beta$, over λY-terms, called the ***β-reduction***, is defined as the smallest relation closed under context such that $(\lambda x.M)\ N \rightarrow_\beta M_{[x \mapsto N]}$ if it is defined.

The δ-reduction provides a semantics to the fixpoint operator. Observe that applying n times this rule to $Y^\tau\ M$ leads to $M(M(\dots M(Y^\tau M)\dots))$.

Definition 2.4.6 (δ-Reduction)**.**

The relation $\to_\delta$ over λY-terms, called the ***δ-reduction***, is defined as the smallest relation closed under context such that for all $M : \tau \to \tau$,

$$Y^\tau\ M \to_\delta M\ (Y^\tau\ M).$$

We use the notation $\to_{\beta,\delta}$ as a shorthand for $\to_\beta \cup \to_\delta$.

In order to apply substitution in any case, one must transform a term that does not fulfil the requirement for substitution by an equivalent term. The α-renaming relation characterises this equivalence. Since we want an equivalence relation, it should be closed by reflexivity, symmetry, and transitivity and in addition it should be a congruence, *i.e.* closed by context. The next definition formalises this idea.

Definition 2.4.7 (α-Renaming)**.**

The relation $\to_\alpha$ over λY-terms is defined by $N_1 \to_\alpha N_2$ if and only if there exist $x \neq y \in \mathcal{V}$ and $M \in \Lambda(\Sigma)$ such that

- $M_{[x \mapsto y]}$ is defined and $y \notin FV(M)$,
- $N_1 = \lambda x.M$ and $N_2 = \lambda y.M_{[x \mapsto y]}$.

We define the relation $=_\alpha$, called the ***α-equivalence***, as the smallest reflexive, transitive and symmetric relation containing $\to_\alpha$ and closed by context.

Definition 2.4.8 (Reduction system)**.**

The set $\Lambda_\alpha(\Sigma)$ is defined as the quotient of $\Lambda(\Sigma)$ by the equivalence relation $=_\alpha$.
The relation $\to_{\beta,\delta}$ is defined on $\Lambda_\alpha(\Sigma)$ by: for all $t_1, t_2 \in \Lambda_\alpha(\Sigma)$, $t_1 \to_{\beta,\delta} t_2$ if there exists $M \in t_1$ and $N \in t_2$ such that $M \to_{\beta,\delta} N$ (resp. $M \to_\delta N$).

Given $M \in \Lambda(\Sigma)$ we denote by the same symbol M the equivalence class that contains M. The following proposition is a folk result (see for example [Bar85]) and can be proven exactly as it has been for HORS.

Proposition 2.4.1.

The ARS $\langle \Lambda_\alpha(\Sigma), \to_{\beta,\delta} \rangle$ is confluent.

Similarly as in HORS, the presence of some Y^τ or λx in a term witnesses that this term is not fully evaluated. According to this analogy, we introduce the $\bot$-transformation.

Definition 2.4.9 (The $\bot$-transformation)**.**

Given a term $M : o \in \Lambda(\Sigma)$ such that $FV(M) = \emptyset$ we define the tree $M^\bot$ by induction on the structure of M as: $(Y^\tau\ N_1 \dots N_k)^\bot = \bot$, $((\lambda x.N)\ N_1 \dots N_k)^\bot = \bot$ and $(a\ N_1 \dots N_k)^\bot = a\ N_1^\bot \dots N_k^\bot$.

Note that if $M =_\alpha N$ then $M^\bot = N^\bot$. Therefore for every $t \in \Lambda_\alpha(\Sigma)$ we define $t^\bot$ as $M^\bot$ for $M \in t$. Due to the confluence of $\langle \Lambda_\alpha(\Sigma), \to_{\beta,\delta} \rangle$, each term M as a ***value tree*** $\|M\|$ in the decorated ARS $\langle \Lambda_\alpha(\Sigma), \to_{\beta,\delta}, (\cdot)^\bot \rangle$.

The following proposition is well known and is described in [SW11].

Proposition 2.4.2 (Equivalence between HORS and λY-terms)**.**

For all λY-term $M : o \in \Lambda_\alpha(\Sigma)$ such that $FV(M) = \emptyset$ there exists a scheme $\mathcal{G}$ such that $\|\mathcal{G}\| = \|M\|$. For all scheme $\mathcal{G}$ there exists a λY-term $M : o \in \Lambda(\Sigma)$ such that $FV(M) = \emptyset$ such that $\|M\| = \|\mathcal{G}\|$.

Informally, the idea to transform a λY-term into a scheme consists of associating a non-terminal to each subterm of the lambda term where the initial symbol corresponds to the whole term. To each Y^τ appearing in the scheme we associate a non terminal $\overline{Y^\tau}$ of the same type as Y^τ with the rule $\overline{Y^\tau}\varphi\ x_1 \dots x_k \to (\varphi(\overline{Y^\tau}\ \varphi))x_1 \dots x_k$ (the variables $x_1, \dots, x_k$ are just here to make the terms ground), and to each term we associate the non terminal whose argument are the free variables of the term. To transform a scheme into a λY-term, we describe each rule of a non terminal as a function where non terminals are replaced by variables and then we use the fixpoint operators Y^τ to describe the recursion. This transformation is illustrated in Example 2.4.1.

Example 2.4.1. As an illustration, we give a λY-term equivalent to the scheme of Example 2.3.2. Recall that $\Sigma = \{\mathtt{or_u} : o^2 \to o, \mathtt{or_s} : o^2 \to o, \mathtt{upgrade} : o \to o, \mathtt{playWith} :$

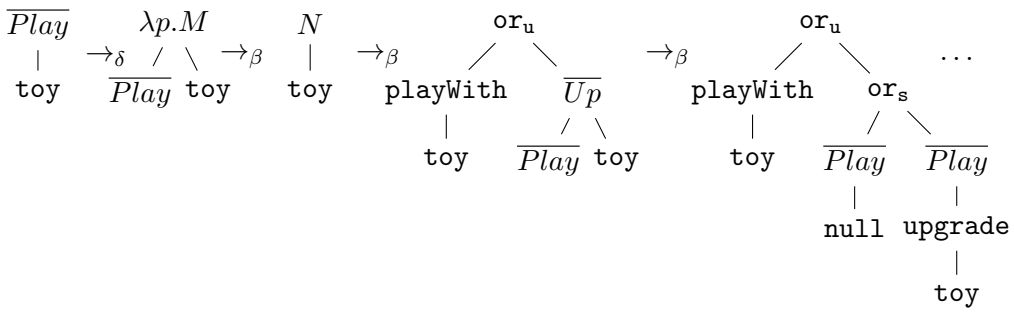


Figure 2.7.: An example of derivation of the λY-term of example 2.4.1.

$o \to o, \mathtt{toy} : o, \mathtt{null} : o\}$ and that the rewrite rules are the following:

$$\begin{array}{lcl} S & \to & Play\ \mathtt{toy} \\ Play\ x & \to & \mathtt{or_u}\ (\mathtt{playWith}\ x)\ (Up\ Play\ x) \\ Up\ \varphi\ x & \to & \mathtt{or_s}\ (\varphi\ \mathtt{null})\ (\varphi\ (\mathtt{upgrade}\ x)). \end{array}$$

First remark that the rule associated to the non terminal Up does not involve any non terminal, therefore we define the λY-term $\overline{Up} : (o \to o) \to o \to o$ as $\overline{Up} = \lambda\varphi.\lambda x.\mathtt{or_s}\ (\varphi\ \mathtt{null})\ (\varphi\ (\mathtt{upgrade}\ x))$.

The rule associated with $Play$ involves the non terminals $Play$ and Up. We have a λY-term equivalent to Up, but to replace the occurrence of $Play$ in the production of the rule, we first use a variable $p : o \to o$ of the same type as $Play$, and we define the term $M = \lambda x.\mathtt{or_u}\ (\mathtt{playWith}\ x)\ (\overline{Up}\ p\ x)$. Now to indicate that there is a recursion here, we will use the operator $Y^{o\to o}$ defining $\overline{Play} = Y^{o\to o}(\lambda p.M)$. Finally we define the λY-term $\overline{S} = \overline{Play}\ \mathtt{toy}$, and we have $\|\overline{S}\| = \|\mathcal{G}\|$. An example of a derivation is depicted in Figure 2.7, where the λY-term N is equal to $M_{[p\mapsto\overline{Play}]} = \lambda x.\mathtt{or_u}\ (\mathtt{playWith}\ x)\ (\overline{Up}\ \overline{Play}\ x)$.

To study the λY-calculus, Salvati and Walukievicz used ***Krivine Machines*** as a theoretical tool working as an interpreter for λY-terms, and analysed properties of these objects. The behaviour of a Krivine Machine on a λY-term is comparable to the run of a CPDA simulating a scheme, as introduced in the next section (Example 2.4.2). We do not formally present Krivine machines here, see [SW11] for a detailed definition.

2.4.2. Collapsible pushdown automata

Collapsible pushdown automata are another model equi-expressive with HORS in the sense that they produce the same trees. It is a pushdown machine, where the stacks are more general than regular ones. As an analogy with programming language, if a scheme represents a higher order program, a collapsible pushdown automaton may represent an interpreter of the program. Another way to look at it, as we will see later on, is to see collapsible pushdown automata as machines able to derive recursive schemes.

Definition 2.4.10 (Stack)**.**

Given a set X, the set $\mathrm{Stacks}(X)$ of ***stacks*** on X is defined as the set of non-empty finite sequences of elements of X.

We write $[\gamma_1\gamma_2\dots\gamma_\ell]$ the stack that contains $\gamma_1,\dots,\gamma_\ell$ in that order. For any stack $s = [\gamma_1\dots\gamma_\ell]$, we say that γ_ℓ is the ***top*** of s, written $top(s)$. We fix a finite alphabet Γ called the ***stack alphabet***, and we refer to its elements as the stack symbols.

Definition 2.4.11 (Higher order stack)**.**

The set $\mathrm{Stacks}_n(\Gamma)$ of ***order-n stacks*** on Γ is defined by induction on n by: $\mathrm{Stacks}_0(\Gamma) = \Gamma$ and $\mathrm{Stacks}_{n+1}(\Gamma) = \mathrm{Stacks}(\mathrm{Stacks}_n(\Gamma))$.

We write $\mathrm{HOS}(\Gamma)$ the set of all higher order stacks, $\mathrm{HOS} = \bigcup_{n\geq 1}\mathrm{Stacks}_n(\Gamma)$. We precise the order of a stack by a subscript, for example $[s_1s_2]_4$ denote an order-4 stack containing two order-3 stacks s_1 and s_2.

We now generalise to higher-order the three main operations over order-1 stacks: top, push, pop. Consistently with the definition, "popping" an order-i stack within a global stack requires the presence of at least two order-i stacks in this stack. First we extend the notion of top of the stack.

Definition 2.4.12 (Tops of a higher order stack)**.**

Given an order-n stack s with $n > 0$, we define inductively its top_i stack for all $i \leq n+1$ by:

- $top_n([s_1\dots s_\ell]_n) = s_\ell$,
- $top_i([s_1\dots s_\ell]_n) = top_i(s_\ell)$ if $i < n$.

Intuitively, $top_i(s)$ is the top of the topmost order-i stack, therefore it is an order-$(i-1)$ stack. Furthermore, we define the notion of pushing an order-k stack s on top of an order-n stack with $n > k$.

Definition 2.4.13 (Generalised push operation)**.**

The stack $[s_1 \dots s_\ell]_n +\!\!+ s$ is inductively defined by:

- $[s_1 \dots s_\ell]_n +\!\!+ s = [s_1 \dots s_\ell s]_n$ if $n = k + 1$,
- $[s_1 \dots s_\ell]_n +\!\!+ s = [s_1 \dots (s_\ell +\!\!+ s)]_n$ if $n > k + 1$.

In the following, we are mainly interested in two operations, derived from $+\!\!+$. The operation $push_\gamma$ that consists in adding a symbol γ to the topmost order-1 stack, and $copy_k$ for $k \geq 1$ that consists in duplicating the topmost order-$k-1$ stack[2].

- For all order-n stack s with $n \geq 1$ and for all $\gamma \in \Gamma$, the operation $push_\gamma$ is defined on s by $push_\gamma(s) = s +\!\!+ \gamma$.
- For all order-n stack s and for all $1 \leq k \leq n$, the operation $copy_k$ is defined on s by $copy_k(s) = s +\!\!+ top_k(s)$.

Proposition 2.4.3 (The higher-order pop operation)**.**

Let s be an order-n stack s and $1 \leq i \leq n$ such that $top_{i+1}(s)$ (or s itself if $i = n$) contains at least two elements. There exists a unique order-n stack s' such that $s = s' +\!\!+ top_i(s)$. We write $pop_i(s)$ this unique stack.

In the following definition we summarise all order-n stacks operations.

Definition 2.4.14 (Operations of order n)**.**

Given $n \geq 1$, the set Op_n^{HOS} of ***operations on order-n stacks*** is defined as

$$Op_n^{\mathrm{HOS}} = \{push_\gamma \mid \gamma \in \Gamma, k \leq n\} \cup \{copy_k \mid k \leq n\} \cup \{pop_k \mid k \leq n\}.$$

We now add another feature to stacks: the possibility for a stack symbol inside a stack s, to point toward a substack of s as illustrated in Figure 2.8.

We define the set Γ_{links} of ***stack symbols with links*** as $\Gamma_{\mathrm{links}} = \Gamma \times \mathbb{N} \times \mathbb{N}$ where (γ, k, h) represents the symbol γ with a link to the h-th stack of order $k-1$ inside the stack of order-k containing the symbol (γ, k, h).

[2] Also written $push_k$ in the literature.

$$\big[[a\ b\ c]_1\ [d\ e]_1\ [f\ (g,2,1)]_1\big]_2 = \big[[a\ b\ c\ ]_1\ [d\ e]_1\ [f\ g]_1\big]_2$$

$$\Big[\big[\dots\big]_2\ \big[\dots\big]_2\big[\dots[\dots\ (d,3,2)]_1\big]_2\big[[\dots]_1\ \dots\ [\dots\ (g,2,1)]_1\big]_2\Big]_3 = \Big[\big[\dots\big]_2\ \big[\dots\big]_2\big[\dots[\dots\ d]_1\big]_2\big[[\dots]_1\ \dots\ [\dots\ g]_1\big]_2\Big]_3$$

Figure 2.8.: Graphical representation of collapsible stacks.

Definition 2.4.15 (Collapsible Stacks)**.**

The set $\mathrm{CS}_n(\Gamma)$ of ***order-n collapsible stacks*** is defined by $\mathrm{CS}_n(\Gamma) = \mathrm{Stacks}_n(\Gamma \uplus \Gamma_{\mathrm{links}})$. We write $\mathrm{CS}(\Gamma)$ for the set of all collapsible stacks, $\mathrm{CS}(\Gamma) = \bigcup_{n\geq 0} \mathrm{CS}_n(\Gamma)$.

Adding links enhance stacks with a new operation: the collapse operation that will be described later. As it is defined a link could point to nowhere, or could point forward in the stack. However starting from a stack reduced to one item, and applying the allowed operations, only produce stacks with backward links.

We add two new operations using stack symbols with links, the operation $push_{\gamma,k}$, that pushes the symbol γ with a link to the second topmost order-$(k-1)$ stack, and the operation *collapse* that is defined if the stack symbol top_1 has a link, and remove everything above the stack that is pointed. Formally given an order-n stack s:

- $push_{\gamma,k}(s)$, is defined if and only if $k \leq n$ and $top_{k+1}(s) = [s_1 \dots s_\ell]_k$ with $\ell > 1$, and is inductively defined by

$$push_{\gamma,k}([s_1 \dots s_\ell]_n) = [s_1 \dots push_{\gamma,k}(s_\ell)] \text{ if } n > k, \text{ and}$$

$$push_{\gamma,k}([s_1 \dots s_\ell]_k) = [s_1 \dots s_\ell]_k +\!\!+ (\gamma, k, \ell - 1).$$

- $collapse(s)$,is defined if and only if the symbol $top_1(s)$ is a stack symbol with links (γ, k, h), if $n \geq k$ and if $size(top_{k+1}) > h$.

$$collapse([s_1 \dots s_\ell]_n) = [s_1 \dots collapse(s_\ell)]_n \text{ if } n > k, \text{ and}$$

$$collapse([s_1 \dots s_\ell)]_k) = [s_1 \dots s_h].$$

In the following definition we summarise all operations defined on collapsible stacks. All the operations on stacks are illustrated in Figure 2.9.

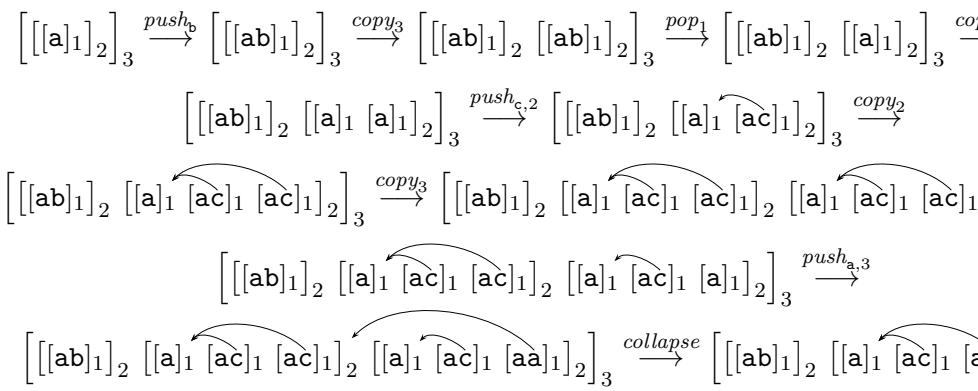


Figure 2.9.: A sequence of operations on collapsible stacks

Definition 2.4.16 (Operations of order-n collapsible stack)**.**

Given $n \geq 0$ the set Op_n of ***operations on order-n collapsible stacks*** is:

$$Op_n = \{collapse\} \cup \{push_{\gamma,k} \mid \gamma \in \Gamma, k \leq n\} \cup \{push_\gamma \mid \gamma \in \Gamma\} \cup \{copy_k \mid k \leq n\} \cup \{pop_k \mid k \leq n\}.$$

A collapsible pushdown automaton is a deterministic machine generating a possibly infinite tree labelled by a ranked alphabet Σ.

Definition 2.4.17 (Collapsible pushdown automaton)**.**

An ***order-n (deterministic) collapsible pushdown automaton (CPDA)*** is a tuple $\mathcal{C} = \langle \Sigma, \Gamma, Q, q_0, \gamma_0, \delta \rangle$ where

- Σ is a finite ranked alphabet,
- Γ is a finite set called the ***stack alphabet*** with $\gamma_0 \in \Gamma$ the ***bottom of the stacks***,
- Q is a finite set called the ***set of states*** with $q_0 \in Q$ the ***initial state***,
- δ is the ***transition function***, that maps a pair (q, γ) in $Q \times \Gamma$ to a tuple $\delta(q, \gamma)$ defined below:
  - either $\delta(q, \gamma) = (a, (q_1, op_1), \ldots, (q_k, op_k))$ with $a \in \Sigma$ of arity k, $q_1, \ldots, q_k \in Q$, $op_1, \ldots, op_k \in Op_n$,
  - or $\delta(q, \gamma) = (q', op)$ with $q' \in Q$, $op \in Op_n$.

In the following we fix a CPDA $\mathcal{C} = \langle \Sigma, \Gamma, Q, q_0, \gamma_0, \delta \rangle$ and describe how its associated tree is generated. We will define an infinite set of terms of order 0 and 1, and once again define a reduction relation $\to_\delta$, a $\perp$-transformation, and then define the value tree of the induced decorated ARS.

Definition 2.4.18 (Configurations)**.**

The set $Conf_{\mathcal{C}}$ of ***configurations*** of the CPDA $\mathcal{C}$ is defined as $Conf_{\mathcal{C}} = Q \times \mathrm{CS}_n(\Gamma)$. The ***initial configuration*** is $(q_0, [\ldots[\gamma_0]_1 \ldots]_n)$.

Definition 2.4.19 (Ranked Alphabet $\Sigma_{\mathcal{C}}$)**.**

The ranked alphabet $\Sigma_{\mathcal{C}}$ associated with $\mathcal{C}$ is defined by $\Sigma_{\mathcal{C}} = \Sigma \uplus Conf_{\mathcal{C}}$ where configurations have arity 0.

Definition 2.4.20 (Rewrite relation)**.**

The relation $\rightarrow_{\mathcal{C}}$ over $\mathcal{T}(\Sigma_{\mathcal{C}})$ is defined as $t \rightarrow_{\mathcal{C}} t'$ if there exists a context $C[\bullet]$ and a configuration (q, s) such that $t = C[(q, s)]$ and:

- either $\delta(q, \gamma) = (a, (q_1, op_1), \ldots, (q_k, op_k))$ with $\gamma = top_1(s)$, such that for all i $op_i(s)$ is defined and $t' = C[a\ (q_1, op_1(s))\ \ldots\ (q_k, op_k(s))]$,
- or $\delta(q, \gamma) = (q', op)$, such that $op(s)$ is defined and $t' = C[(q', op(s))]$.

Definition 2.4.21 (Reduction system)**.**

Given a term t in $\mathcal{T}(\Sigma_{\mathcal{C}})$ the tree $t^{\perp} \in \Sigma \uplus \{\perp : o\}$ is inductively defined by $(q, s)^{\perp} = \perp$ and $(a\ t_1 \ldots t_k)^{\perp} = a\ t_1^{\perp} \ldots t_k^{\perp}$.
We have that if $t \rightarrow_{\mathcal{C}} t'$ then $t^{\perp} \sqsubseteq (t')^{\perp}$. Furthermore the ARS $\langle \mathcal{T}(\Sigma_{\mathcal{C}}), \rightarrow_{\delta} \rangle$ is confluent. Then to any term t we associate the tree $\|t\|_{\mathcal{C}}$ as the maximum of all trees obtained by deriving t according to $\rightarrow_{\mathcal{C}}$. Finally the ***value tree*** $\|\mathcal{C}\|$ of $\mathcal{C}$ is defined as the value tree of the initial configuration, $\|\mathcal{C}\| = \|(q_0, [\ldots[\gamma_0]_1 \ldots]_n)\|_{\mathcal{C}}$.

The following theorem emphasises the link between HORS and CPDA, and has been used for designing decision procedures (see the next section).

Theorem 2.4.4 ([HMOS08, CS12])**.**

There exists an algorithm that takes as input an order-n CPDA $\mathcal{C}$ and outputs an order-n scheme $\mathcal{G}$ such that $\|\mathcal{C}\| = \|\mathcal{G}\|$.

There exists an algorithm that takes as input an order-n scheme $\mathcal{G}$ and outputs an order-n CPDA $\mathcal{C}$ such that $\|\mathcal{G}\| = \|\mathcal{C}\|$.

Example 2.4.2. We will describe a CPDA equivalent to the scheme of Example 2.3.2. Since the whole CPDA is very big, we will give the general idea and an example of derivation. The derivations of the CPDA will simulate the OI derivations of the scheme, therefore the configurations of the automaton will correspond to redexes of the scheme. In order to do so, we define the stack alphabet as the set of subterms appearing in the rewrite rules of the schemes, we use the stack to describe the recursive call of the non terminal, when the automaton wants to rewrite a non terminal, it simply pushes its production on top of the stack, when it deals with a terminal, it adds it to the tree, and

when it works with a variable it duplicates the stack to derive the term that the variable represents, and when it is done, it collapses the last stack to go back to the main term. In Figure 2.10 is depicted a derivation of such a scheme, note how the CPDA is actually deriving the terms of the scheme.

2.5. State of the art

A bit of history. Recursion schemes were introduced in the 70s as a model of computation, in order to describe the syntactical part of a functional program [Niv72, Cou78a, Cou78b, CN78]. Informally, an interpretation of a scheme is a domain *i.e.* a set with a functional interpretation for both the terminals and the nonterminals symbols such that the interpretation of a term is unchanged by application of the rule. Originally they only considered order-1 schemes. First they established that two schemes compute the same function under any interpretation if and only if they generate the same value tree [CN78]. Second they pointed out the equivalence between such schemes and deterministic pushdown automata (DPDA) [Cou78a, Cou78b]. As a consequence, Sénizergues' theorem [Sén97], that states the decidability of DPDA equivalence, implies the decidability of order-1 schemes equivalence. Higher-order schemes were later introduced to deal with functions taking functions as arguments [Ind76, Dam77a, Dam77b].The status of the general HORS equivalence problem remains open.

MSO Model checking. In the 2000s, higher-order recursive schemes were reintroduced as generators of infinite structures, more precisely infinite labelled trees. Knapik, Niwinski and Urzyczyn [KNU02] proved the decidability of MSO model-checking for the subclass of schemes satisfying the so called *safety* condition. The hierarchy of trees generated by such schemes was proven to be equal to the Caucal hierarchy [Cau02]. They showed that these schemes are equivalent to a class of higher order stack automata (which can be viewed as a subclass of CPDA), and proved the decidability of the MSO model checking of the trees generated by these automata. Later on, Ong proved, using the notion of *traversals*, the decidability of MSO model-checking for the whole class of trees generated by recursion schemes [Ong06]. Since then, this result has been proven again using different approaches. Hague, Murawski, Ong and Serre introduced collapsible pushdown automata (CPDA) as an extension of higher order pushdown automata, established the equivalence of schemes and CPDA, and then showed the MSO decidability by reduction to a parity game over a finite arena [HMOS08]. Kobayashi and Ong [KO09] built a type system in order to produce another parity game. Finally, Salvati and Walukiewicz have presented some order-preserving transformations between λ-terms and schemes, and used Krivine machines to establish the MSO decidability of λ-Y-calculus [SW11] again by reduction to parity games.

Embedding an interpretation in a HORS. A folk result states that there exists an interpretation of a scheme on the lattice domain $\{0, 1\}$ such that a ground term is interpreted as 0 if and only if its value tree is $\bot$.

Aehlig [Aeh06] and then Kobayashi [Kob09] addressed the safety property model checking for HORS where such a property is specified within a subclass of TAC automata:

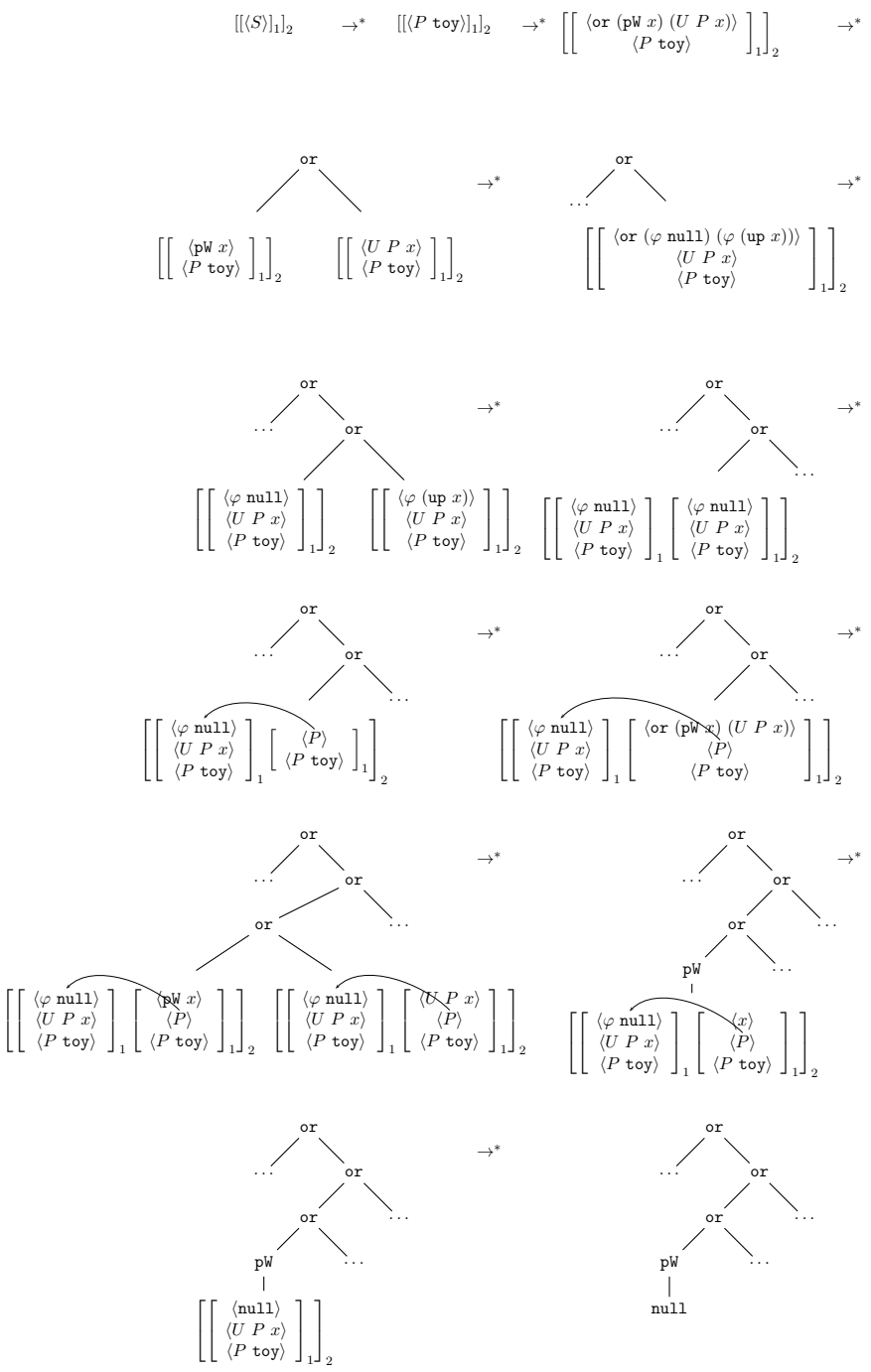


Figure 2.10.: A derivation of the CPDA of Example 2.10.

$\bot$-blind TAC automata. Such an automaton accepts the symbol $\bot$ from every state. Such a constraint disregards the unproductive branches of the value tree when evaluating the safety property. Given a scheme and a $\bot$-blind TAC automaton, they built an interpretation of the scheme on the domain of the subsets of the automaton states. The interpretation of the initial symbol is the set of states accepting the value tree of the scheme. Furthermore this interpretation is computable, which yields a decision procedure for the model checking problem.

Our contributions (Chapter 3). We developed this idea in [Had12] and showed that one can embed their construction into schemes. This embedding construction allowed to show the computability of the $\bot$-elimination of a scheme, and of the automata reflection for the class of automata they considered. In this article we used this construction to show the computability of the IO-simulation problem and we solved the OI-simulation problem with a standard transformation.

Given a (possibly non $\bot$-blind) TAC automaton, Salvati and Walukiewicz provided in [SW12] an interpretation for the λY-calculus, which associate with each ground typed λY-term the set of states that accepts the value tree of this term. As a consequence, one can use the embedding construction to provide a procedure for the TAC automata reflection.

Reflection and selection. Broadbent, Carayol, Ong and Serre proved the computability of MSO reflection in [BCOS10]. The technique they used relies on the equivalence between schemes and CPDA and on a notion of regularity for the winning region for parity games on CPDA. Afterwards, Carayol and Serre considered in [CS12] the selection problem and proved the computability of the MSO selection. The technique they used relies on the equivalence between schemes and CPDA.

Our contributions (Chapter 4). We presented new proofs of MSO reflection and selection for schemes, based on the model checking proof of [KO09]. The constructions we introduce preserve the structure of the scheme, which is not the case for the previous results.

2.6. Scheme transformations

In this section we informally introduce a general framework for the scheme transformations. These scheme transformations are *shape-preserving* in the sense that the structure of the new scheme is an extension of the original one, and that one can easily construct back the original scheme from the new scheme.

All the scheme transformations presented in this thesis will be obtained by *annotating* and *duplicating* some terms. More precisely, symbols in the new scheme will contain annotated versions of symbols in the original scheme, *i.e.* they will have the form x^θ

with x a symbol in the initial scheme and θ an annotation. Depending on whether we want the value tree of the new scheme to be annotated or not, we will either let the new ranked alphabet be the same as the old one, or be an annotated version of it.

In order for symbols to handle different annotated versions of their arguments we will also change the type, typically by increasing their arity. Let us formalise a little bit these ideas. Given a scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$, a transformation of $\mathcal{G}$ is usually given as follows.

Types. We associate with any type τ a integer n_τ that corresponds to how many times we will duplicate arguments of type τ in a term (if there are no duplication, $n_\tau = 1$). We also define a type transformation f usually defined by induction, defining $f(o)$ and then setting $f(\tau \to \tau') = f(\tau)^{n_\tau} \to f(\tau')$. In most cases $f(o) = o$. Note that this transformation is unambiguous and for a type τ', one can easily decide if there exists τ such that $\tau' = f(\tau)$ and in that case construct τ.

If $f(o) = o$ then a straightforward induction shows that for any type τ, $\mathit{order}(f(\tau)) = \mathit{order}(\tau)$. Since in most cases $f(o) = o$, we know that the order will be preserved in these transformations. We can generalise this observation by stating that if $\mathit{order}(f(o)) = n$ then for any type τ, $\mathit{order}(f(\tau)) = \mathit{order}(\tau) + n$.

Symbols. Given a set X of ***annotations***, we define to any symbol $\alpha : \tau$ and annotation $x \in X$, the symbol $\alpha^x : f(\tau)$. Sometimes we may define different sets of annotations and sometimes the set X is a singleton, simply used to distinguish the symbol α^x of type $f(\tau)$ from the symbol α of type τ.

The symbols of the new scheme will contain annotated versions of the symbols of $\mathcal{G}$ along with some other fresh symbols. As we said, for some problems, we want the value tree of the new scheme to be defined on the same ranked alphabet of the value tree of the initial scheme, in that case the new ranked alphabet contains Σ.

Term transformation. We then define, to any term $t : \tau$ of the original scheme, a set of terms of type $f(\tau)$, $\mathit{An}(t)$, of the new scheme, called the ***annotated versions of*** t. Usually for a symbol α, $\mathit{An}(\alpha)$ contains annotated versions of α, and for a term $t = t_1\ t_2$ with $t_1 : \tau' \to \tau$ and $t_2 : \tau'$, $\mathit{An}(t)$ contains terms of the form

$$s_1\ s_2^1 \dots s_2^\ell,$$

with $s_1 \in \mathit{An}(t_1)$, $\ell = n_{\tau'}$ and for all i, $s_2^i \in \mathit{An}(t_2)$. Note that since $s_1 : f(\tau' \to \tau) = f(\tau')^{n_{\tau'}} \to f(\tau)$ and for all i, $s_2^i : f(\tau')$, we have $s_1\ s_2^1 \dots s_2^\ell : f(\tau)$. We use the notation $\vec{s_2} : f(\tau)$ to denote a tuple of n_τ elements of type $f(\tau)$ and we write $s_1\ \vec{s_2}$ for the term $s_1\ s_2^1 \dots s_2^\ell$. Again remark that for all term s of the new scheme, one can decide if there exists t such that $s \in \mathit{An}(t)$ and if it is true, construct t.

Rewrite rules. For all rewrite rule $F\ x_1 \dots x_k \to e \in \mathcal{R}$, and for all annotated version F^x of F, we pick $e' \in \mathit{An}(e)$ and, if $f(o) = o$ we have $e' = o$ therefore we associate with F^x the rewrite rule

$$F^x\ \vec{x_1} \dots \vec{x_k} \to e'.$$

If $f(o)$ is different from o, for example if $f(o) = o \to o$, we use a fresh symbol $\mathtt{p}$ of type o and associate with F^x the rewrite rule

$$F^x\ \vec{x_1} \dots \vec{x_k} \to e'\ \mathtt{p}.$$

From the previous observations, one can easily construct back the original scheme from the new one by removing the annotations, the duplications, and applying f^{-1} to the types. The solutions of all the problems we introduced are based on such scheme transformations.

3. Alternative Semantics for Higher Order Recursion Schemes

3.1. Introduction 71
3.2. Morphisms 72
3.2.1. Presentation and properties 72
3.2.2. Embedding a morphism into a scheme 77
3.3. Extending scheme evaluation 84
3.3.1. Continuous mappings 85
3.3.2. Domain dependent value of a scheme 88
3.3.3. The domain of morphisms 91
3.3.4. The device morphism 93
3.3.5. The device morphism computes the domain dependent value 95
3.3.6. Applications 102
3.4. Simulation of the IO evaluation policy 104
3.4.1. Recognising IO-unproductive terms 104
3.4.2. The IO-simulation 111
3.5. Simulation of the OI evaluation policy 116

3.1. Introduction

In Chapter 2, we provided a particular semantics for HORS, namely the value tree. This semantics suffers a drawback: it is only defined for ground typed terms. In order to address this issue, we introduce *morphisms* for defining (full) semantics[1] for HORS. Informally, a morphism associates with any term a value in a (typed) set equipped with an application law compatible with the application law of terms (such set is called a *typed application system (TAS)*). Given a scheme, we require that such morphisms are *stable by rewrite*, *i.e.* the image of a term is unchanged by the application of any rule of the scheme. They can be viewed as semantics of a scheme.

From a computational point of view we are particularly interested in morphisms which associate with any type a finite set, and thus can be effectively described when ranging over a finite set of types. The first contribution of this chapter is a procedure that takes as input a scheme and a finite morphism stable by rewrite and produces another scheme which simulates the original scheme while computing the image of all subterms

[1] Note that the problem we look at is different (actually simpler) from the problem of giving a semantics for the λY-calculus.

occurring during a derivation by the morphism. We call such a scheme an *embedding* of the morphism in the initial scheme.

We consider the following problem: given a scheme $\mathcal{G}$, a TAS A, and a (typed preserving) mapping ζ from the terminal symbols to A, does there exists a morphism stable by rewrite that extends the mapping ζ? Such morphism may not exist, therefore inspired by λ-calculus results, we propose to characterise situations where there always exists such a morphism. The TAS A must fulfil the following requirements: (1) A^o must be a domain (in the sense of Scott), and $A^{\tau\to\tau'}$ is the set of Scott-continuous mappings from A^τ to $A^{\tau'}$. The continuity allows to define a morphism as a fixpoint of a morphism transformation.

Once this requirement is fulfilled, we develop an abstract construction of one of the possible morphisms. In a nutshell, it is defined as the smallest fixpoint of a mapping defined from the rewrite rules of $\mathcal{G}$. When the domain A^o is finite, this construction is effective. We apply this result to provide an original procedure for the reflection problem for $\bot$-blind TAC automata, and the $\bot$-elimination problem. The corresponding TAS is in the first case defined by $A^o = 2^Q$ where Q is the set of states of the automaton, and in the second case A^o is the two elements set $\{x_\bot, x_\top\}$.

In the last section we focus on the IO and OI simulation problems. Using a similar method as previously we construct a morphism on terms that can describe if the IO value tree of a term is equal to $\bot$. Then by embedding this morphism inside a scheme, one can construct a scheme whose value tree is equal to the IO value tree of the initial scheme, solving the IO simulation problem. Finally, we present another scheme transformation, unrelated with morphisms, that takes a scheme and produces another one whose IO value tree is equal to the value tree of the initial scheme, solving the OI simulation problem.

3.2. Morphisms

3.2.1. Presentation and properties

Building a full semantics for terms of a scheme requires to introduce typed applicative structures. Informally, it is a typed set with a binary law consistent with the types.

Definition 3.2.1 (Typed Applicative Structure)**.**

A ***typed applicative structures (TAS)***, is a tuple $\langle A, \cdot \rangle$ where:

- A is a typed set,
- $\cdot : A^2 \to A$ is a partial binary operation such that:
  $a \cdot b$ is defined only if there exists τ, τ' with $a : \tau \to \tau'$ and $b : \tau$.
  In this case $a \cdot b$ is typed by τ'.

Given $a, b \in A$ we write $a\ b$ the element $a \cdot b \in A$, and we consider that this notation is associative to the left, *i.e.* $a\ b\ c$ denotes $(a\ b)\ c$.

Example 3.2.1. Given a typed alphabet Γ, the set of typed terms over Γ, $\mathcal{T}(\Gamma)$, is a typed applicative structure where $t \cdot t'$ is the term $t\ t'$. Consistently, it is also called the ***free TAS generated by*** Γ.

Example 3.2.2. Let X be an arbitrary set. Then $\mathbf{Map}(X)$ is the typed set inductively defined as follows:

- $\mathbf{Map}(X)^o = X$,
- $\mathbf{Map}(X)^{\tau \to \tau'}$ is the set of total functions from $\mathbf{Map}(X)^\tau$ to $\mathbf{Map}(X)^{\tau'}$.

Then $\langle \mathbf{Map}(X), \cdot \rangle$ is a TAS where $f \cdot h = f(h)$.

Morphisms on TAS are defined, as usual, as mappings keeping the structure of the TAS: the types must be preserved and the morphism "commutes" with the application.

Definition 3.2.2 (Morphism)**.**

Let $\langle A, \cdot \rangle$ and $\langle B, \star \rangle$ be two TAS. A ***TAS morphism*** is a mapping $\varphi : A \to B$ such that:

- it is ***type preserving***, *i.e.* for all type τ and $a : \tau$, $\varphi(a) : \tau$,
- for all $a, b \in A$, if $a \cdot b$ is defined, then $\varphi(a) \star \varphi(b)$ is defined and

$$\varphi(a \cdot b) = \varphi(a) \star \varphi(b).$$

Example 3.2.2 continued. Let $x \in X$. Let φ_x be the mapping from $\mathbf{Map}(X)$ to $\mathbf{Map}(X)$ that maps any function $f : \tau_1 \to \cdots \to \tau_k \to o$ to the constant function of type

$\tau_1 \to \cdots \to \tau_k \to o$ returning x, *i.e.* for all $h_1 : \tau_1, \ldots, h_k : \tau_k$, $\varphi_x(f)\ h_1 \ldots h_k = x$. Then φ_x is a morphism. Indeed given $f : \tau \to \tau'$ and $h : \tau$, $\varphi(f)\ \varphi(h) = \varphi(f\ h)$ which is the constant function of type τ' returning x.

Let Γ be a typed alphabet, $\langle A, \cdot \rangle$ be a TAS, and $\varphi : \mathcal{T}(\Gamma) \to A$ be a morphism. Observe that φ is entirely defined by its restriction on the symbols of Γ. More formally for all type preserving mapping $f : \Gamma \to A$, there exists a unique morphism φ such that for all $\alpha \in \Gamma$, $\varphi(\alpha) = f(\alpha)$.

In the following we consider a scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$, and we study morphisms from $\mathcal{T}(\Sigma \uplus \mathcal{N})$ to some TAS $\langle A, \cdot \rangle$. We denote $\mathbf{Morph}(\mathcal{G}, A)$ those morphisms or simply **Morph** when the scheme and A are clear from the context.

Example 3.2.3. Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be defined by:

- $\mathcal{V} = \{y : o \to o, x : o\}$,
- $\Sigma = \{\mathtt{a} : o^2 \to o, \mathtt{b} : o, \mathtt{c} : o\}$,
- $\mathcal{N} = \{S : o, H : o \to o, J : o \to o, F : (o \to o) \to o\}$,
- $\mathcal{R}$ contains the following rewrite rules:

$$\begin{array}{lcl} S & \to & \mathtt{a}\ (F\ H)\ (F\ J) \\ H\ x & \to & \mathtt{a}\ x\ (H\ x) \\ J\ x & \to & \mathtt{a}\ (J\ x)\ (J\ x) \\ F\ y & \to & \mathtt{a}\ (y\ \mathtt{b})\ (y\ \mathtt{c}). \end{array}$$

The value tree of $\mathcal{G}$ is (partially) depicted in Figure 3.1. We write $[u, v]$ the mapping $f : \{0, 1\} \to \{0, 1\}$ such that $f(0) = u$ and $f(1) = v$ for all u, v. We define the morphism $\varphi \in \mathbf{Morph}(\mathcal{G}, \mathbf{Map}(\{0, 1\}))$ as follows (here $\vee$ denotes the boolean operator 'or'):

$$\begin{array}{lll} \varphi(\mathtt{b}) = 0 & \varphi(\mathtt{c}) = 1 & \varphi(S) = 1 \\ \varphi(\mathtt{a})\ u\ v = u \vee v & & \\ \varphi(H)\ u = u & & \varphi(J)\ u = 0 \\ \varphi(F)\ [u, v] = u \vee v. & & \end{array}$$

Let us evaluate φ on some terms, for example $\varphi(F\ H) = 1$ and $\varphi(\mathtt{a}\ \mathtt{b}\ (J\ \mathtt{c})) = 0$.

Since variables occur in schemes, we introduce the notion of environment[2] that enlarges morphisms with the evaluation of variables.

[2] Also called *valuation*.

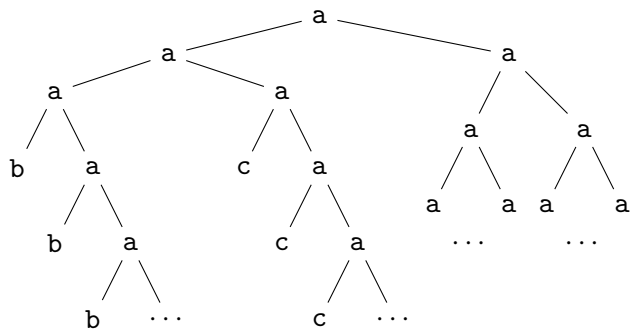


Figure 3.1.: The value tree of the scheme of example 3.2.3.

Definition 3.2.3 (Environment)**.**

Let $\mathcal{G}$ be a scheme and A be a TAS. An ***environment*** ρ is a partial mapping mapping from $\mathcal{V}$ to A which is type preserving.
Let $\varphi \in \mathbf{Morph}(\mathcal{G}, A)$ be a morphism, $x_1, \ldots, x_k$ be some variables, and ρ be an environment defined on $x_1, \ldots, x_k$. The morphism φ_ρ from $\mathcal{T}(\Sigma \uplus \mathcal{N} \uplus \{x_1, \ldots, x_k\})$ to A is defined by:

- for all $\alpha \in \Sigma \uplus \mathcal{N}$, $\varphi_\rho(\alpha) = \varphi(\alpha)$,
- for all i, $\varphi_\rho(x_i) = \rho(x_i)$.

The following proposition is a straightforward consequence of the previous definition.

Proposition 3.2.1.

Let $\mathcal{G}$ be a scheme, A be a TAS, $\varphi \in \mathbf{Morph}$, and ρ be an environment.
Assume that $x_1, \ldots, x_k \in \mathcal{V}$ and $t_1, \ldots, t_k \in \mathcal{T}(\Sigma \uplus \mathcal{N})$ with $\rho(x_i) = \varphi(t_i)$ for all i.
For all $e \in \mathcal{T}(\Sigma \uplus \mathcal{N} \uplus \{x_1, \ldots, x_k\})$,

$$\varphi(e_{[\forall i \ x_i \mapsto t_i]}) = \varphi_\rho(e).$$

Pursuing the goal of using morphisms to provide a semantics for a scheme, one need to constrain morphisms with respect to the rewrite rules. We introduce the notion of stability by rewrite. In a nutshell, when a morphism is stable, the evaluation of a non terminal F is equal to the evaluation of the production of its associated rewrite rule under any environment.

For later developments, we generalise the stability by the notion of compatibility between two morphisms.

Definition 3.2.4 (Compatibility between morphisms)**.**

Let $\mathcal{G}$ be a scheme, A be a TAS and $\varphi, \psi \in \mathbf{Morph}(\mathcal{G}, A)$ be two morphisms. The pair (φ, ψ) is ***compatible with*** $\rightarrow_{\mathcal{G}}$, written $\varphi \rightarrow_{\mathcal{G}} \psi$, if for all $a \in \Sigma$, $\varphi(a) = \psi(a)$ and for all $F\ x_1 \dots x_k \rightarrow e \in \mathcal{R}$ and for all environment ρ such that $x_1, \dots, x_k$ are in the domain of ρ,

$$\varphi_\rho(F\ x_1 \dots x_k) = \psi_\rho(e).$$

Intuitively, given two morphisms φ and ψ such that $\varphi \rightarrow_{\mathcal{G}} \psi$, and two terms t, t' such that $t \rightarrow t'$, $\varphi(t) = \psi(t')$.

Definition 3.2.5 (Stability by rewrite)**.**

Let $\mathcal{G}$ be a scheme, A be a TAS and $\varphi \in \mathbf{Morph}(\mathcal{G}, A)$. The morphism φ is ***stable by rewrite*** if $\varphi \rightarrow_{\mathcal{G}} \varphi$.
It is ***weakly stable by rewrite*** if for all $t, t' \in \mathcal{T}(\Sigma \uplus \mathcal{N})$ such that $t \rightarrow_{\mathcal{G}} t'$, $\varphi(t) = \varphi(t')$.

The next proposition relates the two notions of stability. Informally the weak stability is only concerned with values in A that are images of some terms.

Proposition 3.2.2.

Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$, A be a TAS, $\varphi \in \mathbf{Morph}(\mathcal{G}, A)$ be a morphism. If φ is stable by rewrite then it is weakly stable by rewrite.

Proof. Direct application of Proposition **3.2.1**. □

Note that the converse is not true in general, for example take the scheme on $\Sigma = \{\mathsf{c} : o\}$, with the non-terminals symbols $\mathcal{N} = \{S : o, F : (o \rightarrow o) \rightarrow o, H : ((o \rightarrow o) \rightarrow o) \rightarrow o\}$ and the variables $\mathcal{V} = \{x : o \rightarrow o, y : (o \rightarrow o) \rightarrow o\}$, with the following rewrite rules:

$$\begin{array}{rcl} S & \rightarrow & H\ F, \\ F\ x & \rightarrow & \mathsf{c}, \\ H\ y & \rightarrow & \mathsf{c}. \end{array}$$

Take the morphism φ in the TAS $\mathbf{Map}(\{0,1\})$ defined by $\varphi(S) = \varphi(\mathtt{c}) = 1$, for all $g : o \to o$, $\varphi(F)\ g = 0$ and for all $f : (o \to o) \to o$, $\varphi(H)\ f = 1$.

There are only five terms that one can construct on the alphabet $\Sigma \uplus \mathcal{N}$: S, $\mathtt{c}$, F, H, and $H\ F$, and there are only two possible rewrites: $S \to H\ F$ and $H\ F \to \mathtt{c}$. Since $\varphi(S) = \varphi(H\ F) = \varphi(\mathtt{c}) = 1$, φ is weakly stable by rewrite. However let $g : o \to o$, we have $\varphi(F)\ g = 0$, however $\varphi(\mathtt{c}) = 1$, therefore φ is not stable by rewrite.

Example 3.2.3 continued. One can check that the morphism φ is stable by rewrite. For example $\varphi(F)\ [1,0] = 1$ and if $\rho(y) = [1,0]$ we have $\varphi_\rho(\mathtt{a}\ (y\ \mathtt{b})\ (y\ \mathtt{c})) = \varphi_\rho(\mathtt{a})\ 1\ 0 = 1$.

Later, we will use morphisms on finite TAS (*i.e.* whose domain is finite) to *recognise*[3] some properties of a term, *e.g.* its value tree is infinite, its IO value tree is $\bot$, it satisfies some MSO formula, etc.

Definition 3.2.6 (Recognition by morphism)**.**

Let A be a TAC, $\mathcal{G}$ be a scheme, φ be a morphism in $\mathbf{Morph}(\mathcal{G}, A)$.
Given a property $\mathcal{P} : \mathcal{T}(\Sigma \uplus \mathcal{N}) \to \mathit{Bool}$, φ ***recognises*** $\mathcal{P}$ if there exists $A' \subseteq A$ such that for all term t, $\varphi(t) \in A'$ if and only if $\mathcal{P}(t)$ holds.

Example 3.2.3 continued. The morphism φ recognises the property "t has type o, and its value tree contains a $\mathtt{c}$", with the subset $A' = \{1\}$. Indeed, by induction on the size of the terms:

- $\|\mathtt{b}\| = \mathtt{b}$, $\|\mathtt{c}\| = \mathtt{c}$,
- $\|\mathtt{a}\ t_1\ t_2\|$ contains $\mathtt{c}$ if $\|t_1\|$ or $\|t_2\|$ contains $\mathtt{c}$, by induction it is equivalent to $\varphi(t_1) \vee \varphi(t_2)$.
- $\|S\|$, which is the value tree of the scheme, contains $\mathtt{c}$,
- $\|J\ t\|$ is equal to the tree that only contains $\mathtt{a}$ therefore it never contains $\mathtt{c}$,
- $\|H\ t\|$ contains $\mathtt{c}$ if and only if $\|t\|$ contains $\mathtt{c}$, by induction it is equivalent to $\varphi(t) = 0$,
- For all $t : o \to o$, $\|F\ t\|$ contains $\mathtt{c}$ if $\|t\ \mathtt{b}\|$ contains $\mathtt{c}$ or $\|t\ \mathtt{c}\|$ contains $\mathtt{c}$. By induction it is equivalent to say that $\varphi(t)$ is not the constant mapping equal to 0.

3.2.2. Embedding a morphism into a scheme

Notation. Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ be a scheme, $\langle \mathcal{D}, \cdot \rangle$ be a TAS, and $\varphi \in \mathbf{Morph}(\mathcal{G}, \mathcal{D})$ be a weakly stable morphism. In this section, for ease of exposition, we write $[\![t]\!] = \varphi(t)$.

[3] On words, it corresponds to the usual notion of recognition by morphism.

We assume that for all type τ, $\mathcal{D}^\tau$ is finite. Observe that the subset of types typing terms of the scheme is finite, therefore we assume without loss of generality that $\mathcal{D}$ is finite.

As discussed in the introduction, our goal is to build an enlarged scheme $\mathcal{G}'$ providing on the fly the evaluation of the terms w.r.t. φ occurring during a derivation of $\mathcal{G}$. We first present syntactical requirements for $\mathcal{G}'$. Intuitively each symbol of $\mathcal{G}'$ will carry a symbol of $\mathcal{G}$ along with a tuple of elements of $\mathcal{D}$.

Definition 3.2.7 (Enlarged schemes)**.**

Let $\mathcal{G}$ be a scheme and $\mathcal{D}$ be a finite TAS. The tuple $\langle \mathcal{G}', \mathit{symb}, \mathit{blocks}, \theta \rangle$ ***enlarges*** $\mathcal{G}$ with respect to $\mathcal{D}$ if the following holds.

1. $\mathcal{G}'$ is a scheme.
2. symb is a mapping from $\Sigma' \uplus \mathcal{N}'$ to $\Sigma \uplus \mathcal{N}$ such that for all $\alpha' \in \Sigma'$ (resp. $\alpha' \in \mathcal{N}'$), $\mathit{symb}(\alpha') \in \Sigma$ (resp. $\mathit{symb}(\alpha') \in \mathcal{N}$).
3. blocks is a mapping from symbols of $\mathcal{G}'$ to tuples of integers such that for all $\alpha' \in \Sigma' \uplus \mathcal{N}'$, $\mathit{blocks}(\alpha') = (n_1^{\alpha'}, \ldots, n_k^{\alpha'})$ with $k = \mathit{arity}(\mathit{symb}(\alpha'))$ such that $\mathit{arity}(\alpha') = n_1^{\alpha'} + \ldots + n_k^{\alpha'}$.
4. θ, called the ***annotation*** of $\mathcal{G}'$, is a mapping from $\Sigma' \uplus \mathcal{N}'$ to $\mathcal{D}^*$ such that for all $\alpha' \in \Sigma' \uplus \mathcal{N}'$, $\theta(\alpha') \in \mathcal{D}^k$ with $k = \mathit{arity}(\mathit{symb}(\alpha'))$.

In the following, once $\langle \mathcal{G}', \mathit{symb}, \mathit{blocks}, \theta \rangle$ is fixed, we refer to it as $\mathcal{G}'$.

The following definition explicits the relation between $\mathcal{G}$ and $\mathcal{G}'$ by giving a partial mapping from the terms of the enlarged scheme to the terms of the initial one. The main restriction consists of requiring that the arguments of a block should be all present or absent, and when present should be mapped to the same term.

Definition 3.2.8 (Projection of terms)**.**

Let $\mathcal{G}$ and $\mathcal{G}'$ be a scheme, $\langle \mathcal{D}, \cdot \rangle$ be a finite TAS such that $\mathcal{G}'$ enlarges $\mathcal{G}$ with respect to $\mathcal{D}$. The ***projection from $\mathcal{G}'$ to $\mathcal{G}$*** is the partial mapping π from $\mathcal{T}(\Sigma' \uplus \mathcal{N}')$ to $\mathcal{T}(\Sigma \uplus \mathcal{N})$ inductively defined as follows.
Let $t = \alpha'\ t_1 \dots t_{k'} \in \mathcal{T}(\Sigma' \uplus \mathcal{N}')$. Let $\alpha = symb(\alpha')$ and let $\tau_1 \to \dots \to \tau_k \to o$ be the type of α. Then $\pi(t)$ is defined if the following holds.

- There exists $j \leq k$, such that $k' = n_1^{\alpha'} + \dots + n_j^{\alpha'}$. We write
$$t = \alpha'\ t_{1,1} \dots t_{1,n_1} \dots t_{j,1} \dots t_{j,n_j}.$$
- For all $i \leq j$, there exists $s_i : \tau_i$ such that for all $\ell \leq n_i$, $\pi(t_{i,\ell})$ is defined and is equal to s_i.

When defined, $\pi(t) = \alpha\ s_1 \dots s_j$.

We say that a term $t \in \mathcal{T}(\Sigma' \uplus \mathcal{N}')$ is ***well formed*** if $\pi(t)$ is defined. The projection above defines a structural relation between terms. However the main interest of an enlarged scheme is to capture the evaluation of the terms by morphism φ. With the help of the annotation θ, one can semantically relate the terms. Observe that for technical reasons, the annotation provides the evaluation of the arguments rather than the evaluation of the term itself. This will be sufficient for further results.

Definition 3.2.9 (Correct annotation)**.**

Let $t = \alpha'\ t_{1,1} \dots t_{j,n_j}$ be a well formed term, $\pi(t) = \alpha\ s_1 \dots s_j$ and $\theta(t) = (d_1, \dots, d_k)$ with $k = arity(\alpha)$. Then t is ***correctly annotated*** if for all $i \leq j$:

- $[\![s_i]\!] = d_i$,
- for all $\ell \leq n_i$, $t_{i,\ell}$ is correctly annotated.

In order to achieve the definition of an embedding scheme, it remains to ensure that the dynamics of the rewrite rules preserves the correct annotation. Since we do not want to deal with non-determinism, we only consider the parallel rewriting.

Definition 3.2.10.

Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be a scheme, $\varphi \in \mathbf{Morph}(\mathcal{G}, \mathcal{D})$ be a weakly stable morphism, and $\langle \mathcal{G}', \mathit{symb}, \mathit{blocks}, \theta \rangle$ be an enlarged scheme with respect to $\mathcal{G}$ and φ. Then $\mathcal{G}'$ ***embeds*** φ in $\mathcal{G}$ if:

- $\pi(S') = S$,
- for all correctly annotated term t, if $t \Rightarrow_{\mathcal{G}'} t'$ then t' is well formed and correctly annotated and $\pi(t) \Rightarrow_{\mathcal{G}} \pi(t')$.

The problem we solve is the following one.

Problem 7 (The embedding problem).
Input: *A scheme $\mathcal{G}$, a morphism $\varphi \in \mathbf{Morph}(\mathcal{G}, \mathcal{D})$ where $\mathcal{D}$ is finite.*
Output: *An enlarged scheme $\mathcal{G}'$ that embeds φ.*

We first explain the principle of our procedure before defining it formally. With a symbol α of arity k in the initial scheme, we associate a set of annotated symbols of the form $\alpha^{d_1,\ldots,d_k}$ in the enlarged scheme, where $d_1, \ldots, d_k \in \mathcal{D}$. Every symbol $\alpha^{d_1,\ldots,d_k}$ is annotated with $\theta(\alpha^{d_1,\ldots,d_k}) = (d_1, \ldots, d_k)$. Such a symbol will be produced as a head of a term t' corresponding to a term $t = \alpha\ t_1 \ldots t_k$ produced in the original scheme with $[\![t_i]\!] = d_i$ for all i.

When α is a non terminal, its associated rewrite rule should mimic the rule associated with α in the original scheme. However, an argument x of the original rule may appear several times in the right hand part of the rule with partial application and different arguments. Thus the symbols $x^{d_1,\ldots,d_k}$ occurring in place of the original symbol x may be different and potentially may range over all possible annotations $d_1, \ldots, d_k$ of the original symbol. This requires that the mimicking rule has for arguments all the $x^{d_1,\ldots,d_k}$.

Example 3.2.3 continued. We construct a scheme $\mathcal{G}' = \langle \mathcal{V}', \Sigma', \mathcal{N}', S, \mathcal{R}' \rangle$ that embeds φ into $\mathcal{G}$.

- $\mathcal{V}' = \{x : o,\ y^0, y^1 : o \to o\}$,
- $\Sigma' = \{\mathtt{a}^{0,0}, \mathtt{a}^{0,1}, \mathtt{a}^{1,0}, \mathtt{a}^{1,1} : o^2 \to o,\ \mathtt{b}, \mathtt{c} : o\}$,
- $\mathcal{N}' = \{S : o,\ H^0, H^1, J^0, J^1 : o,\ F^{[0,0]}, F^{[0,1]}, F^{[1,0]}, F^{[1,1]} : (o \to o)^2 \to o\}$,

- $\mathcal{R}'$ contains the following rewrite rules:

$$\begin{array}{lcl}
S & \to & \mathtt{a}^{1,0}\ (F^{[0,1]}\ H^0\ H^1)\ (F^{[0,0]}\ J^0\ J^1) \\
\\
H^0\ x & \to & \mathtt{a}^{0,0}\ x\ (H^0\ x) \\
H^1\ x & \to & \mathtt{a}^{1,1}\ x\ (H^1\ x) \\
\\
J^0\ x & \to & \mathtt{a}^{0,0}\ (J^0\ x)\ (J^0\ x) \\
J^1\ x & \to & \mathtt{a}^{0,0}\ (J^1\ x)\ (J^1\ x) \\
\\
F^{[0,0]}\ y^0\ y^1 & \to & \mathtt{a}^{0,0}\ (y^0\ \mathtt{b})\ (y^1\ \mathtt{c}) \\
F^{[0,1]}\ y^0\ y^1 & \to & \mathtt{a}^{0,1}\ (y^0\ \mathtt{b})\ (y^1\ \mathtt{c}) \\
F^{[1,0]}\ y^0\ y^1 & \to & \mathtt{a}^{1,0}\ (y^0\ \mathtt{b})\ (y^1\ \mathtt{c}) \\
F^{[1,1]}\ y^0\ y^1 & \to & \mathtt{a}^{1,1}\ (y^0\ \mathtt{b})\ (y^1\ \mathtt{c}).
\end{array}$$

Let us explain how the rewrite rule related to $F^{[0,1]}$ has been produced. Recall the original rule:

$$F\ y \to \mathtt{a}\ (y\ \mathtt{b})\ (y\ \mathtt{c}).$$

The first occurrence of y is applied to $\mathtt{b}$, therefore it should be annotated with $\varphi(\mathtt{b}) = 0$. Similarly, the second occurrence of y should be annotated with $\varphi(\mathtt{c}) = 1$. This justifies the occurrence of y^0 and y^1 on the left hand part of the rule.

The annotation $[0, 1]$ means that this rule will be applied to an argument whose evaluation is the mapping $[0, 1]$, thus the evaluation of $y\ \mathtt{b}$ is equal to $[0, 1]\ \varphi(\mathtt{b}) = [0, 1]\ 0 = 0$, and by a similar reasoning the evaluation of $y\ \mathtt{c}$ is equal to 1. Therefore the occurrence of $\mathtt{a}$ should be annotated with $(0, 1)$.

In the general case, this procedure can be iterated inductively upward from the inner subterms to the term.

Example 3.2.4. To illustrate what happens when some arguments are not fully applied let us add to the scheme of Example 3.2.3 the variable $z : o^2 \to o$ and the non terminal $J : (o^2 \to o) \to o$ associated with the following rewrite rule,

$$J\ z\ \to\ a\ (z\ \mathtt{b}\ \mathtt{c})\ (F\ (z\ \mathtt{b})).$$

In this example, the second occurrence of z is not fully applied, since $(z\ \mathtt{b})$ has type $o \to o$. Given $d : o^2 \to o \in \mathbf{Map}(\{0, 1\})$, the rewrite rule associated to J^d is the following:

$$J^d\ z^{0,0}\ z^{0,1}\ z^{1,0}\ z^{1,1}\ \to\ a^{(d\ 0\ 1),(\varphi(F)\ (d\ 0))}\ (z^{0,1}\ \mathtt{b}\ \mathtt{c})\ (F^{d\ 0}\ (z^{0,0}\ \mathtt{b})\ (z^{0,1}\ \mathtt{b})).$$

Recall that $F^{d\ 0}$ requires two arguments while F requires one in the original scheme. The two arguments, represented in the rewrite rule of $F^{d\ 0}$ by y^0 and y^1 describe "y expecting an argument of value 0" and "y expecting an argument of value 1". Therefore in the rewrite rule of J^d we need to represent "$z\ \mathtt{b}$ expecting an argument of value 0"

and "z $\mathtt{b}$ expecting an argument of value 1". They will have the form $z^{u,v}\ \mathtt{b} : o \to o$ with $u, v \in \{0, 1\}$. The first argument of this $z^{u,v}$ will always be $\mathtt{b}$ therefore we know that $u = \varphi(\mathtt{b}) = 0$. This leaves two possible annotations of z : $z^{0,0}$ and $z^{0,1}$, therefore $z^{0,0}\ \mathtt{b}$ (resp. $z^{0,1}\ \mathtt{b}$) represents "z $\mathtt{b}$ expecting an argument of value 0 (resp. 1)".

We now formalise the construction.

Types. With a type $\tau = \tau_1 \to \cdots \to \tau_k \to o$ we associate the integer

$$n_\tau = Card\Big(\big\{(d_1, \ldots, d_k) \mid \forall i\ d_i \in \mathcal{D}^{\tau_i}\big\}\Big)$$

and a complete ordering of $\{(d_1, \ldots, d_k) \mid \forall i\ d_i \in \mathcal{D}^{\tau_i}\}$ denoted $\mathbf{d}^\tau_1, \mathbf{d}^\tau_2 \ldots, \mathbf{d}^\tau_{n_\tau}$.

Given a type $\tau = \tau_1 \to \cdots \to \tau_k \to o$, the type τ^+ is defined inductively by

$$(\tau_1 \to \cdots \to \tau_k \to o)^+ = (\tau_1^+)^{n_{\tau_1}} \to \cdots \to (\tau_k^+)^{n_{\tau_k}} \to o.$$

Since $n_o = 1$, $(o^k \to o)^+ = o^k \to o$.

Symbols. With a terminal $a : o^k \to o$ and a tuple $d_1, \ldots, d_k \in \mathcal{D}^o$, we associate the terminal $a^{d_1,\ldots,d_k} : o^k \to o \in \Sigma'$.

With a non terminal $F : \tau_1 \to \cdots \to \tau_k \to o$ (resp. a variable $x : \tau_1 \to \cdots \to \tau_k \to o$) and a tuple $d_1 : \tau_1, \ldots, d_k : \tau_k$, we associate the non-terminal

$$F^{d_1,\ldots,d_k} : \tau_1^{n_{\tau_1}} \to \cdots \to \tau_k^{n_{\tau_k}} \to o \in \mathcal{N}'$$

(resp. the variable $x^{d_1,\ldots,d_k} : \tau_1^{n_{\tau_1}} \to \cdots \to \tau_k^{n_{\tau_k}} \to o \in \mathcal{V}'$).

Term transformation. Let $x_1, \ldots, x_\ell$ be some variables, $\tau = \tau_1 \to \cdots \to \tau_k \to o$ be a type, $t \in \mathcal{T}(\Sigma \uplus \mathcal{N} \uplus \{x_1, \ldots, x_\ell\})$ be a term of type τ, ρ be an environment such that for all i, $x_i \in dom(\rho)$, and $d_1 : \tau_1, \ldots, d_k : \tau_k$ be elements of $\mathcal{D}$.

The term $t_\rho^{+d_1,\ldots,d_k} : \tau^+ \in \mathcal{T}(\mathcal{V}' \uplus \Sigma' \uplus \mathcal{N}')$ is inductively defined on the size of the term by:

- if $t = \alpha \in \mathcal{V} \uplus \Sigma \uplus \mathcal{N}$, $t_\rho^{+d_1,\ldots,d_k} = \alpha^{d_1,\ldots,d_k}$,
- if $t = t_1\ t_2$ with $t_1 : \tau' \to \tau$ and $t_2 : \tau'$. Let $d = \varphi_\rho(t_2)$. Remark that ${t_1}_\rho^{+d,d_1,\ldots,d_k}$ has type $(\tau'^+)^{n_{\tau'}} \to \tau^+$. We define

$$(t_1\ t_2)_\rho^{+d_1,\ldots,d_k} = {t_1}_\rho^{+d,d_1,\ldots,d_k}\ {t_2}_\rho^{+\mathbf{d}^{\tau'}_1}\ \ldots\ {t_2}_\rho^{+\mathbf{d}^{\tau'}_{n_{\tau'}}}.$$

Note that since this transformation is only duplicating and annotating, given a term $t^{+d_1,\ldots,d_k}$ we can uniquely find the unique term t associated to it. In the same way, a tree on Σ' is simply an annotated tree on Σ then we let $Unlab(t)$ be the tree on Σ consisting in the unlabeling of the tree t on Σ'.

Rewrite rules. Let $F : \tau_1 \to \cdots \to \tau_k \to o$ be a non terminal, $d_1 : \tau_1, \ldots, d_k : \tau_k$ be elements of $\mathcal{D}$, and $\rho = x_1 \mapsto d_1, \ldots, x_k \mapsto d_k$. If $F\ x_1 \ldots x_k \ \to\ e \in \mathcal{R}$, we define in $\mathcal{R}'$ the rule

$$F^{d_1,\ldots,d_k}\ x_1^{\mathbf{d}_1^{\tau_1}}\ \ldots\ x_1^{\mathbf{d}_{n_{\tau_1}}^{\tau_1}}\ \ \ldots\ \ x_k^{\mathbf{d}_1^{\tau_k}}\ \ldots\ x_k^{\mathbf{d}_{n_{\tau_k}}^{\tau_k}}\ \ \to\ \ e_\rho^+.$$

Theorem 3.2.3 (Correctness of the embedding)**.**

For all $t, t' : o \in \mathcal{T}(\Sigma \uplus \mathcal{N})$, if $t \Rightarrow t'$, then $t^+ \Rightarrow (t')^+$.
In particular $\mathit{Unlab}(\|\mathcal{G}'_{t^+}\|) = \|\mathcal{G}_t\|$.

Proof. **Claim 3.2.4.** Let $F : \tau_1 \to \cdots \to \tau_k \to o \in \mathcal{N}$, $t_1 : \tau_1, \ldots, t_k : \tau_k \in \mathcal{T}(\Sigma \uplus \mathcal{N})$, $F\ x_1 \ldots x_k \to e \in \mathcal{R}$ and $\rho = x_1 \mapsto [\![t_1]\!], \ldots, x_k \mapsto [\![t_k]\!]$. Then

$$(e_{[\forall i\ x_i \mapsto t_i]})^+ = e^+_{\left[\forall i,j\ x_i^{\mathbf{d}_j^{\tau_i}} \mapsto t_i^{\mathbf{d}_j^{\tau_i}}\right]}.$$

⌟

Proof. Straightforward induction on the structure of e. □

We prove a slightly more general result that the one stated in the theorem: given two terms $t, t' : \tau'_1 \to \cdots \to \tau'_\ell \to o \in \mathcal{T}(\Sigma \uplus \mathcal{N})$, and $d_1 : \tau'_1, \ldots, d_\ell : \tau'_\ell \in \mathcal{D}$, if $t \Rightarrow t'$, then $t^{+d_1,\ldots,d_\ell} \Rightarrow (t')^{+d_1,\ldots,d_\ell}$. We prove this result by induction on the structure of t.

Assume that $t = \alpha\ t_1, \ldots, t_k$ with $\alpha \in \Sigma \uplus \mathcal{N}$.

- If $\alpha \in \Sigma$ or $\alpha \in \mathcal{N}$ and $k < \mathit{arity}(\alpha)$. Then $t' = \alpha\ t'_1 \ldots t'_k$ such that for all i, $t_i \Rightarrow t'_i$. Therefore

  $$t^{+d_1,\ldots,d_\ell} = \alpha^{[\![t_1]\!],\ldots,[\![t_k]\!],d_1,\ldots,d_\ell}\ \ t_1^{+\mathbf{d}_1^{\tau_1}}\ \ldots\ t_1^{+\mathbf{d}_{n_{\tau_1}}^{\tau_1}}\ \ \ldots\ \ t_k^{+\mathbf{d}_1^{\tau_k}}\ \ldots\ t_k^{+\mathbf{d}_{n_{\tau_k}}^{\tau_k}}.$$

  In this term, the head α is also not fully applied, then let s be such that $t^{+d_1,\ldots,d_\ell} \Rightarrow s$, we have

  $$s = \alpha^{[\![t_1]\!],\ldots,[\![t_k]\!],d_1,\ldots,d_\ell}\ \ s_1^1 \ldots\ s_1^{n_{\tau_1}}\ \ \ldots\ \ s_k^1 \ldots\ s_k^{n_{\tau_k}},$$

  with for all i, j, $t_i^{+\mathbf{d}_j^{\tau_i}} \Rightarrow s_i^j$. Then by induction hypothesis for all i, j we have $s_i^j = t_i'^{+\mathbf{d}_j^{\tau_i}}$. Furthermore, since by weak stability of φ $[\![t_i]\!] = [\![t'_i]\!]$ for all i, $s = (t')^{+d_1,\ldots,d_\ell}$.

- If $\alpha \in \mathcal{N}$ and $k = \mathit{arity}(\alpha)$ (in particular $\ell = 0$). Take $\alpha\ x_1 \ldots x_k \to e \in \mathcal{R}$. We have $t' = e_{\left[\forall i\ x_i \mapsto t'_i\right]}$ with $t_i \to t'_i$. Let s be such that $t^+ \Rightarrow s$. Then

  $$s = e^+_{\rho\left[\forall i,j\ x_i^{\mathbf{d}_j^{\tau_i}} \mapsto s_i^j\right]}$$

with $t_i^{+\mathbf{d}_j^{\tau_i}} \Rightarrow s_i^j$, and $\rho = x_1 \mapsto [\![t_1]\!], \ldots, x_k \mapsto [\![t_k]\!]$. Then by induction hypothesis for all i, j we have $s_i^j = t_i'^{+\mathbf{d}_j^{\tau_i}}$. Claim **3.2.4** states then that

$$s = (e_{[\forall i\ x_i \mapsto t_i]})^+ = (t')^+.$$

□

Application. Let φ be a morphism and assume that the modeller of the functional program forbids some functions to be called with specific value of arguments. Once the embedding scheme is built, it can be modified in order to stop the derivation when a term corresponds to the forbidden situation.

Example 3.2.3 continued. We want to transform the scheme in order to forbid the derivation of a subterm when its associated value tree will not include any $\mathtt{c}$. For instance, the occurrence of a $\mathtt{c}$ would correspond to a completed service, and thus such a situation witnesses a useless derivation. In the embedded scheme, this can be detected by applying the evaluation of the head over its annotation. For instance $F^{[0,0]}t_1\ t_2$ is the annotation of a term $F\ t$ whose value is $\varphi(F)\ [0,0] = 0$. Therefore we might turn the rule associated to $F^{[0,0]}$ into

$$F^{[0,0]}y_1\ y_2 \to \text{FORBIDDEN},$$

where FORBIDDEN : o is a new terminal added to the scheme. Here is the whole set of rewrite rules transformed this way.

$$\begin{array}{lcl}
S & \to & \mathtt{a}^{1,0}\ (F^{[0,1]}\ H^0\ H^1)\ (F^{[0,0]}\ J^0\ J^1) \\
\\
H^0\ x & \to & \text{FORBIDDEN} \\
H^1\ x & \to & \mathtt{a}^{1,1}\ x\ (H^1\ x) \\
\\
J^0\ x & \to & \text{FORBIDDEN} \\
J^1\ x & \to & \text{FORBIDDEN} \\
\\
F^{[0,0]}\ y^0\ y^1 & \to & \text{FORBIDDEN} \\
F^{[0,1]}\ y^0\ y^1 & \to & \mathtt{a}^{0,1}\ (y^0\ \mathtt{b})\ (y^1\ \mathtt{c}) \\
F^{[1,0]}\ y^0\ y^1 & \to & \mathtt{a}^{1,0}\ (y^0\ \mathtt{b})\ (y^1\ \mathtt{c}) \\
F^{[1,1]}\ y^0\ y^1 & \to & \mathtt{a}^{1,1}\ (y^0\ \mathtt{b})\ (y^1\ \mathtt{c}).
\end{array}$$

The value true of this new scheme is depicted in Figure 3.2.

3.3. Extending scheme evaluation

Now we look at the following question. Given a type preserving mapping $\zeta : \Sigma \to A$, can we obtain a morphism stable by rewrite φ such that for all $a \in \Sigma$, $\varphi(a) = \zeta(a)$? This problem corresponds to the semantic problem of a recursive program: given an

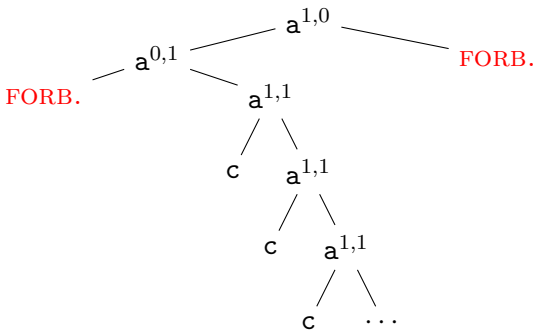


Figure 3.2.: The value tree of the transformed scheme.

interpretation of the built-in functions, can we give an interpretation of the user defined function such that the interpretation of a function agrees with its definition?

First we characterise, thanks to Scott-domains a situation where the answer is positive. Then given such a domain, we generalise the notion of value tree of a ground typed term to a notion of ***domain dependent value***. Finally we construct a morphism such that the evaluation of a ground typed term by this morphism is equal to its domain dependent value.

3.3.1. Continuous mappings

We define in this section the TAS of *continuous mappings* based on *domains*. A domain is a partially ordered set with a smallest element, and such that any monotonic sequence has a limit in the domain[4].

Definition 3.3.1 (Domain)**.**

A ***domain*** is a partially ordered set $\langle X, \sqsubseteq \rangle$ such that:

1. for any sequence $x_1 \sqsubseteq x_2 \sqsubseteq \cdots$, there exists $x_\infty \in X$ such that x_∞ is the supremum of the sequence,
2. there exists a minimum $x_\perp$ of X.

For example, given a ranked alphabet Σ the set of trees over Σ is a domain, where $\perp$ is the smallest element and $t \sqsubseteq t'$ is the order on trees.

In the following, as long as there is no ambiguity, we will use the symbol $\sqsubseteq$ to denote the order of any domain.

[4]In particular, complete partial orders are domains

A continuous mapping between two domains is a monotonic[5] mapping that preserves the limits.

Definition 3.3.2 (Continuous mappings)**.**

Let $\langle X, \sqsubseteq_1 \rangle$ and $\langle Y, \sqsubseteq_2 \rangle$ be two domains. A mapping f from X to Y is ***continuous*** if it satisfies the following conditions.

- For all $x_1 \sqsubseteq_1 x_2$ in X, $f(x_1) \sqsubseteq_2 f(x_2)$ (***monotonicity***).
- For all sequence $x_1 \sqsubseteq_1 x_2 \sqsubseteq_1 \cdots$, let x_∞ be its limit, and y_∞ be the limit of the sequence $f(x_1) \sqsubseteq_2 f(x_2) \sqsubseteq_2 \cdots$. Then $y_\infty = f(x_\infty)$ (***limit preservation***).

We denote $\mathit{Cont}(X, Y)$ the set of continuous mappings from $\langle X, \sqsubseteq \rangle$ to $\langle Y, \sqsubseteq \rangle$.

The following definitions and properties show that $\mathit{Cont}(X, Y)$ is also a domain.

Definition 3.3.3 (The relation $\sqsubseteq_{\mathit{Cont}}$)**.**

Let $\langle X, \sqsubseteq_1 \rangle$ and $\langle Y, \sqsubseteq_2 \rangle$ be two domains. For all $f, g \in \mathit{Cont}(X, Y)$, $f \sqsubseteq_{\mathit{Cont}} g$ if for all $x_1 \sqsubseteq_1 x_2$ in X, $f(x_1) \sqsubseteq_2 g(x_2)$.

In the following, we always use the symbol $\sqsubseteq$ to denote partial orders, unless it raises ambiguity.

Proposition 3.3.1.

Let $\langle X, \sqsubseteq \rangle$ and $\langle Y, \sqsubseteq \rangle$ be two domains. The relation $\sqsubseteq$ is a partial order over $\mathit{Cont}(X, Y)$.

Proof.
Reflexivity. The reflexivity is exactly the monotonicity.
Transitivity. Take $f_1 \sqsubseteq f_2 \sqsubseteq f_3 \in \mathit{Cont}(X, Y)$ and $x_1 \sqsubseteq x_2 \in X$. By reflexivity we have $x_1 \sqsubseteq x_1$ therefore $f_1(x_1) \sqsubseteq f_2(x_1) \sqsubseteq f_3(x_2)$ therefore $f_1 \sqsubseteq f_3$.
Antisymmetry. Take $f_1 \sqsubseteq f_2 \sqsubseteq f_1 \in \mathit{Cont}(X, Y)$ and $x \in X$. By reflexivity we have $x \sqsubseteq x$ therefore $f_1(x) \sqsubseteq f_2(x) \sqsubseteq f_1(x)$. By antisymmetry of $\langle Y, \sqsubseteq \rangle$, we have $f_1(x) = f_2(x)$, therefore $f_1 = f_2$. □

[5] In domain theory, a monotonic mapping is a monotonically increasing mapping in the usual sense.

Once the order has been defined, we introduce the limit of a monotonic sequence of continuous mapping.

Definition 3.3.4 (Limit mapping)**.**

Given a sequence $f_1 \sqsubseteq f_2 \sqsubseteq \cdots \in \mathit{Cont}(X, Y)$, the ***limit*** f_∞ of the sequence is the mapping from X to Y, defined by letting for all $x \in X$, $f_\infty(x)$ be the limit of the monotonic sequence $f_1(x) \sqsubseteq f_2(x) \sqsubseteq \cdots$.

Proposition 3.3.2.

Let $\langle X, \sqsubseteq \rangle$ and $\langle Y, \sqsubseteq \rangle$ be two domains. Given a sequence $f_1 \sqsubseteq f_2 \sqsubseteq \cdots$ in $\mathit{Cont}(X, Y)$, its limit f_∞ is continuous.

Proof. Take $x_1 \sqsubseteq x_2 \in X$. For all i we have $f_i(x_1) \sqsubseteq f_i(x_2) \sqsubseteq f_\infty(x_2)$, therefore $f_\infty(x_2)$ is an upper bound of the sequence $f_1(x_1) \sqsubseteq f_2(x_1) \sqsubseteq \cdots$, and since $f_\infty(x_1)$ is the least upper bound of this sequence, we have $f_\infty(x_1) \sqsubseteq f_\infty(x_2)$. Hence f_∞ is monotonic.

Let $x_1 \sqsubseteq x_2 \sqsubseteq \cdots$ be a sequence of elements of X, and let x_∞ be its limit. We need to show that the limit of $f_\infty(x_1) \sqsubseteq f_\infty(x_2) \sqsubseteq \cdots$ is $f_\infty(x_\infty)$. It is clearly an upper bound since for all i, $x_i \sqsubseteq x_\infty$ therefore $f_\infty(x_i) \sqsubseteq f_\infty(x_\infty)$.

Let y be an upper bound of the sequence $f_\infty(x_1) \sqsubseteq f_\infty(x_2) \sqsubseteq \cdots$. For all i, j we have $f_i(x_j) \sqsubseteq f_\infty(x_j) \sqsubseteq y$ therefore y is an upper bound of the sequence $f_i(x_1) \sqsubseteq f_i(x_2) \sqsubseteq \cdots$. Since f_i is limit preserving, $f_i(x_\infty)$ is the least upper bound of this sequence, therefore $f_i(x_\infty) \sqsubseteq y$, thus y is an upper bound of the sequence $f_1(x_\infty) \sqsubseteq f_2(x_\infty) \sqsubseteq \cdots$. By definition, $f_\infty(x_\infty)$ is the least upper bound of this sequence, thus $f_\infty(x_\infty) \sqsubseteq y$. □

Proposition 3.3.3.

Let $\langle X, \sqsubseteq \rangle$ and $\langle Y, \sqsubseteq \rangle$ be two domains. Given a sequence $f_1 \sqsubseteq f_2 \sqsubseteq \cdots$ in $\mathit{Cont}(X, Y)$, its limit f_∞ is the supremum of the sequence.

Proof. First we show that f_∞ is an upper bound of the sequence. Take $i \geq 1$ and $x_1 \sqsubseteq x_2 \in X$. Since $f_\infty(x_2)$ is the limit of the sequence $f_1(x_2) \sqsubseteq f_2(x_2) \sqsubseteq \cdots$ we have $f_i(x_2) \sqsubseteq f_\infty(x_2)$. Furthermore since $x_1 \sqsubseteq x_2$, we have $f_i(x_1) \sqsubseteq f_i(x_2) \sqsubseteq f_\infty(x_2)$, therefore $f_i \sqsubseteq f_\infty$.

Then we show that f_∞ is the lowest upper bound. Let g be an upper bound of the sequence, and $x_1 \sqsubseteq x_2 \in X$. Since g is an upper bound of the sequence, $g(x_1)$ is an

upper bound of the sequence $f_1(x_1) \sqsubseteq f_2(x_1) \sqsubseteq \cdots$. By definition $f_\infty(x_1)$ is the lowest upper bound of this sequence, then $f_\infty(x_1) \sqsubseteq g(x_1) \sqsubseteq g(x_2)$. □

To conclude, we exhibit the smallest continuous mapping as the constant mapping returning the smallest element of the image domain.

Proposition 3.3.4.

Let $\langle X, \sqsubseteq\rangle$ and $\langle Y, \sqsubseteq\rangle$ be two domains, and let $y_\perp$ be the minimum of Y. The mapping $f_\perp \in \mathit{Cont}(X, Y)$ such that for all $x \in X$, $f_\perp(x) = y_\perp$, is the minimum of $\mathit{Cont}(X, Y)$.

Proof. Straightforward. □

Corollary 3.3.5.

Let $\langle X, \sqsubseteq\rangle$ and $\langle Y, \sqsubseteq\rangle$ be two domains. The partial ordered set $\langle \mathit{Cont}(X, Y), \sqsubseteq\rangle$ is a domain.

Finally, the TAS of continuous mapping is inductively defined from a domain X as follows.

Definition 3.3.5 (The TAS of continuous mappings)**.**

Let X be a domain. The ***mappings TAS*** $\langle \mathbf{Map}(X), \cdot\rangle$ is inductively defined by:

- $\mathbf{Map}^o(X) = X$,
- $\mathbf{Map}^{\tau\to\tau'}(X)$ is the domain $\mathit{Cont}\big(\mathbf{Map}^\tau(X), \mathbf{Map}^{\tau'}(X)\big)$,
- given $f : \tau \to \tau' \in \mathbf{Map}(X)$ and $g : \tau \in \mathbf{Map}(X)$, $f \cdot g = f(g)$.

3.3.2. Domain dependent value of a scheme

We fix a domain X, a ranked alphabet Σ and a type preserving mapping $\zeta : \Sigma \to \mathbf{Map}(X)$. In the following we write $\mathbf{Map} = \mathbf{Map}(X)$. Given a scheme, we define the morphism φ_ζ as the morphism that evaluates all symbols of Σ according to ζ and all other symbols as the constant mapping returning $x_\perp$. The morphism φ_ζ generalises the notion of $\perp$-transformation where X is the set of trees, and $x_\perp = \perp$.

Definition 3.3.6 (The morphism φ_ζ)**.**

Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be a scheme. The morphism φ_ζ is defined by letting, for all $a \in \Sigma$, $\varphi_\zeta(a) = \zeta(a)$, for all non-terminal symbol $F : \tau_1 \to \cdots \to \tau_k$, and for all $h_1 : \tau_1, \ldots, h_k : \tau_k \in \mathbf{Map}$:

$$\varphi_\zeta(F)\ h_1 \ldots h_k = x_\perp.$$

Remark that if we enlarge ζ to $\perp$ as $\zeta(\perp) = x_\perp$, we have $\varphi_\zeta(t) = \varphi_\zeta(t^\perp)$.

Now we are in position to define the domain dependent value of a ground typed term, as the limit of the sequence obtained by applying φ_ζ to a derivation, as we did when we defined the value tree of a scheme.

Proposition 3.3.6 (domain dependent value)**.**

Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be a scheme, $t_0 : o \in \mathcal{T}(\Sigma \uplus \mathcal{N})$, and $t_0 \to t_1 \to \cdots$ be a derivation from t_0. Then we have

$$\varphi_\zeta(t_0) \sqsubseteq \varphi_\zeta(t_1) \sqsubseteq \varphi_\zeta(t_2) \sqsubseteq \cdots .$$

We write $\|t_0\|^\zeta$ the limit of this sequence, and we call it the ***domain dependent value*** of t_0 with respect to ζ.

Proof. It suffices to notice that given a redex $r = F\ t_1 \ldots t_k$ that rewrites into a term t, $\varphi_\zeta(r) = x_\perp \sqsubseteq \varphi_\zeta(t)$. □

Given an scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$, let $\|\mathcal{G}\|^\zeta = \|S\|^\zeta$ be the ***domain dependent value*** of $\mathcal{G}$ with respect to ζ. For example if X is the set of trees, and for all $a : o^k \to o \in \Sigma$, $\zeta(a)$ maps the tuple of trees $(T_1, \ldots, T_k)$ to the tree $a\ T_1 \ldots T_k$, then $\|\mathcal{G}\|^\zeta$ is the value tree of the scheme.

Example 3.3.1 (Unproductive terms). Given a term $t : o$, knowing that $\|t\|_\mathcal{G} = \perp$ is useful since it means that there is no interest in rewriting t. Let $X = \{x_\perp, x_\top\}$ with $x_\perp \sqsubseteq x_\top$. For all $a \in \Sigma$, let $\zeta_U(a)$ be the constant mapping that to any arguments maps the value $x_\top$. Then it is easy to observe that:

$$\|\mathcal{G}\|^{\zeta_U} = x_\perp \text{ if and only if } \|\mathcal{G}\| = \perp.$$

Indeed, let $S = t_0 \Rightarrow t_1 \Rightarrow t_2 \Rightarrow \cdots$ be the parallel derivation of $\mathcal{G}$. If $\|\mathcal{G}\| = \perp$ then for all i, $(t_i)^\perp = \perp$ *i.e.* t_i is a redex, therefore $\varphi_\zeta(t_i) = x_\perp$, hence $\|\mathcal{G}\|^{\zeta_U} = x_\perp$. On the other hand, if there exists i such that $t_i = a\ s_1 \ldots s_k$ for some $a \in \Sigma$ and $s_1, \ldots, s_k$,

then for all $j \geq i$, $t_j = a\ s'_i \dots s'_k$ for some $s'_1, \dots, s'_k$, therefore for all $j \geq i$, $\varphi_\zeta(t_j) = x_\top$, hence $\|\mathcal{G}\|^\zeta = x_\top$.

Example 3.3.2 ($\bot$-Blind TAC Automata). A TAC automaton $\mathcal{A} = \langle \Sigma \uplus \{\bot\}, Q, q_0, \delta\rangle$ is $\bot$-blind if $\bot$ is accepted from any states, *i.e.* for all $q \in Q$, $(q, \bot) \in \delta$. The poset $X_\mathcal{A} = \mathcal{P}(Q)$ is the set of subsets of states, with $S_1 \sqsubseteq S_2$ be defined by $S_1 \sqsubseteq S_2$ if and only if $S_1 \supseteq S_2$. The smallest element is Q and the limit of a sequence $S_1 \sqsubseteq S_2 \sqsubseteq S_3 \sqsubseteq \cdots$ is $S_\infty = \bigcap_i S_i$.

For all $a : o^k \to o$ and $S_1, \dots, S_k \subseteq Q$, we define

$$\zeta_\mathcal{A}(a)\ S_1 \dots S_k = \{q \mid \exists q_1 \in S_1, \dots, q_k \in S_k\ q \xrightarrow{a} (q_1, \dots, q_k) \in \delta\}.$$

Proposition 3.3.7.

For all state $q \in Q$, $q \in \|\mathcal{G}\|^{\zeta_\mathcal{A}}$ if and only if $\mathcal{A}$ accepts the tree $\|\mathcal{G}\|$ from state q.

In order to prove the proposition we first establish some claims.

For all tree T, we define the mapping $Q_T : dom^T \to 2^Q$ as $Q_T(u) = \mathcal{A}(T[u])$ for all $u \in dom^T$. Recall that $T[u]$ is the subtree of T rooted in u, and $\mathcal{A}(T[u])$ is the set of states q such that $\mathcal{A}$ accepts $T[u]$ from state q.

Claim 3.3.8. For all term $t : o$, we have $\varphi_{\zeta_\mathcal{A}}(t) = Q_{t^\bot}(\varepsilon)$. ⌟

Proof. Straightforward induction of the structure of t (including redexes as basis case). □

Claim 3.3.9. Given two trees $T \sqsubseteq T'$, then for all $u \in dom(T)$, $Q_{T'}(u) \subseteq Q_T(u)$. ⌟

Proof. Given a run r of $\mathcal{A}$ from state q on $T[u]$. We define the tree $r' : dom^{T'[u]} \to \Sigma_\bot \times Q$ as for all $v \in dom^{T'[u]}$, $r'(v) = r(v)$. Then clearly r' is a run of $\mathcal{A}$ from state q on $T'[u]$. □

Claim 3.3.10. For all sequence of trees $T_1 \sqsubseteq T_2 \sqsubseteq \cdots$ whose limit is denoted T_∞, for all $u \in dom^{T_\infty}$ let i_u such that for all $j \geq i_u$, $u \in dom^{T_j}$. Then $Q_{T_\infty}(u) = \bigcap_{j \geq i_u} Q_{T_j}(u)$. ⌟

Proof. From the previous claim, we have $Q_{T_\infty}(u) \subseteq \bigcap_{j \geq i_u} Q_{T_j}(u)$. We write $f(u) = \bigcap_{j \geq i_u} Q_{T_j}(u)$. Then we have to show that $f(u) \subseteq Q_{T_\infty}(u)$, *i.e.* for all $q \in \bigcap_{j \geq i_u} Q_{T_j}(u)$ there exists a run of $\mathcal{A}$ on $T_\infty[u]$ from q.

First, observe that for all $u \in dom^{T_\infty}$, and $q \in f(u)$ if $T_\infty(u) = a : o^k \to k$ then there exists $q \xrightarrow{a} (q_1, \dots, q_k)$ such that for all ℓ, $q_\ell \in f(u \cdot \ell)$. Indeed there exists $i \geq i_u$ such that for all $j \geq i$, $T_j(u) = a$. We have $\bigcap_{j \geq i_u} Q_{T_j}(u) = \bigcap_{j \geq i} Q_{T_j}(u)$ and for all $j \geq i$

there exists $q \xrightarrow{a} (q_1, \ldots, q_k)$ such that for all ℓ, $q_\ell \in Q_{T_j}(u \cdot \ell)$. For all j we define the set Tuples_j as follows:

$$\mathit{Tuples}_j = \{(q_1, \ldots, q_k) \mid \forall \ell\ q_\ell \in Q_{T_j}(u \cdot \ell), q \xrightarrow{a} (q_1, \ldots, q_k) \in \delta\}.$$

Since for all ℓ, $Q_{T_i}(u \cdot \ell) \supseteq Q_{T_{i+1}}(u \cdot \ell) \supseteq \cdots$, we have $\mathit{Tuples}_i \supseteq \mathit{Tuples}_{i+1} \supseteq \cdots$. Since we now that Tuples_j is never empty, there exists $(q_1, \ldots, q_k) \in \mathit{Tuples}_j$ for all $j \geq i$, *i.e.* for all ℓ, and for all $j \geq i$, $q_\ell \in Q_{T_j}(u \cdot \ell)$, therefore $q_\ell \in f(u \cdot \ell)$.

Now we can inductively construct the run r of $\mathcal{A}$ on $T_\infty[u]$ from q, showing that for all node v, $r(v) \in f(u \cdot v)$. We set $r(\varepsilon) = q \in f(u)$. Assume that the run is defined up to node v, and let $q' = r(v)$. By induction hypothesis, $q' \in f(u \cdot v)$. If $T_\infty(u) = a : o^k \to k$ then there exists $q' \xrightarrow{a} (q_1, \ldots, q_k)$ such that for all ℓ, $q_\ell \in f(u \cdot v \cdot \ell)$. We set for all ℓ, $r(v \cdot \ell) = q_\ell \in f(u \cdot v \cdot \ell)$.

□

Proof of Proposition **3.3.7**. If $\mathcal{A}$ accepts $\|\mathcal{G}\|$ from state q, then $q \in Q_{\|\mathcal{G}\|}(\varepsilon)$, therefore, since for all i, $t_i^\perp \sqsubseteq \|\mathcal{G}\|$, we have $q \in Q_{t_i^\perp}(\varepsilon)$ (Claim **3.3.9**). Furthermore $Q_{t_i^\perp}(\varepsilon) = \varphi_\zeta(t_i)$ (Claim **3.3.8**), therefore $q \in \varphi_\zeta(t_i)$ for all i, hence $q \in \|\mathcal{G}\|^\zeta$.

If $q \in \|\mathcal{G}\|$ then $q \in \bigcap_i \varphi_\zeta(t_i) = \bigcap_i Q_{t_i^\perp}(\varepsilon) = Q_{\|\mathcal{G}\|}(\varepsilon)$ (Claim **3.3.10**), therefore $\mathcal{A}$ accepts $\|\mathcal{G}\|$ from state q.

□

Remark that if we define the $\perp$-blind TAC automaton $\mathcal{B} = \langle \Sigma \uplus \{\perp\}, \{q_\perp\}, q_\perp, \{(q_\perp, \perp)\}\rangle$, *i.e.* the automaton that only accepts the tree $\perp$, and if we identify $x_\top$ with the empty set and $x_\perp$ with the set $\{q_\perp\}$, we get $\zeta_\mathcal{B} = \zeta_U$, thus the first example can be seen as a specialisation of the second one.

3.3.3. The domain of morphisms

When the domain X is finite, we want to compute the domain dependent value of a scheme (*e.g.* the set of accepting states of a $\perp$-blind TAC automaton). In order to do so, we will construct a morphism $\varphi^\mathcal{G}$, called the ***device morphism of*** ζ such that for all term $t : o$, $\varphi^\mathcal{G}(t) = \|t\|^\zeta$.

In order to construct $\varphi^\mathcal{G}$ we will show that the set of morphisms from the terms to **Map** which extend ζ is a domain. Then we will define a continuous transformation of morphisms $\mathcal{F}$ from the set of rewrite rules of the scheme, and construct $\varphi^\mathcal{G}$ as the smallest fixpoint of $\mathcal{F}$.

When X is finite, the number of morphisms is finite and $\mathcal{F}$ is effective. Thus the fixpoint iteration is computable and terminates, providing the morphism $\varphi^\mathcal{G}$.

Given a scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R}\rangle$, we let $\mathbf{Morph}_\zeta(\mathcal{G})$ be the set of all morphisms φ from $\mathcal{T}(\Sigma \uplus \mathcal{N})$ to **Map** such that for all $a \in \Sigma$, $\varphi(a) = \zeta(a)$. By definition, $\varphi_\zeta \in \mathbf{Morph}_\zeta(\mathcal{G})$.

We extend the relation $\sqsubseteq$ to $\mathbf{Morph}_\zeta(\mathcal{G})$, and we show that it is a partial order.

Definition 3.3.7.

Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be a scheme. Given $\varphi, \psi \in \mathbf{Morph}_\zeta(\mathcal{G})$, we define $\varphi \sqsubseteq \psi$ as for all $F \in \mathcal{N}$, $\varphi(F) \sqsubseteq \psi(F)$.

Proposition 3.3.11.

Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be a scheme. The relation $\sqsubseteq$ is a partial order on $\mathbf{Morph}_\zeta(\mathcal{G})$.

Proof. Straightforward from the definition. □

Proposition 3.3.12 (Limit Morphism)**.**

Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be a scheme. Given a sequence of morphisms $\varphi_1 \sqsubseteq \varphi_2 \sqsubseteq \cdots \in \mathbf{Morph}_\zeta(\mathcal{G})$, the sequence has a lowest upper bound φ_∞, inductively defined as follows.
For all $F \in \mathcal{N}$, $\varphi_\infty(F)$ is the limit of the monotonic sequence $\varphi_1(F) \sqsubseteq \varphi_2(F) \sqsubseteq \cdots \in \mathbf{Map}$.

Proof. Straightforward from the definition. □

Proposition 3.3.13 (Smallest Morphism)**.**

Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be a scheme. The morphism φ_ζ is the smallest element of $\mathbf{Morph}_\zeta(\mathcal{G})$ with respect to $\sqsubseteq$.

Proof. Straightforward since for all $F : \tau \in \mathcal{N}$ $\varphi_\zeta(F)$ is the constant mapping of type τ returning $x_\perp$. □

As a corollary we get that $\mathbf{Morph}_\zeta(\mathcal{G})$ is a domain. The following lemma combines limits of morphisms and limits of environments.

Lemma 3.3.14.

Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be a scheme and let $\varphi^1 \sqsubseteq \varphi^2 \sqsubseteq \cdots \in \mathbf{Morph}_\zeta(\mathcal{G})$ be a sequence of morphisms and φ^∞ be its limit.
Let $x_1, \ldots, x_k \in \mathcal{V}$ and $\rho_1, \rho_2, \ldots$ a sequence of environments defined for $x_1, \ldots, x_k$ and such that for all i, j $\rho_i(x_j) \sqsubseteq \rho_{i+1}(x_j)$.
Let ρ be the environment defined by, for all j, $\rho(x_j)$ is the limit of the sequence $\rho_1(x_j) \sqsubseteq \rho_2(x_2) \sqsubseteq \cdots$.
Let $e \in \mathcal{T}(\Sigma \uplus \mathcal{N} \uplus \{x_1, \ldots, x_k\})$. Then:
the sequence $\varphi^1_{\rho_1}(e), \varphi^2_{\rho_2}(e), \ldots$ satisfies $\varphi^1_{\rho_1}(e) \sqsubseteq \varphi^2_{\rho_2}(e) \sqsubseteq \cdots$ and its limit is $\varphi^\infty_\rho(e)$.

Proof. The proof proceeds by structural induction on e, the only non trivial case is when $e = e_1\ e_2$. In this case, for all i we write $f_i = \varphi^i_{\rho_i}(e_1)$ and $h_i = \varphi^i_{\rho_i}(e_2)$.

By induction hypothesis, for all i, $f_i \sqsubseteq f_{i+1}$ and $h_i \sqsubseteq h_{i+1}$, therefore $f_i\ h_i \sqsubseteq f_{i+1}\ h_i \sqsubseteq f_{i+1}\ h_{i+1}$. Let f_∞ and h_∞ be the respective limits of the sequences $f_1 \sqsubseteq f_2 \sqsubseteq \cdots$ and $h_1 \sqsubseteq h_2 \sqsubseteq \cdots$. By induction hypothesis $f_\infty = \varphi^\infty_\rho(e_1)$ and $h_\infty = \varphi^\infty_\rho(e_2)$.

We need to prove that $f_\infty\ h_\infty$ is the limit of the sequence $f_1\ h_1 \sqsubseteq f_2\ h_2 \sqsubseteq \cdots$, since $f_\infty\ h_\infty = \varphi^\infty_\rho(e_1)\ \varphi^\infty_\rho(e_2) = \varphi^\infty_\rho(e)$.

It is an upper bound since for all i $f_i\ h_i \sqsubseteq f_\infty\ h_i \sqsubseteq f_\infty\ h_\infty$. Let g be an upper bound of that sequence. For all i, j setting $i_{\max} = \max(i, j)$ we have $f_i\ h_j \sqsubseteq f_{i_{\max}}\ h_{i_{\max}} \sqsubseteq g$. Then g is an upper bound of the sequence $f_1\ h_j \sqsubseteq f_2\ h_j \sqsubseteq \cdots$ therefore $f_\infty\ h_j \sqsubseteq g$. Thus g is an upper bound of the sequence $f_\infty\ h_1 \sqsubseteq f_\infty\ h_2 \sqsubseteq \cdots$ therefore $f_\infty\ h_\infty \sqsubseteq g$ since $f_\infty\ h_\infty$ is the limit of this sequence (by continuity of f_∞). □

3.3.4. The device morphism

We now introduce the continuous mappings between morphisms, which is the basis of the fixpoint construction of Section 3.3.5. Recall that the type preserving mapping ζ has been fixed.

Proposition 3.3.15.

Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be a scheme. Given $\psi \in \mathbf{Morph}_\zeta$ there exists a unique morphism $\varphi \in \mathbf{Morph}_\zeta$ such that $\varphi \to_\mathcal{G} \psi$, we denote this morphism $\mathcal{F}_\mathcal{G}(\psi)$. This morphism is defined by for all $H\ x_1 \ldots x_k \to e \in \mathcal{R}$, for all $h_1, \ldots, h_k \in \mathbf{Map}$,

$$\mathcal{F}_\mathcal{G}(\psi)(H)\ h_1 \ldots h_k = \psi_\rho(e),$$

with $\rho(x_i) = h_i$ for all i.

Proof. We first check that $\mathcal{F}_\mathcal{G}(\psi)$ is in $\mathbf{Morph}_\zeta$, *i.e.* we prove that for all H, $\mathcal{F}_\mathcal{G}(\psi)(H)$ is continuous.

Let $h_1^1 \sqsubseteq h_1^2 \sqsubseteq \cdots, \ldots, h_k^1 \sqsubseteq h_k^2 \sqsubseteq \cdots$ be k sequences in **Map** and let $h_1^\infty, \ldots, h_k^\infty$ be their limits. Let $\rho_1, \rho_2, \ldots$ and ρ_∞ be some environments such that for all i, j, $\rho_j(x_i) = h_i^j$ and $\rho_\infty(x_i) = h_i^\infty$.

For all j, we have $\mathcal{F}_\mathcal{G}(\psi)(H)\ h_1^j \ldots h_k^j = \psi_{\rho_j}(e)$ and $\varphi(H)\ h_1^\infty \ldots h_k^\infty = \psi_{\rho_\infty}(e)$. Thanks to Lemma **3.3.14**, $\psi_{\rho_1}(e) \sqsubseteq \psi_{\rho_2}(e) \sqsubseteq$ and its limit is $\psi_{\rho_\infty}(e)$, therefore $\mathcal{F}_\mathcal{G}(\psi)(H)$ is continuous.

We now establish the unicity. Observe that if $\varphi' \to_\mathcal{G} \psi$ then for all $H\ x_1 \ldots x_k \to e \in \mathcal{R}$, for all $h_1, \ldots, h_k \in \mathbf{Map}$, $\varphi'(H)\ h_1 \ldots h_k = \psi_\rho(e)$, with $\rho(x_i) = h_i$ for all i. Thus $\varphi'(H) = \mathcal{F}_\mathcal{G}(\psi)(H)$.

□

The following two propositions show that the mapping $\mathcal{F}_\mathcal{G}$ is continuous.

Proposition 3.3.16.

Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be a scheme. $\mathcal{F}_\mathcal{G}$ is monotonic, *i.e.* for all $\psi \sqsubseteq \psi'$, $\mathcal{F}_\mathcal{G}(\psi) \sqsubseteq \mathcal{F}_\mathcal{G}(\psi')$.

Proof. We prove that for all $H \in \mathcal{N}$, $\mathcal{F}_\mathcal{G}(\psi)(H) \sqsubseteq \mathcal{F}_\mathcal{G}(\psi')(H)$. Let $H\ x_1 \ldots x_k \to e$ be the rewrite rule associated with H. Let $h_1, \ldots, h_k, h'_1, \ldots, h'_k \in \mathbf{Map}$ such that for all i, $h_i \sqsubseteq h'_i$. We need to show that $\mathcal{F}_\mathcal{G}(\psi)(H)\ h_1 \ldots h_k \sqsubseteq \mathcal{F}_\mathcal{G}(\psi')(H)\ h'_1 \ldots h'_k$. Let ρ_1, ρ_2 be two environments whose domains contain $x_1, \ldots, x_k$ and such that for all i, $\rho_1(x_i) = h_i$ and $\rho_2(x_i) = h'_i$. To conclude we need to show that $\psi_{\rho_1}(e) \sqsubseteq \psi'_{\rho_2}(e)$. We prove that by induction on e:

- if $e = x_i$ for some i then $\psi_{\rho_1}(x_i) = \rho_1(x_i) \sqsubseteq \rho_2(x_i) = \psi'_{\rho_2}(x_i)$,
- if $e = a \in \Sigma$ then $\psi_{\rho_1}(a) = f(a) = \psi'_{\rho_2}(x_i)$,
- if $e = F \in \mathcal{N}$ then $\psi_{\rho_1}(F) = \psi(F) \sqsubseteq \psi'(F) = \psi'_{\rho_2}(F)$,
- if $e = e_1\ e_2$ then by induction hypothesis $\psi_{\rho_1}(e_1) \sqsubseteq \psi'_{\rho_2}(e_1)$ and $\psi_{\rho_1}(e_2) \sqsubseteq \psi'_{\rho_2}(e_2)$ therefore $\psi_{\rho_1}(e_1\ e_2) = (\psi_{\rho_1}(e_1)\ \psi_{\rho_1}(e_2)) \sqsubseteq (\psi'_{\rho_1}(e_1)\ \psi_{\rho_1}(e_2)) \sqsubseteq (\psi'_{\rho_2}(e_1)\ \psi'_{\rho_2}(e_2)) = \psi'_{\rho_2}(e_1\ e_2)$.

□

Proposition 3.3.17.

Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be a scheme. Let $\varphi^1 \sqsubseteq \varphi^2 \sqsubseteq \cdots$ be a sequence of morphisms, and φ^∞ be its limit. Since $\mathcal{F}_\mathcal{G}$ is monotonic, the sequence $\mathcal{F}_\mathcal{G}(\varphi^1), \mathcal{F}_\mathcal{G}(\varphi^2), \ldots$ satisfies that $\mathcal{F}_\mathcal{G}(\varphi^1) \sqsubseteq \mathcal{F}_\mathcal{G}(\varphi^2) \sqsubseteq \cdots$. Let ψ^∞ be the limit of this sequence, then

$$\mathcal{F}_\mathcal{G}(\varphi^\infty) = \psi^\infty.$$

Proof. For all $H\ x_1 \ldots x_k \to e \in \mathcal{R}$ $\mathcal{F}_\mathcal{G}(\varphi^\infty)(H)\ h_1 \ldots h_k$ is defined by

$$\mathcal{F}_\mathcal{G}(\varphi^\infty)(H)\ h_1 \ldots h_k = {\varphi^\infty}_\rho(e)$$

with $\rho(x_i) = h_i$ for all i. On the other hand $\psi^\infty(H)\ h_1 \ldots h_k$ is defined as the limit of the sequence $\mathcal{F}_\mathcal{G}(\varphi^i)(H)\ h_1 \ldots h_k$, *i.e.* as the limit of the sequence $\varphi^i_\rho(e)$.

Using Lemma **3.3.14**, $\varphi_\rho(e)$ is the limit of the sequence $\varphi^i_\rho(e)$ which concludes the proof. □

Definition 3.3.8 (Device morphism)**.**

Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be a scheme. The ***least sequence of morphisms*** associated with $\mathcal{G}$, $\varphi^\mathcal{G}_0, \varphi^\mathcal{G}_1, \varphi^\mathcal{G}_2, \ldots$ is defined by $\varphi^\mathcal{G}_0 = \varphi_\zeta$ and for all i, $\varphi^\mathcal{G}_{i+1} = \mathcal{F}_\mathcal{G}(\varphi_i)$.
By induction this sequence is monotonic, we call its limit $\varphi^\mathcal{G}$ the ***device morphism*** of ζ.

Proposition 3.3.18.

The morphism $\varphi^\mathcal{G}$ is the smallest fixpoint of $\mathcal{F}_\mathcal{G}$. In particular it is stable by rewrite.

Proof. Proposition **3.3.17** states that $\mathcal{F}_\mathcal{G}(\varphi^\mathcal{G})$ is the limit of the sequence $\mathcal{F}_\mathcal{G}(\varphi^\mathcal{G}_0) \sqsubseteq \mathcal{F}_\mathcal{G}(\varphi^\mathcal{G}_1) \sqsubseteq \mathcal{F}_\mathcal{G}(\varphi^\mathcal{G}_2) \sqsubseteq \cdots$ which is equal to the sequence $\varphi^\mathcal{G}_1 \sqsubseteq \varphi^\mathcal{G}_2 \sqsubseteq \varphi^\mathcal{G}_3 \sqsubseteq \cdots$. But the limit of this sequence is also $\varphi^\mathcal{G}$, hence $\varphi^\mathcal{G} = \mathcal{F}_\mathcal{G}(\varphi^\mathcal{G})$.

Take another fixpoint φ. By induction on i we show that $\varphi^\mathcal{G}_i \sqsubseteq \varphi$, since $\varphi^\mathcal{G}_0 = \varphi_\zeta \sqsubseteq \varphi$ and if $\varphi^\mathcal{G}_i \sqsubseteq \varphi$ then $\varphi^\mathcal{G}_{i+1} = \mathcal{F}_\mathcal{G}(\varphi^\mathcal{G}_i) \sqsubseteq \mathcal{F}_\mathcal{G}(\varphi) = \varphi$. Therefore φ is an upper bound of the least sequence, thus $\varphi^\mathcal{G} \sqsubseteq \varphi$. □

3.3.5. The device morphism computes the domain dependent value

We intend to prove that given a scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$, $\varphi^\mathcal{G}(S) = \|\mathcal{G}\|^\zeta$. In order to do so we prove that $\|\mathcal{G}\|^\zeta \sqsubseteq \varphi^\mathcal{G}(S)$ (Proposition **3.3.19**) and that $\varphi^\mathcal{G}(S) \sqsubseteq \|\mathcal{G}\|^\zeta$

(Proposition **3.3.28**). The first part is relatively easy, the second part relies on the use of the strong normalisation theorem, a classical λ-calculus result.

Given the parallel derivation $S \Rightarrow t_1 \Rightarrow t_2 \Rightarrow \cdots$, we show that the sequences

$$\varphi_0^{\mathcal{G}}(S) \sqsubseteq \varphi_1^{\mathcal{G}}(S) \sqsubseteq \varphi_2^{\mathcal{G}}(S) \sqsubseteq \cdots \quad \text{and} \quad \varphi_\zeta(S) \sqsubseteq \varphi_\zeta(t_1) \sqsubseteq \varphi_\zeta(t_2) \sqsubseteq \cdots$$

have the same limit.

The first sequence increases faster than the second, *i.e.* $\varphi_\zeta(t_i) \sqsubseteq \varphi_i^{\mathcal{G}}(S)$ for all i. As a consequence, we get that $\|\mathcal{G}\|^\zeta \sqsubseteq \varphi^{\mathcal{G}}(S)$. On the other hand, we need to show that even though the first sequence increases faster, the second sequence manage to get above any element of the first sequence at some point, *i.e.* for all $\varphi_i^{\mathcal{G}}(S)$ there exists $\varphi_\zeta(t_j)$ with $j \geq i$ such that $\varphi_i^{\mathcal{G}}(S) \sqsubseteq \varphi_\zeta(t_j)$. As a consequence we get $\varphi^{\mathcal{G}}(S) \sqsubseteq \|\mathcal{G}\|^\zeta$.

This proof is an adaptation of Kobayashi's proof in [Kob09], it follows the same steps, the difference is that we talk about morphisms when he talks about types, and in [Kob09], the proof is presented when X contains sets of states of a TAC-automaton, while in here, X can be any domain. These differences do not raise the difficulty of the problem.

Proposition 3.3.19.

Given a scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$,

$$\|\mathcal{G}\|^\zeta \sqsubseteq \varphi^{\mathcal{G}}(S).$$

Claim 3.3.20. Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$. For all terms t, t', for all k, if $t \Rightarrow t'$, then

$$\varphi_k^{\mathcal{G}}(t') \sqsubseteq \varphi_{k+1}^{\mathcal{G}}(t).$$

⌟

Proof. We proceed by induction on t. If $t = a\ t_1 \dots t_k$ with $a \in \Sigma$ then $t' = a\ t'_1 \dots t'_k$ such that for all i, $t_i \Rightarrow t'_i$. Then $\varphi_k^{\mathcal{G}}(t') = \zeta(a)\ \varphi_k^{\mathcal{G}}(t'_1) \dots \varphi_k^{\mathcal{G}}(t'_k)$. By induction hypothesis, for all i, $\varphi_k^{\mathcal{G}}(t'_i) \sqsubseteq \varphi_{k+1}^{\mathcal{G}}(t_i)$, therefore $\varphi_k^{\mathcal{G}}(t') \sqsubseteq \zeta(a)\ \varphi_{k+1}^{\mathcal{G}}(t_1) \dots \varphi_{k+1}^{\mathcal{G}}(t_k) = \varphi_{k+1}^{\mathcal{G}}(t)$.

If $t = F\ t_1 \dots t_k$ and t is not a redex (*i.e.* $k < \mathit{arity}(F)$), then $t' = F\ t'_1 \dots t'_k$ such that for all i, $t_i \Rightarrow t'_i$. Then $\varphi_k^{\mathcal{G}}(t') = \varphi_k^{\mathcal{G}}(F)\ \varphi_k^{\mathcal{G}}(t'_1) \dots \varphi_k^{\mathcal{G}}(t'_k)$. By induction hypothesis, for all i, $\varphi_k^{\mathcal{G}}(t'_i) \sqsubseteq \varphi_{k+1}^{\mathcal{G}}(t_i)$. Furthermore $\varphi_k^{\mathcal{G}}(F) \sqsubseteq \varphi_{k+1}^{\mathcal{G}}(F)$, therefore $\varphi_k^{\mathcal{G}}(t') \sqsubseteq \varphi_{k+1}^{\mathcal{G}}(F)\ \varphi_{k+1}^{\mathcal{G}}(t_1) \dots \varphi_{k+1}^{\mathcal{G}}(t_k) = \varphi_{k+1}^{\mathcal{G}}(t)$.

If $t = F\ t_1 \dots t_k$ and t is a redex (*i.e.* $k = \mathit{arity}(F)$), let $F\ x_1 \dots x_k \to e \in \mathcal{R}$.

- Since t is a redex, $t' = e_{\left[\forall i\ x_i \mapsto t'_i\right]}$ such that for all i, $t_i \Rightarrow t'_i$. Then $\varphi_k^{\mathcal{G}}(t') = \varphi_k^{\mathcal{G}}(e_{\left[\forall i\ x_i \mapsto t'_i\right]})$.
- Furthermore, since $\varphi_{k+1}^{\mathcal{G}} = \mathcal{F}_{\mathcal{G}}(\varphi_k^{\mathcal{G}})$, we have $\varphi_{k+1}^{\mathcal{G}}(t) = \varphi_{k\,\rho}^{\mathcal{G}}(e)$, where ρ is the environment such that for all i, $\rho(x_i) = \varphi_{k+1}^{\mathcal{G}}(t_i)$.

Therefore, to conclude we need to prove the following:

$$\varphi_k^{\mathcal{G}}(e_{\lfloor \forall i\ x_i \mapsto t'_i \rfloor}) \sqsubseteq \varphi_{k\,\rho}^{\mathcal{G}}(e).$$

We prove this by a side induction on e.

- If $e = x_i$ for some i, then $\varphi_k^{\mathcal{G}}(e_{\lfloor \forall i\ x_i \mapsto t'_i \rfloor}) = \varphi_k^{\mathcal{G}}(t'_i)$, and, by main induction hypothesis, $\varphi_k^{\mathcal{G}}(t'_i) \sqsubseteq \varphi_{k+1}^{\mathcal{G}}(t_i) = \varphi_{k\,\rho}^{\mathcal{G}}(e)$.
- If $e = \alpha \in \Sigma \uplus \mathcal{N}$, $\varphi_k^{\mathcal{G}}(e_{\lfloor \forall i\ x_i \mapsto t'_i \rfloor}) = \varphi_k^{\mathcal{G}}(\alpha) = \varphi_{k\,\rho}^{\mathcal{G}}(e)$.
- If $e = e_1\ e_2$ then $\varphi_k^{\mathcal{G}}(e_{\lfloor \forall i\ x_i \mapsto t'_i \rfloor}) = \varphi_k^{\mathcal{G}}(e_{1\lfloor \forall i\ x_i \mapsto t'_i \rfloor})\ \varphi_k^{\mathcal{G}}(e_{2\lfloor \forall i\ x_i \mapsto t'_i \rfloor})$ and by side induction hypothesis, for $j \in \{1, 2\}$, $\varphi_k^{\mathcal{G}}(e_{j\lfloor \forall i\ x_i \mapsto t'_i \rfloor}) \sqsubseteq \varphi_{k\,\rho}^{\mathcal{G}}(e_j)$.

  Therefore $\varphi_k^{\mathcal{G}}(e_{\lfloor \forall i\ x_i \mapsto t'_i \rfloor}) \sqsubseteq \varphi_{k\,\rho}^{\mathcal{G}}(e_1)\ \varphi_{k\,\rho}^{\mathcal{G}}(e_2) = \varphi_{k\,\rho}^{\mathcal{G}}(e)$.

□

Claim 3.3.21. Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be a scheme. Let $S = t_0 \Rightarrow t_1 \Rightarrow \cdots$ be the parallel derivation from S. For all k, $\varphi_\zeta(t_k) \sqsubseteq \varphi_k^{\mathcal{G}}(S)$. ⌟

Proof. We prove by induction on $i \leq k$ that $\varphi_{k-i}^{\mathcal{G}}(t_i) \sqsubseteq \varphi_k^{\mathcal{G}}(S)$. When $i = 0$ the result is trivial. Assume that $\varphi_{k-i}^{\mathcal{G}}(t_i) \sqsubseteq \varphi_k^{\mathcal{G}}(S)$ then, since $t_i \Rightarrow t_{i+1}$, Claim **3.3.20** states that $\varphi_{k-i-1}^{\mathcal{G}}(t_{i+1}) \sqsubseteq \varphi_{k-i}^{\mathcal{G}}(t_i)$, therefore $\varphi_{k-(i+1)}^{\mathcal{G}}(t_{i+1}) \sqsubseteq \varphi_k^{\mathcal{G}}(S)$. When $i = k$ we obtain (1). □

Proof of Proposition **3.3.19**. From Claim **3.3.21** we get that for all k, $\varphi_\zeta(t_k) \sqsubseteq \varphi_k^{\mathcal{G}}(S) \sqsubseteq \varphi^{\mathcal{G}}(S)$, therefore $\varphi^{\mathcal{G}}(S)$ is an upper bound of the sequence $\varphi_\zeta(t_0) \sqsubseteq \varphi_\zeta(t_1) \sqsubseteq \cdots$ whose limit is $\|\mathcal{G}\|^\zeta$, then $\|\mathcal{G}\|^\zeta \sqsubseteq \varphi^{\mathcal{G}}(S)$. □

Now we prove $\varphi^{\mathcal{G}}(S) \sqsubseteq \|\mathcal{G}\|^\zeta$. We need to introduce the scheme $\mathcal{G}^{(m)}$ with $m \geq 0$. The intuition is that $\mathcal{G}^{(m)}$ is obtained from $\mathcal{G}$ by allowing at most m recursive calls.

Definition 3.3.9 (The scheme $\mathcal{G}^{(m)}$)**.**

Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be a scheme. Let $m \geq 0$ be a non negative integer. We define the scheme $\mathcal{G}^{(m)} = \langle \Sigma \uplus \{\texttt{bot} : o\}, \mathcal{V}, \mathcal{N}^{(m)}, \mathcal{R}^{(m)}, S^m \rangle$ where $\mathcal{N}^{(m)} = \mathcal{N} \times \{1, \ldots, m\}$ in which we write F^i the non terminal (F, i).
For all $F\ x_1 \ldots x_k \to e \in \mathcal{N}$ the rule associated with F^0 in $\mathcal{R}^{(m)}$ is

$$F^0\ x_1 \ldots x_k \to \texttt{bot}.$$

For all $i > 0$ the rule associated with F^i is

$$F^i\ x_1 \ldots x_k \to e_{[\forall H \in \mathcal{N}\ H \mapsto H^{i-1}]}.$$

We extend ζ on $\Sigma \uplus \{\mathtt{bot}\}$ as $\zeta(\mathtt{bot}) = x_\perp$. For ease of exposition, we write $\mathcal{F}^{(m)} = \mathcal{F}_{\mathcal{G}^{(m)}}$, for all i $\varphi_i^{(m)} = \varphi_i^{\mathcal{G}^{(m)}}$ and $\varphi^{(m)} = \varphi^{\mathcal{G}^{(m)}}$.

The following lemma states that (1) the device morphism of $\mathcal{G}^{(m)}$ is reached after m iteration of the mapping $\mathcal{F}^{(m)}$ and (2) that before this morphisms is reached the morphisms obtained after some iterations of $\mathcal{F}^{(m)}$ are the same as those obtained in $\mathcal{G}$ after the same amount of iterations of $\mathcal{F}_\mathcal{G}$, *i.e.* for all $i \leq m$ and all term t, $\varphi_i^\mathcal{G}(t) = \varphi_i^{(m)}(t_{lab})$, where t_{lab} is the term t where the nonterminals have been labelled by i.

Lemma 3.3.22.

Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be a scheme. The limit of the least sequence of $\mathcal{G}^{(m)}$ is reached at $\varphi_m^{(m)}$ *i.e.*

$$\varphi^{(m)} = \varphi_m^{(m)}. \tag{1}$$

Furthermore given a term $t \in \mathcal{T}(\Sigma \uplus \mathcal{N})$, for all $i \leq m$ we have

$$\varphi_i^\mathcal{G}(t) = \varphi_i^{(m)}(t_{[\forall H \in \mathcal{N}\ H \mapsto H^i]}). \tag{2}$$

Proof. To prove (1), we prove a more general result: for all $i \leq m$, for all $F^j \in \mathcal{N}^{(m)}$ with $j \leq i$,

$$\varphi^{(m)}(F^j) = \varphi_i^{(m)}(F^j).$$

Let $F^j\ x_1 \dots x_k \to e_j$ be the rewrite rule associated with F^j.

For $j = 0$ we have for all $i > 0$, $\varphi_i^{(m)}(F^0)\ h_1 \dots h_k = \zeta(\mathtt{bot}) = x_\perp$. When $i = 0$, we also have $\varphi_i^{(m)}(F^0)\ h_1 \dots h_k = x_\perp$. And since the limit of the constant sequence $x_\perp$ is $x_\perp$, we have $\varphi^{(m)}(F^0)\ h_1 \dots h_k = \zeta(\mathtt{bot}) = \perp$.

For $0 < j \leq i$, since $\varphi_i^{(m)}(F^j)\ h_1 \dots h_k = \varphi_{i-1_\rho}^{(m)}(e_j)$ and $\varphi_\rho^{(m)}(e_j) = \varphi^{(m)}(F_j)\ h_1 \dots h_k$ we need to prove

$$\varphi_{i-1_\rho}^{(m)}(e_j) = \varphi_\rho^{(m)}(e_j),$$

with $\rho(x_i) = h_i$ for all i. Therefore, we prove by a side induction that for any $e \in \mathcal{T}(\Sigma \uplus \{F^{j-1} \mid F \in \mathcal{N}\} \uplus \{x_1, \dots, x_k\})$,

$$\varphi_{i-1_\rho}^{(m)}(e) = \varphi_\rho^{(m)}(e).$$

- If $e = a \in \Sigma$, then $\varphi_{i-1_\rho}^{(m)}(a) = \zeta(a) = \varphi_\rho^{(m)}(a)$,
- if $e = x_i$ for some i, then $\varphi_{i-1_\rho}^{(m)}(x_i) = \rho(x_i) = \varphi_\rho^{(m)}(x_i)$,
- if $e = F^{j-1} \in \mathcal{N}^{(m)}$, then $\varphi_{i-1_\rho}^{(m)}(F^{j-1}) = \varphi_\rho^{(m)}(F^{j-1})$ by the main induction hypothesis,

- if $e = e_1\ e_2$, then

$$\varphi^{(m)}_{i-1\,\rho}(e) = \varphi^{(m)}_{i-1\,\rho}(e_1)\ \varphi^{(m)}_{i-1\,\rho}(e_2) = \varphi^{(m)}_{\rho}(e_1)\ \varphi^{(m)}_{\rho}(e_2) = \varphi^{(m)}_{\rho}(e)$$

  by the side induction hypothesis.

We prove a slightly more general result than (2). We prove by induction on i that for all variables $\{x_1, \ldots, x_k\}$, for all environment ρ whose domains contains x_i for all i, for all $e \in \mathcal{T}(\Sigma \uplus \mathcal{N} \uplus \{x_1, \ldots, x_k\})$,

$$\varphi^{\mathcal{G}}_{i\ \rho}(e) = \varphi^{(m)}_{i\ \rho}(e_{[\forall H \in \mathcal{N}\ H \mapsto H^i]}).$$

If $i = 0$ then they are both equal to $\varphi_{\zeta\,\rho}(e)$. For $i > 0$, we prove the result by a side induction on e, and the only interesting case is when $e = F \in \mathcal{N}$. Let $F\ y_1 \ldots y_\ell \to e_F$ be the corresponding rewrite rule. For all $h_1, \ldots, h_k$,

$$\varphi^{\mathcal{G}}_i(F)\ h_1 \ldots h_\ell = \varphi^{\mathcal{G}}_{i-1\,\rho_F}(e_F)$$

with $\rho_F(y_j) = h_j$ for all j. By main induction hypothesis,

$$\varphi^{\mathcal{G}}_{i-1\,\rho_F}(e_F) = \varphi^{(m)}_{i-1\,\rho_F}(e_{F[\forall H \in \mathcal{N}\ H \mapsto H^{i-1}]}),$$

and since $F^m\ y_1 \ldots y_\ell \to e_{F[\forall H \in \mathcal{N}\ H \mapsto H^{i-1}]} \in \mathcal{R}^{(m)}$, we have

$$\varphi^{(m)}_{i-1\,\rho_F}(e_{F[\forall H \in \mathcal{N}\ H \mapsto H^{i-1}]}) = \varphi^{(m)}_i(F^i)\ h_1 \ldots h_\ell,$$

therefore $\varphi^{\mathcal{G}}_i(F)\ h_1 \ldots h_\ell = \varphi^{(m)}_i(F^i)\ h_1 \ldots h_\ell$.

□

Corollary 3.3.23.

Given a scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$, if $S^m \Rightarrow^*_{\mathcal{G}^{(m)}} t$ then

$$\varphi^{\mathcal{G}}_m(S) = \varphi^{(m)}(t).$$

Proof. If $S^m \Rightarrow^*_{\mathcal{G}^{(m)}} t$, then there exists a derivation $S^m \Rightarrow t_1 \Rightarrow \cdots \Rightarrow t_k = t$, therefore using the previous lemma,

$$\varphi^{\mathcal{G}}_m(S) = \varphi^{(m)}_m(S^m) = \varphi^{(m)}(S^m) = \varphi^{(m)}(t_1) = \cdots = \varphi^{(m)}(t_k) = \varphi^{(m)}(t).$$

□

Definition 3.3.10 (Recursion graph of a scheme)**.**

Given a scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$, the ***recursion graph*** of $\mathcal{G}$ is the graph $\langle \mathcal{N}, E \rangle$, such that for all $F, H \in \mathcal{N}$, with $F\ x_1 \dots x_k \to e \in \mathcal{R}$, $(F, H) \in E$ if and only if there is an occurrence of H in e.

The following theorem is a classical λ-calculus result[6] (see for example [Bar85] or [Miq]).

Theorem 3.3.24 (Strong Normalisation)**.**

Given a scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ if the recursion graph of $\mathcal{G}$ is acyclic, then there exists a term $t \in \mathcal{T}(\Sigma)$ such that $S \Rightarrow^* t$.

Remark that for all scheme $\mathcal{G}$ and $m \geq 0$, the recursion graph of $\mathcal{G}^{(m)}$ is acyclic. Therefore the following claim is a direct corollary of the previous theorem.

Corollary 3.3.25.

Given a scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ and $m \geq 0$. There exists a term $t^{(m)} \in \mathcal{T}(\Sigma \uplus \{\mathtt{bot}\})$ such that $S^m \Rightarrow^*_{\mathcal{G}^{(m)}} t^{(m)}$.

One can link the value of S by the morphism $\varphi^{\mathcal{G}}_m$ and the value of $t^{(m)}$ by φ_ζ.

Lemma 3.3.26.

Given a scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$, then

$$\varphi^{\mathcal{G}}_m(S) = \varphi_\zeta(t^{(m)}).$$

Proof. $\varphi^{\mathcal{G}}_m(S) = \varphi^{(m)}(S^m) = \varphi^{(m)}(t^{(m)}) = \varphi_\zeta(t^{(m)})$. □

Now we establish a correspondence with derivations in $\mathcal{G}^{(m)}$ and in $\mathcal{G}$.

[6] Actually, the result is stated for non-recursive typed λ-terms (*i.e.* λ-Y-terms without any occurrence of Y^τ for some τ), but the equivalence with non-recursive schemes is straightforward.

Lemma 3.3.27.

If $S^m \Rightarrow^*_{\mathcal{G}^{(m)}} t$ then there exists t' such that $S \Rightarrow^*_{\mathcal{G}} t'$ and $(t^\bot)_{[\mathtt{bot}\mapsto\bot]} \sqsubseteq (t')^\bot$.

Proof. We define the relation R between $\mathcal{T}(\Sigma \uplus \{\mathtt{bot}\} \uplus \mathcal{N}^{(m)} \uplus \mathcal{V})$ and $\mathcal{T}(\Sigma \uplus \mathcal{N} \uplus \mathcal{V})$ as follows. For all $t \in \mathcal{T}(\Sigma \uplus \{\mathtt{bot}\} \uplus \mathcal{N}^{(m)} \uplus \mathcal{V})$ and $t' \in \mathcal{T}(\Sigma \uplus \mathcal{N} \uplus \mathcal{V})$, $(t, t') \in R$ if and only if:

- either $t = \mathtt{bot}$,
- or $t = \alpha\ t_1 \dots t_k$ and $t' = \alpha'\ t'_1 \dots t'_k$ such that for all i, $(t_i, t'_i) \in R$, if $\alpha \in \Sigma \uplus \mathcal{V}$ then $\alpha' = \alpha$, and if $\alpha = F^j \in \mathcal{N}^{(m)}$ then $\alpha' = F \in \mathcal{N}$.

Let $t \in \mathcal{T}(\Sigma \uplus \{\mathtt{bot}\} \uplus \mathcal{N}^{(m)})$ and $t' \in \mathcal{T}(\Sigma \uplus \mathcal{N}^{(m)})$ such that $(t, t') \in R$. Observe that by induction on the structure of t, $(t^\bot)_{[\mathtt{bot}\mapsto\bot]} \sqsubseteq (t')^\bot$. Furthermore, given $e \in \mathcal{T}(\Sigma \uplus \{\mathtt{bot}\} \uplus \mathcal{N}^{(m)} \uplus \mathcal{V})$ and $e' \in \mathcal{T}(\Sigma \uplus \mathcal{N} \uplus \mathcal{V})$ such that $(e, e') \in R$, and given $x \in \mathcal{V}$, one can prove by induction on e that $(e_{[x\mapsto t]}, e'_{[x\mapsto t']}) \in R$. Finally, a last induction on the structure of t' shows that if $t \Rightarrow_{\mathcal{G}} s$ and $t' \Rightarrow_{\mathcal{G}^{(m)}} s'$, then $(s, s') \in R$.

Therefore, since $(S^m, S) \in R$, if $S^m \Rightarrow^k_{\mathcal{G}^{(m)}} t$ and $S \Rightarrow^k_{\mathcal{G}} t'$ for some k, then $(t, t') \in R$ (recall that $S^m \Rightarrow^k t$ means that there exists $t_1, \dots, t_{k-1}$ such that $S^m \Rightarrow t_1 \Rightarrow \dots \Rightarrow t_{k-1} \Rightarrow t$). In particular $(t^\bot)_{[\mathtt{bot}\mapsto\bot]} \sqsubseteq (t')^\bot$. □

Finally, one can state the following proposition.

Proposition 3.3.28.

Given a scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$, we have $\varphi^{\mathcal{G}}(S) \sqsubseteq \|\mathcal{G}\|^\zeta$.

Proof. Let $S \Rightarrow t_1 \Rightarrow t_2 \Rightarrow \cdots$ be the parallel derivation of $\mathcal{G}$. First we prove that For all m, there exist N such that

$$\varphi^{\mathcal{G}}_m(S) \sqsubseteq \varphi_\zeta(t_N).$$

From Claim **3.3.27**, let t' be such that $S \Rightarrow^*_{\mathcal{G}} t'$ and $(t^{(m)})^\bot{}_{[\mathtt{bot}\mapsto\bot]} \sqsubseteq (t')^\bot$. Since $\zeta(\bot) = \zeta(\mathtt{bot})$, and for all $t : o$, $\varphi_\zeta(t) = \varphi_\zeta(t^\bot)$, therefore we have $\varphi_\zeta(t^{(m)}) \sqsubseteq \varphi_\zeta(t')$. From Claim **3.3.26**, we have $\varphi^{\mathcal{G}}_m(S) \sqsubseteq \varphi_\zeta(t^{(m)}) \sqsubseteq \varphi_\zeta(t')$.

For all m, there exists N such that $\varphi^{\mathcal{G}}_m(S) \sqsubseteq \varphi_\zeta(t_N)$, therefore $\|\mathcal{G}\|_\zeta$ is an upper-bound of $\varphi^{\mathcal{G}}_0(S) \sqsubseteq \varphi^{\mathcal{G}}_1(S) \sqsubseteq \cdots$, and since $\varphi^{\mathcal{G}}(S)$ is the lowest upper-bound of this sequence, $\varphi^{\mathcal{G}}(S) \sqsubseteq \|\mathcal{G}\|_\zeta$.

□

Summary. Given a domain X, a scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ and a type preserving mapping $\zeta : \Sigma \to \mathbf{Map}(X)$, we have defined a mapping $\mathcal{F}^{\mathcal{G}}$ between morphisms, such that the least fixpoint of $\mathcal{F}^{\mathcal{G}}$, called the *device morphism* $\varphi^{\mathcal{G}}$ associated with $\mathcal{G}$ and ζ is (1) stable by rewrite, (2) equal to ζ on the terminal symbols, and (3) satisfies that $\varphi^{\mathcal{G}}(S) = \|\mathcal{G}\|^{\zeta}$.

When X is finite, one can compute $\mathcal{F}$ and therefore one can compute $\varphi^{\mathcal{G}}$. This problem is n-EXPTIME complete, n being the order of $\mathcal{G}$ [CW07].

3.3.6. Applications

Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be a scheme and φ be a morphism from the terms of $\mathcal{G}$ to some TAS $\langle A, \cdot \rangle$. We say that φ ***describes a tree property*** if for all terms $t, t' : o$, if $\|t\|_{\mathcal{G}} = \|t'\|_{\mathcal{G}}$ then $\varphi(t) = \varphi(t')$. Such morphisms satisfy the following proposition.

Proposition 3.3.29.

Let $\mathcal{G}$ be a scheme and φ be a morphism from the terms of $\mathcal{G}$ to some TAS $\langle A, \cdot \rangle$, such that φ describes a tree property. Then φ is weakly stable by rewrite.

Proof. Let t, t' such that $t \to t'$. There exists a context $C[\bullet]$ a redex $r : o$ and a term $r' : o$ such that $t = C[r]$, $t' = C[r']$ and $r \to r'$. Then $\|r\| = \|r'\|$ therefore $\varphi(r) = \varphi(r')$, thus $\varphi(t) = \varphi(t')$. □

$\perp$-elimination

Recall that ζ_U is the type preserving mapping such that the associated domain dependant value satisfies that, for all term $t : o$, $\|t\|^{\zeta_U} = x_\perp$ if and only if $\|t\| = \perp$. Since $\mathbf{Map}^{\tau}(\{x_\perp, x_\top\})$ is finite for all τ, one can compute the fixpoint of the mapping $\mathcal{F}^{\mathcal{G}}$ associated with ζ_U, obtaining the device morphism that we denote φ_U. It also satisfies that $\varphi_U(t) = x_\perp$ if and only if $\|t\| = \perp$. Therefore it describes a tree property, and as a consequence it is weakly stable by rewrite.

Let $\mathcal{G}' = \langle \mathcal{V}', \Sigma', \mathcal{N}', S, \mathcal{R}' \rangle$ be the embedding of φ_U into $\mathcal{G}$. Remark that for any non terminal symbol, $F^{h_1,\dots,h_k} : \tau_1^{n_{\tau_1}} \to \cdots \to \tau_k^{n_{\tau_k}} \to o$ and for any redex

$$r = F^{h_1,\dots,h_k}\ t_1^{\mathbf{d}_1^{\tau_1}}\ \dots\ t_1^{\mathbf{d}_{n_{\tau_1}}^{\tau_1}}\ \dots\ t_k^{\mathbf{d}_1^{\tau_k}}\ \dots\ t_k^{\mathbf{d}_{n_{\tau_k}}^{\tau_k}}$$

in $\mathcal{G}'$, $\|r\| = \perp$ if and only if $\varphi(F)\ h_1 \dots h_k = x_\perp$.

Let $\mathcal{G}'' = \langle \mathcal{V}'', \Sigma \uplus \{\perp : o\}, \mathcal{N}'', S, \mathcal{R}'' \rangle$ the scheme obtained from $\mathcal{G}'$ as follows.

- $\mathcal{V}'' = \mathcal{V}' \uplus \{\eta_1, \dots, \eta_{k_{\max}} : o\}$ where $k_{\max}$ is the maximum arity of symbols in Σ.
- $\mathcal{N}'' = \Sigma' \uplus \mathcal{N}'$.

- $\mathcal{R}''$ contains the same rewriting rules as $\mathcal{R}$ with the following modifications. For all $F^{h_1,\dots,h_k}\tau_1^{n_{\tau_1}} \to \cdots \to \tau_k^{n_{\tau_k}} \to o \in \mathcal{N}'$ such that $\varphi(F)\ h_1 \dots h_k = x_\perp$ the rule associated with $F^{h_1,\dots,h_k}$ becomes

$$F^{h_1,\dots,h_k}\ \ x_1^{\mathbf{d}_1^{\tau_1}} \dots x_1^{\mathbf{d}_{n_{\tau_1}}^{\tau_1}} \ \dots\ \ x_k^{\mathbf{d}_1^{\tau_k}} \dots x_k^{\mathbf{d}_{n_{\tau_k}}^{\tau_k}} \to \mathtt{bot}.$$

  For all $a^{h_1,\dots,h_k} \in \Sigma'$ the rule associated with $a^{h_1,\dots,h_k}$ is

$$a^{h_1,\dots,h_k}\ \eta_1 \dots \eta_k \to a\ \eta_1 \dots \eta_k.$$

Intuitively, it removes the annotations on the terminal symbols, and when the value tree of a redex is $\perp$, the redex is rewritten in `bot`. Therefore one can easily show that $\|\mathcal{G}''\| = \mathit{Unlab}(\|\mathcal{G}'\|)_{[\perp\mapsto\mathtt{bot}]} = \|\mathcal{G}\|_{[\perp\mapsto\mathtt{bot}]}$.

This construction describes an n-EXPTIME algorithm that solves the $\perp$-elimination problem (n being the order of the input scheme), summarised below.

Input: An order-n scheme $\mathcal{G}$.

1. Compute the device morphism φ_U.
2. Construct the embedding $\mathcal{G}'$ of φ_U in $\mathcal{G}$.
3. Transform $\mathcal{G}'$ into $\mathcal{G}''$ that produces `bot` for every redex whose value by φ_U is equal to $x_\perp$.

Output: The scheme $\mathcal{G}''$ such that $\|\mathcal{G}''\| = \|\mathcal{G}\|_{[\perp\mapsto\mathtt{bot}]}$.

TAC-automata reflection

Let $\mathcal{G}$ be a scheme and φ be a morphism from the terms of $\mathcal{G}$ to some TAS $\langle A, \cdot\rangle$, such that φ describes a tree property. Once we have embedded φ into $\mathcal{G}$ obtaining the scheme $\mathcal{G}'$, one can turn $\mathcal{G}'$ into a scheme $\mathcal{G}''$ on the ranked alphabet $\Sigma \times A^o$ such that $\mathit{Unlab}\|\mathcal{G}''\| = \|\mathcal{G}\|$ and for all node u, if $\|\mathcal{G}''\|(u) = (a, h)$, then $\varphi(\|\mathcal{G}\|[u]) = h$ (recall that $\|\mathcal{G}\|[u]$ denotes the subtree of $\|\mathcal{G}\|$ rooted on u). Indeed it suffices to turn any $a^{h_1,\dots h_k} \in \Sigma'$ into a non terminal and associate with it the following rewrite rule:

$$a^{h_1,\dots,h_k}\ \eta_1 \dots \eta_k \to (a, \varphi(a)\ h_1 \dots h_k)\ \eta_1 \dots \eta_k.$$

In particular, given a $\perp$-blind TAC-automaton $\mathcal{A}$, recall that $\zeta_\mathcal{A}$ is a type preserving mapping such that for all term $t : o$, $\|t\|^{\zeta_\mathcal{A}}$ is the set of state q such that $\mathcal{A}$ accepts $\|t\|$ from state q.

Again, one can compute the device morphism $\varphi_\mathcal{A}$ associated with $\zeta_\mathcal{A}$. This morphism describes a tree property, therefore one can embed it into $\mathcal{G}$ and applying the transformation described above, one can construct a scheme $\mathcal{G}''$ such that $\|\mathcal{G}''\|$ is the reflection of $\mathcal{A}$ in $\|\mathcal{G}\|$.

Furthermore take a non $\perp$-blind automaton $\mathcal{B}$. One can perform the $\perp$-elimination algorithm on $\mathcal{G}$ to obtain $\mathcal{G}''$, and one can turn the automaton $\mathcal{B}$ into the $\perp$-blind

automaton $\mathcal{B}'$ where for all transitions $(q, \bot)$ in $\mathcal{B}$ there exists a transition $(q, \texttt{bot})$ in $\mathcal{B}'$. Therefore for all tree T and state q, $\mathcal{B}$ accepts T from q if and only if $\mathcal{B}'$ accepts $T_{[\bot \mapsto \texttt{bot}]}$ from q. In particular if we transform $\mathcal{G}''$ in $\mathcal{G}^\star$ such that $\|\mathcal{G}^\star\|$ is the reflection of $\mathcal{B}'$ in $\|\mathcal{G}''\|$, we have that $\|\mathcal{G}^\star\|$ is also the reflection of $\mathcal{B}$ in $\|\mathcal{G}\|$ in which all $\bot$ symbols are replaced by $\texttt{bot}$.

This construction gives an n-EXPTIME algorithm to solve the TAC-automata reflection, summarised below.

Input: An order-n scheme $\mathcal{G}$, a TAC-automaton $\mathcal{B}$.

1. Construct the scheme $\mathcal{G}''$ obtained by applying the $\bot$-elimination construction on $\mathcal{G}$.
2. Construct the $\bot$-blind TAC automaton $\mathcal{B}'$ obtained by giving to $\texttt{bot}$ the transitions of $\bot$ in $\mathcal{B}$.
3. Compute the device morphism $\varphi_{\mathcal{B}'}$ associated with $\mathcal{G}''$ and $\zeta_{\mathcal{B}'}$.
4. Construct the embedding $\mathcal{G}^0$ of $\varphi_{\mathcal{B}'}$ in $\mathcal{G}''$.
5. Transform $\mathcal{G}^0$ into $\mathcal{G}^\star$ where the symbols $a^{h_1,\dots,h_k}$ are replaced by $(a, \varphi(a)\ h_1 \dots h_k)$ for all $a \in \Sigma$.

Output: The scheme $\mathcal{G}^\star$ such that $\|\mathcal{G}^\star\|$ is the reflection of $\mathcal{B}$ in $\|\mathcal{G}\|$ (in which all $\bot$ symbols are replaced by $\texttt{bot}$).

3.4. Simulation of the IO evaluation policy

In this section, we solve the IO simulation problems. Given a scheme $\mathcal{G}$, we construct, in a similar manner, a morphism that recognises whether the IO value tree of a term t is equal to $\bot$ or not. Then by embedding this morphism into $\mathcal{G}$, using techniques from section 3.2.2, we are able to simulate with a scheme $\mathcal{G}'$ the IO behaviour of $\mathcal{G}$.

3.4.1. Recognising IO-unproductive terms

We have seen that there exists a morphism that recognises whether the value tree of a term t is equal to $\bot$ or not. However it may be the case that given a term $t : o$, its value tree $\|t\|$ is not equal to $\bot$, but its IO value tree is, *i.e.* $\|t\|_{\mathrm{IO}} = \bot$. We present here a construction, similar to the previous one, to produce a morphism recognising this property.

Assume that there exists a TAS A, a morphism φ from the terms to A, and a subset A' of A such that for all term $t : o$, $\varphi(t) \in A'$ if and only if $\|t\|_{\mathrm{IO}} = \bot$. Then clearly this morphism is not weakly stable by rewrite. Indeed take the scheme introduced in section 2.3.2 on the ranked alphabet $\{a : o\}$ whose rewrite rules are the following:

$$S \ \to \ F\ H \qquad F\ x \ \to \ \mathtt{a} \qquad H \ \to \ H.$$

We have $F\ H \to \mathtt{a}$ but $\|F\ H\|_{\mathrm{IO}} = \bot$, thus $\varphi(F\ H) \in A'$ while $\varphi(\mathtt{a}) \notin A'$.

We introduce a new notion, the ***stability by*** IO ***rewrite***. A morphism φ is stable by IO rewrite if for all t, t', if $t \to_{\text{IO}} t'$ then $\varphi(t) = \varphi(t')$. Even though the morphism we construct is not stable by (general) rewrite, it is ***stable by*** IO ***rewrite***.

As we did in the general case, we define the IO ***parallel rewriting***, as the relation between terms such that $t \Rightarrow_{\text{IO}} t'$ if t' has been obtained from t by rewriting all the IO redexes.

Definition 3.4.1 (Parallel rewriting)**.**

Let $\mathcal{G}$ be a scheme and t be a term. The term $t^{\star_{\text{IO}}}$ is defined by induction as follows.

- if $t = F\ t_1 \dots t_k$ is an IO redex with $Fx_1 \dots x_k \to e$, then $t^{\star_{\text{IO}}} = e_{[\forall i\ x_i \mapsto t_i]}$,
- otherwise $t = \alpha\ t_1 \dots t_k$ and $t^{\star_{\text{IO}}} = \alpha\ t_1^{\star_{\text{IO}}} \dots t_k^{\star_{\text{IO}}}$.

We write $t \Rightarrow_{\text{IO}} t'$ the relation "$t' = t^{\star_{\text{IO}}}$", and we call it the ***parallel rewriting***.

As previously, one can show that if $t \Rightarrow_{\text{IO}} t'$ then $t \to_{\text{IO}}^* t'$ and that given $t_0 : o$, if $t_0 \Rightarrow_{\text{IO}} t_1 \Rightarrow_{\text{IO}} t_2 \Rightarrow_{\text{IO}} \cdots$, then the limit tree of $t_0^\perp \sqsubseteq t_1^\perp \sqsubseteq t_2^\perp \sqsubseteq \cdots$, is equal to the IO value tree of t_0, $\|t_0\|_{\text{IO}}$.

Given an IO parallel derivation $t_0 \Rightarrow_{\text{IO}} t_1 \Rightarrow_{\text{IO}} t_2 \Rightarrow_{\text{IO}} \cdots$, we say that the derivation is ***finite*** if there exists t_i such that t_i does not contain any redex. In particular for all $j \geq i$, $t_j = t_i$.

The typed applicative system

Before constructing a typed applicative system for studying IO evaluation policy, we state the following proposition.

Proposition 3.4.1 (Product of Domains)**.**

Given two domains $\langle X, \sqsubseteq \rangle$ and $\langle Y, \sqsubseteq \rangle$, we define the relation $\sqsubseteq$ on $X \times Y$ as $(x, y) \sqsubseteq (x', y')$ if and only if $x \sqsubseteq x'$ and $y \sqsubseteq y'$.
Then $\langle X \times Y, \sqsubseteq \rangle$ is a domain such that $(x_\perp, y_\perp)$ is the minimum of $X \times Y$ and the limit of the sequence $(x_1, y_1) \sqsubseteq (x_2, y_2) \sqsubseteq \cdots$ is equal to (x_∞, y_∞), where x_∞ (resp. y_∞) is the limit of the sequence $x_1 \sqsubseteq x_2 \sqsubseteq \cdots$ (resp. $y_1 \sqsubseteq y_2 \sqsubseteq \cdots$).

Proof. Straightforward. □

We want to construct a morphism Φ that recognises, given a term $t : o$, whether $\|t\|_{\text{IO}} = \perp$ or not. This morphism must know for each redex $F\ t_1 \dots t_k$ if this redex will

be rewritten or not in the IO-parallel derivation. Indeed if it is not rewritten, its value tree is $\bot$ but if it is rewritten it may be anything. In order to store this information, observe that $F\ t_1 \dots t_k$ is rewritten in the IO-parallel derivation if and only if the IO-parallel derivation from each of the t_i is finite. Therefore, for all term t, $\Phi(t)$ "knows" whether the IO-parallel derivation from t is finite or not, *i.e.* there exists a mapping π from the TAS of the morphism to the set $\{\texttt{fin}, \texttt{inf}\}$ such that $\pi(\Phi(t)) = \texttt{fin}$ if and only if the IO-parallel derivation from t is finite. Therefore on a term t of type o, the morphism may have three different values: either $x_\bot$ denoting the fact that $\|t\|_{\text{IO}} = \bot$, x_∞ if $\|t\|_{\text{IO}} \neq \bot$ and the IO-parallel derivation from t is infinite, and x_Δ if the IO-parallel derivation from t is finite, *i.e.* if $\|t\|_{\text{IO}}$ is a finite tree that does not contain $\bot$.

Therefore for all term $t : \tau' \to \tau$, $\Phi(t)$ must contain two pieces of information. First it must know whether the IO-parallel derivation from t is finite or not, and second it must contain a mapping that takes as input the value $\Phi(t')$ of a term $t' : \tau'$, and returns the value of $\Phi(t\ t')$.

In the following we fix:

- a domain $X = \{x_\bot, x_\infty, x_\Delta\}$ with the order $x_\bot \sqsubseteq x_\infty \sqsubseteq x_\Delta$,
- a domain $\mathit{Fin} = \{\texttt{fin}, \texttt{inf}\}$ with the order $\texttt{inf} \sqsubseteq \texttt{fin}$,
- and the mapping π from X to Fin defined by $\pi(x_\Delta) = \texttt{fin}$ and $\pi(x_\infty) = \pi(x_\bot) = \texttt{inf}$.

We define the TAS $\langle \mathbf{M}, \cdot \rangle$ by induction as:

- $\mathbf{M}^o = X$,
- $\mathbf{M}^{\tau \to \tau'}$ is the domain $\mathit{Fin} \times \mathit{Cont}(\mathbf{M}^\tau, \mathbf{M}^{\tau'})$,
- given $(p, f) : \tau \to \tau' \in \mathbf{M}$ and $h : \tau \in \mathbf{M}$, $(p, f) \cdot h = f(h)$.

We extend the mapping π to $\mathbf{M}$ as for all $(p, f) \in \mathbf{M}$, $\pi((p, f)) = p$.

Given a type $\tau = \tau_1 \to \dots \to \tau_k \to o$ such that $\tau \neq o$, we define the set Υ^τ of tuples $(\nu_0, \dots, \nu_k)$ satisfying the following.

- For all $i < k$, ν_i is a continuous mapping from $\mathbf{M}^{\tau_1} \times \dots \times \mathbf{M}^{\tau_i}$ to Fin. In particular ν_0 is simply an element of Fin.
- ν_k is a continuous mapping from $\mathbf{M}^{\tau_1} \times \dots \times \mathbf{M}^{\tau_k}$ to X.

The following proposition states that there is a bijection from $\mathbf{M}^\tau$ to Υ^τ. Therefore in the following we define an element in $\mathbf{M}^\tau$ as its corresponding tuple in Υ^τ.

Proposition 3.4.2.

Let $\tau = \tau_1 \to \cdots \to \tau_k \to o$ be a type such that $\tau \neq o$. For all $h \in \mathbf{M}^\tau$, we define the tuple $\mathit{tuple}(h) = (\nu_0, \ldots, \nu_k) \in \Upsilon^\tau$ as follows. Let $h_1 : \tau_1, \ldots, h_k : \tau_k$ in $\mathbf{M}$.

- For all $0 \leq i < k$, $\nu_i(h_1, \ldots, h_i) = \pi(h\ h_1 \ldots h_i)$,
- $\nu_k(h_1, \ldots, h_k) = h\ h_1 \ldots h_k$.

The mapping *tuple* is a continuous bijection.

Proof. Let $\vec{\nu} = (\nu_0, \ldots, \nu_k) \in \Upsilon^\tau$. For all $0 \leq i \leq k$ and all $h_1, \ldots, h_i$, we define $g^{\vec{\nu}}_{h_1,\ldots,h_i} \in \mathbf{M}^{\tau_{i+1}\to\cdots\to\tau_k\to o}$ by induction on $k - i$ as follows.

- If $i = k$, we define $g^{\vec{\nu}}_{h_1,\ldots,h_k} = \nu_k(h_1, \ldots, h_k)$.
- If $i < k$ we define $g^{\vec{\nu}}_{h_1,\ldots,h_i} = (p, f')$ with $p = \nu_i(h_1, \ldots, h_i)$ and f' maps $h : \tau_{i+1}$ to $g^{\vec{\nu}}_{h_1,\ldots,h_i,h}$.

Let $\mathit{tuple}^{-1}(\vec{\nu}) = g^{\vec{\nu}}$, and let $(\nu'_0, \ldots, \nu'_k) = \mathit{tuple}\big(\mathit{tuple}^{-1}(\vec{\nu})\big) = \mathit{tuple}(g^{\vec{\nu}})$. In order to prove that *tuple* is a bijection, we show that $(\nu'_0, \ldots, \nu'_k) = (\nu_0, \ldots, \nu_k)$. First remark that for all $h_1, \ldots, h_i$, $g^{\vec{\nu}}\ h_1, \ldots, h_i = g^{\vec{\nu}}_{h_1,\ldots,h_i}$. Then, let $h_1 : \tau_1, \ldots, h_k : \tau_k$ in $\mathbf{M}$.

- $\nu'_k(h_1, \ldots, h_k) = g^{\vec{\nu}}\ h_1, \ldots, h_k = g^{\vec{\nu}}_{h_1,\ldots,h_k} = \nu_k(h_1, \ldots, h_k)$.
- For all $i < k$, $\nu'_i(h_1, \ldots, h_i) = \pi(g^{\vec{\nu}}\ h_1, \ldots, h_i) = \pi(g^{\vec{\nu}}_{h_1,\ldots,h_i}) = \nu_i(h_1, \ldots, h_k)$.

The continuity of *tuple* is straightforward from the definition. □

Since $\mathbf{M}^\tau$ is a domain for all τ, we are interested in giving explicitely its minimum.

Proposition 3.4.3 (Smallest value of type τ)**.**

For all type τ, let $h^{\min}_\tau : \tau \in \mathbf{M}$ defined as $h^{\min}_\tau = \mathit{tuple}^{-1}(\nu_0, \ldots, \nu_k)$ with, for all $h_1, \ldots, h_k$,

- $\nu_k(h_1, \ldots, h_k) = x_\perp$,
- for all $\nu_i(h_1, \ldots, h_k) = \texttt{inf}$.

The value $h^{\min}_\tau$ is the smallest element of type τ in $\mathbf{M}$.

Proof. Straightforward. □

The following definition matches the idea that in a term $t = \alpha\ t_1 \dots t_j$ if the IO derivation from one of the t_i is infinite, then the IO derivation from t is infinite.

Definition 3.4.2 (`inf`-restriction)**.**

We say that an element $h : \tau_1 \to \cdots \to \tau_k \to o \in \mathbf{M}$ is **`inf`-*restricted*** if the following holds:

$$\forall j \leq k\ \forall h_1, \dots, h_j\ (\exists i \leq j\ \pi(h_i) = \texttt{inf})\ \Rightarrow\ \pi(h\ h_1 \dots h_j) = \texttt{inf}.$$

Remark that if we define $\mathbf{M}_{\texttt{inf}}$ as the set of `inf`-restricted elements of $\mathbf{M}$, $\langle \mathbf{M}_{\texttt{inf}}, \cdot \rangle$ is a TAC.

Morphisms

In the following we fix a ranked alphabet Σ. The goal of this section is to build, given a scheme $\mathcal{G}$, a morphism $\Phi^{\mathcal{G}}$ such that the following holds.

(1) For all term t, $\pi(\Phi^{\mathcal{G}}(t)) = \texttt{inf}$ if and only if there exists an infinite IO derivation from t.

(2) Furthermore, if $t : o$ then $\Phi^{\mathcal{G}}(t) = x_\perp$ if and only if $\|t\|_{\text{IO}} = \perp$.

We define the value ζ of the terminal symbols following that idea. Remark that if a term $t = a\ t_1 \dots t_k$ with $a \in \Sigma$ then $\|t\|_{\text{IO}} \neq \perp$. In particular one cannot have $\Phi^{\mathcal{G}}(t) = x_\perp$.

Definition 3.4.3 (The mapping ζ)**.**

Let ζ be the type preserving mapping from Σ to $\mathbf{M}_{\texttt{inf}}$ defined by:
for all $a : o^k \to o \in \Sigma$, if $k = 0$ then $\zeta(a) = x_\Delta$, if $k > 0$ then $\zeta(a)$ is defined using Proposition **3.4.2**, as the mapping corresponding to the tuple $(\nu_1, \dots, \nu_k)$ defined as follows.
For all $h_1, \dots, h_k \in X$,

- for all $1 \leq i < k$, if $h_j = x_\Delta$ for all $j \leq i$, then $\nu_i(h_1, \dots, h_i) = \texttt{fin}$, otherwise $\nu_i(h_1, \dots, h_i) = \texttt{inf}$,
- if $h_i = x_\Delta$ for all $i \leq k$, then $\nu_k(h_1, \dots, h_k) = x_\Delta$, otherwise $\nu_k(h_1, \dots, h_k) = x_\infty$.

Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be a scheme. Let $\mathbf{Morph}_{\mathtt{inf}}(\mathcal{G})$ denotes the set of all morphisms φ from $\mathcal{T}(\Sigma \uplus \mathcal{N})$ to $\mathbf{M}_{\mathtt{inf}}$ such that for all $a \in \Sigma$, $\varphi(a) = \zeta(a)$ and for all term t, $\varphi(t)$ is $\mathtt{inf}$-restricted.

We extend the relation $\sqsubseteq$ on $\mathbf{Morph}_{\mathtt{inf}}(\mathcal{G})$ as $\varphi \sqsubseteq \psi$ if for all term t, $\varphi(t) \sqsubseteq \psi(t)$. As previously, $\langle \mathbf{Morph}_{\mathtt{inf}}(\mathcal{G}), \sqsubseteq \rangle$ is a domain.

Definition 3.4.4 (Smallest morphism)**.**

The morphism $\Phi_\zeta \in \mathbf{Morph}_{\mathtt{inf}}(\mathcal{G})$ defined by for all $F : \tau \in \mathcal{N}$, $\Phi_\zeta(F) = h^{\min}_\tau$, is the smallest element of $\mathbf{Morph}_{\mathtt{inf}}(\mathcal{G})$.

Proposition 3.4.4.

Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be a scheme. Let $t_0 : o$ be a term of $\mathcal{T}(\Sigma \uplus \mathcal{N})$ and $t_0 \Rightarrow_{\mathrm{IO}} t_1 \Rightarrow_{\mathrm{IO}} t_2 \Rightarrow_{\mathrm{IO}} \cdots$ be the IO parallel derivation from t_0. Then the sequence $\Phi_\zeta(t_0), \Phi_\zeta(t_1), \Phi_\zeta(t_2), \ldots$ is increasing, and its limit, denoted $\|t_0\|^\zeta_{\mathrm{IO}}$ satisfies the following.

(1) $\|t_0\|^\zeta_{\mathrm{IO}} = x_\Delta$ if and only if $\|t_0\|_{\mathrm{IO}}$ is a finite tree that does not contain $\bot$.

(2) $\|t_0\|^\zeta_{\mathrm{IO}} = x_\bot$ if and only if $\|t_0\|_{\mathrm{IO}} = \bot$.

Proof. If $\|t_0\|_{\mathrm{IO}}$ is a finite tree that does not contain $\bot$, then there exists i such that for all $j \geq i$, $t_j = \|t_0\|_{\mathrm{IO}}$. In particular $t_j \in \mathcal{T}(\Sigma)$. A simple induction shows that for all term $t : o \in \mathcal{T}(\Sigma)$, $\Phi_\zeta(t) = x_\Delta$. Indeed, for all $a : o \in \Sigma$, $\Phi_\zeta(a) = \zeta(a) = x_\Delta$, and for all $t = a\ t_1 \ldots t_k$ such that for all i, $t_i \in \mathcal{T}(\Sigma)$. By induction hypothesis, for all i, $\Phi_\zeta(t_i) = x_\Delta$, therefore $\Phi_\zeta(t) = \zeta(a)\ x_\Delta \ldots x_\Delta = x_\Delta$.

If $\|t_0\|_{\mathrm{IO}}$ is not a finite tree, then for all i, t_i contains a redex. Again a simple induction shows that if a term $t : o$ contains a redex, $\pi(\Phi_\zeta(t)) = \mathtt{inf}$. Indeed if $t = F\ t_1 \ldots t_k$ is a redex, then $\Phi_\zeta(F\ t_1 \ldots t_k) = h^{\min}_\tau(F)\ \Phi_\zeta(t_1) \ldots \Phi_\zeta(t_k) = x_\bot$ with τ being the type of F. If $t = a\ t_1 \ldots t_k$ with $a \in \Sigma$ contains a redex, then there exists i such that t_i contains a redex, therefore by induction hypothesis, $\pi(t_i) = \mathtt{inf}$, *i.e.* $\Phi_\zeta(t_i) \neq x_\Delta$. Thus, $\Phi_\zeta(t) = \zeta(a)\ \Phi_\zeta(t_1) \ldots \Phi_\zeta(t_k) = x_\infty$.

If $\|t_0\|_{\mathrm{IO}} = \bot$ then for all i, t_i is a redex therefore $\Phi_\zeta(t_i) = x_\bot$. Otherwise there exists i and $a : o^k \to o \in \Sigma$ such that for all $j \geq i$, $t_j = a\ s_1 \ldots s_k$ for some $s_1, \ldots, s_k$, therefore $\Phi_\zeta(t_j) = \zeta(a)\ \Phi_\zeta(s_1) \ldots \Phi_\zeta(s_k)$ which is either equal to x_∞ or x_Δ. □

Remark that (1) can be reformulated as $\Phi(\|t_0\|_{\mathrm{IO}}) = x_\Delta$ if and only if there exists j such that t_j does not contain any redex.

Now we define the mapping $\mathcal{H}^{\mathcal{G}}$ that transforms a mapping according to the rewrite rules. It is defined as $\mathcal{F}^{\mathcal{G}}$ with one exception: we must ensure that a redex $F\ t_1 \dots t_k$ is always evaluated as $x_\perp$ if there exists an infinite IO derivation from one of the t_i.

Definition 3.4.5 (The mapping $\mathcal{H}^{\mathcal{G}}$)**.**

Let $\mathcal{G}$ be a scheme. Let φ be a morphism in $\mathbf{Morph}_{\mathtt{inf}}(\mathcal{G})$. We define the morphism $\psi = \mathcal{H}^{\mathcal{G}}(\varphi)$ by for all $F : \tau \in \mathcal{N}$, and $F\ x_1 \dots x_k \to e \in \mathcal{R}$,

- if $\tau = o$, *i.e.* $k = 0$, $\psi(F) = \varphi(e)$,
- if $k > 0$ then $\psi(F)$ is defined using Proposition **3.4.2** as the element corresponding to the tuple $(\nu_0, \dots, \nu_k)$ defined as follows.

  for all $h_1 : \tau_1, \dots, h_k : \tau_k$ in $\mathbf{M}$,
  - for all $i < k$, if $\pi(h_j) = \mathtt{fin}$ for all $j \leq i$, then $\nu_i(h_1, \dots, h_j) = \mathtt{fin}$, otherwise $\nu_i(h_1, \dots, h_j) = \mathtt{inf}$,
  - if $\pi(h_i) = \mathtt{fin}$ for all $i \leq k$, then $\nu_k(h_1, \dots, h_j) = \varphi_\rho(e)$ with for all i, $\rho(x_i) = h_i$, otherwise $\nu_k(h_1, \dots, h_j) = x_\perp$.

Proposition 3.4.5.

The mapping $\mathcal{H}^{\mathcal{G}}$ is continuous.

Proof. First we prove that $\mathcal{H}^{\mathcal{G}}$ is monotonic, *i.e.* for all $\varphi_1 \sqsubseteq \varphi_2$, for all $F \in \mathcal{N}$,

$$\mathcal{H}^{\mathcal{G}}(\varphi_1)(F) \sqsubseteq \mathcal{H}^{\mathcal{G}}(\varphi_2)(F).$$

If $F : o$, let $F \to e \in \mathcal{R}$, $\mathcal{H}^{\mathcal{G}}(\varphi_1)(F) = \varphi_1(e) \sqsubseteq \varphi_2(e) = \mathcal{H}^{\mathcal{G}}(\varphi_1)(F)$. Otherwise, let $(\nu_0^1, \dots, \nu_k^1) = \mathit{tuple}(\mathcal{H}^{\mathcal{G}}(\varphi_1)(F))$ and $(\nu_0^2, \dots, \nu_k^2) = \mathit{tuple}(\mathcal{H}^{\mathcal{G}}(\varphi_2)(F))$. By definition, for all $i < k$, $\nu_i^1 = \nu_i^2$. Furthermore for all $h_1, \dots, h_k$ if there exists i such that $\pi(h_i) = \mathtt{inf}$, $\nu_k^1(h_1, \dots, h_k) = x_\perp = \nu_k^2(h_1, \dots, h_k)$. If for all i, $\pi(h_i) = \mathtt{fin}$, we have $\nu_k^1(h_1, \dots, h_k) = {\varphi_1}_\rho(e)$ with for all i, $\rho(x_i) = h_i$, and $\nu_k^2(h_1, \dots, h_k) = {\varphi_2}_\rho(e)$ therefore $\nu_k^1(h_1, \dots, h_k) \sqsubseteq \nu_k^2(h_1, \dots, h_k)$. Therefore $\mathcal{H}^{\mathcal{G}}$ is monotonic. The limit preservation of $\mathcal{H}^{\mathcal{G}}$ is similarly proven. □

Definition 3.4.6 (Morphism $\Phi^{\mathcal{G}}$)**.**

Let $\mathcal{G}$ be a scheme. Let $\Phi_0^{\mathcal{G}} \sqsubseteq \Phi_1^{\mathcal{G}} \sqsubseteq \Phi_2^{\mathcal{G}} \sqsubseteq \cdots$ be the increasing sequence of morphisms defined by $\Phi_0^{\mathcal{G}} = \Phi_\zeta$ and $\Phi_{i+1}^{\mathcal{G}} = \mathcal{H}(\Phi_i^{\mathcal{G}})$. The morphism $\Phi^{\mathcal{G}}$ is defined as the limit of this sequence.

Theorem 3.4.6.

Let $t_0 : o$ be a term of $\mathcal{T}(\Sigma \uplus \mathcal{N})$. Then $\|t_0\|_{\mathrm{IO}}^{\zeta} = \Phi^{\mathcal{G}}(t_0)$.

Proof. Once we have remarked that for all i, and for all term t, if t contains no redex then $\pi(\Phi_i^{\mathcal{G}}(t)) = \texttt{fin}$, the proof proceeds exactly as the proof of section 3.3.5 (we just change "redex" by "IO redex", "$\Rightarrow$" by "$\Rightarrow_{\mathrm{IO}}$" and so on). For the sake of completeness, the proof is presented in Appendix A.1. □

3.4.2. The IO-simulation

Now we can solve the IO simulation problem, that ask, given a scheme $\mathcal{G}$ to produce a scheme $\mathcal{G}''$ such that $\|\mathcal{G}''\| = \|\mathcal{G}\|_{\mathrm{IO}}$. The procedure consists of computing the morphism $\Phi^{\mathcal{G}}$, embedding it inside the scheme $\mathcal{G}$, and modifying the set of rewrite rules such that all redexes that would produce $\bot$ are rewritten to `bot`.

First we show that the embedding construction works also for morphisms stable by IO-rewrite.

Lemma 3.4.7.

Let $\mathcal{G}$ be a scheme, φ be a morphism stable by IO rewrite, and $\mathcal{G}'$ be the embedding of φ into $\mathcal{G}$. If $t \Rightarrow_{\mathrm{IO}} t'$ then $t^+ \Rightarrow_{\mathrm{IO}} (t')^+$.

Proof. The same proof works as in the case where φ is stable by rewrite. □

Let $\mathcal{G}$ be a scheme, $\Phi^{\mathcal{G}}$ be the morphism as defined previously, and $\mathcal{G}'$ be the embedding of $\Phi^{\mathcal{G}}$ into $\mathcal{G}$.

Corollary 3.4.8.

For all term $t : o$, $\|t\|_{\mathcal{G},\mathrm{IO}} = \mathit{Unlab}(\|t^+\|_{\mathcal{G}',\mathrm{IO}})$.

We define $\mathcal{G}'' = \langle \Sigma \uplus \{\mathtt{bot}\}, \mathcal{V}', \mathcal{N}', S, \mathcal{R}\rangle$ as the scheme obtained from $\mathcal{G}'$ as follows. First, for all $a^{d_1,\dots,d_k} \in \Sigma'$, we replace all occurrences of $a^{d_1,\dots,d_k}$ in the production of the rewrite rules by a.

Then we add the fresh terminal symbol $\mathtt{bot} : o$, and for all $F \in \mathcal{N}$ and $d_1, \dots, d_k \in \mathbf{M}_{\mathtt{inf}}$ such that $\Phi^{\mathcal{G}}(F)\ d_1 \dots d_k = x_\perp$, we replace the rewrite rule associated with $F^{d_1,\dots,d_k}$ by:

$$F^{d_1,\dots,d_k}\ \vec{x_1} \dots \vec{x_k}\ \to\ \mathtt{bot}.$$

Lemma 3.4.9.

For all term $t : o$, $\mathit{Unlab}(\|t^+\|_{\mathcal{G}',\mathrm{IO}}) = (\|t^+\|_{\mathcal{G}'',\mathrm{IO}})_{[\mathtt{bot}\mapsto\perp]}$.

Proof. Straightforward. □

Lemma 3.4.10.

For all term $t : o$, $(\|t^+\|_{\mathcal{G}'',\mathrm{IO}})_{[\mathtt{bot}\mapsto\perp]} = (\|t^+\|_{\mathcal{G}'',\mathrm{IO}})_{[\mathtt{bot}\mapsto\perp]}$. In particular:

$$(\|\mathcal{G}''\|)_{[\mathtt{bot}\mapsto\perp]} = (\|\mathcal{G}''\|_{\mathrm{IO}})_{[\mathtt{bot}\mapsto\perp]}.$$

Proof. We already know that $\|t^+\|_{\mathcal{G}'',\mathrm{IO}} \sqsubseteq \|t^+\|_{\mathcal{G}''}$ therefore $(\|t^+\|_{\mathcal{G}'',\mathrm{IO}})_{[\mathtt{bot}\mapsto\perp]} \sqsubseteq (\|t^+\|_{\mathcal{G}'',\mathrm{IO}})_{[\mathtt{bot}\mapsto\perp]}$. To show that it is equal, for all node u, we show that if $\|t^+\|_{\mathcal{G}'',\mathrm{IO}}(u)$ is equal to $\perp$ or $\mathtt{bot}$, then $\|t^+\|_{\mathcal{G}''}(u)$ is equal to $\perp$ of $\mathtt{bot}$. If $\|t^+\|_{\mathcal{G}'',\mathrm{IO}}(u) = \mathtt{bot}$, since $\|t^+\|_{\mathcal{G}'',\mathrm{IO}} \sqsubseteq \|t^+\|_{\mathcal{G}''}$, we have $\|t^+\|_{\mathcal{G}''}(u) = \mathtt{bot}$. Assume that $\|t^+\|_{\mathcal{G}'',\mathrm{IO}}(u) = \perp$, we prove, by induction on u, that $\|t^+\|_{\mathcal{G}''}(u) = \mathtt{bot}$.

Assume that $u = \varepsilon$. We have $\|t\|_{\mathcal{G},\mathrm{IO}} = \mathit{Unlab}(\|t^+\|_{\mathcal{G}',\mathrm{IO}}) = (\|t^+\|_{\mathcal{G}'',\mathrm{IO}})_{[\mathtt{bot}\mapsto\perp]} = \perp$, therefore $\|t\|_{\mathcal{G},\mathrm{IO}} = \perp$, then t is a redex $F\ t_1 \dots t_k$ and $\Phi_{\mathcal{G}}(F)\ \varphi(t_1) \dots \varphi(t_k) = x_\perp$. Therefore

$$t^+ = F^{\varphi(t_1),\dots,\varphi(t_k)}\ t_1^{\mathbf{d}_1^{\tau_1}} \dots t_1^{\mathbf{d}_{n_{\tau_1}}^{\tau_1}} \dots t_k^{\mathbf{d}_1^{\tau_k}} \dots t_k^{\mathbf{d}_{n_{\tau_k}}^{\tau_k}},$$

And since $F^{\varphi(t_1),\dots,\varphi(t_k)}\ \vec{x_1} \dots \vec{x_k}\ \to\ \mathtt{bot} \in \mathcal{R}$, we have $\|t^+\|_{\mathcal{G}''}(u) = \mathtt{bot}$.

Let $u = j \cdot u'$ and $a = \|t^+\|_{\mathcal{G}'',\mathrm{IO}}(\varepsilon)$. Then $t^+ \Rightarrow^*_{\mathrm{IO}} a\ t_1^+ \cdots t_k^+$, and $\|t^+\|_{\mathcal{G}'',\mathrm{IO}} = a\ \|t_1^+\|_{\mathcal{G}'',\mathrm{IO}} \dots \|t_k^+\|_{\mathcal{G}'',\mathrm{IO}}$. Therefore $t^+ \to^* a\ t_1^+ \dots t_k^+$ and as a consequence, $\|t^+\|_{\mathcal{G}''} =$

$a\ \|t_1^+\|_{\mathcal{G}''} \dots \|t_k^+\|_{\mathcal{G}''}$. By induction hypothesis, $\|t_j^+\|_{\mathcal{G}''}(u') = \texttt{bot}$, therefore $\|t^+\|_{\mathcal{G}''}(u) = \texttt{bot}$. □

As a corollary, one can state the following.

Theorem 3.4.11 (IO-Simulation)**.**

$$\|\mathcal{G}''\|_{[\texttt{bot}\mapsto\bot]} = \|\mathcal{G}\|_{\mathrm{IO}}.$$

We summarise the IO-simulation algorithm below.

Input: A scheme $\mathcal{G}$.

1. Compute the morphism $\Phi^{\mathcal{G}}$.
2. Embed $\Phi^{\mathcal{G}}$ into $\mathcal{G}$, obtaining $\mathcal{G}'$.
3. Compute the scheme $\mathcal{G}''$ obtained from $\mathcal{G}$ by changing the rewrite rule associated with any nonterminal $F^{h_1,\dots,h_k}$ such that $\Phi^{\mathcal{G}}(F)\ h_1 \dots h_k = x_\bot$, by $F^{h_1,\dots,h_k}\ \vec{x_1} \dots \vec{x_k} \to \texttt{bot}$.

Output: $\mathcal{G}''$ such that $\|\mathcal{G}''\|_{[\texttt{bot}\mapsto\bot]} = \|\mathcal{G}\|_{\mathrm{IO}}$.

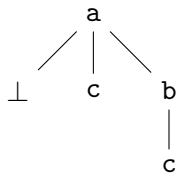
This algorithm is in n-EXPTIME, where n is the order of the scheme and the size of the new scheme is an exponential tower of height n with respect to the one of the original scheme. The order of the new scheme is still n.

Example 3.4.1. Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be the scheme defined as follows.

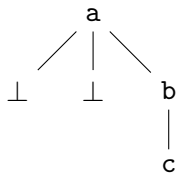
- $\Sigma = \{\texttt{a} : o^3 \to o, \texttt{b} : o \to o, \texttt{c} : o\}$,
- $\mathcal{N} = \{F : (o \to o) \to o, H, J : o \to o, S : o\}$,
- $\mathcal{V} = \{\varphi : o \to o, x : o\}$,
- $\mathcal{R}$ contains the following rewrite rules:

$$\begin{array}{lcl} S & \to & \texttt{a}\ (H\ \texttt{c})\ (J\ (H\ \texttt{c}))\ (F\ \texttt{b}), \\ H\ x & \to & F\ H, \\ J\ x & \to & \texttt{c}, \\ F\ \varphi & \to & \varphi\ \texttt{c}. \end{array}$$

The value tree and the IO-value tree of $\mathcal{G}$ are depicted below.



$\|\mathcal{G}\|$



$\|\mathcal{G}\|_{\mathrm{IO}}$

We have $\zeta(\mathtt{a}) = \mathit{tuple}(\nu_0, \nu_1, \nu_2, \nu_3)$, such that

- $\nu_0 = \mathtt{fin}$,
- $\nu_1(x_\Delta) = \mathtt{fin}$ and $\nu_1(x_1) = \mathtt{inf}$ for all $x_1 \neq x_\Delta$,
- $\nu_2(x_\Delta, x_\Delta) = \mathtt{fin}$ and $\nu_2(x_1, x_2) = \mathtt{inf}$ for all $(x_1, x_2) \neq (x_\Delta, x_\Delta)$,
- $\nu_3(x_\Delta, x_\Delta, x_\Delta) = x_\Delta$ and $\nu_3(x_1, x_2, x_3) = x_\infty$ for all $(x_1, x_2, x_3) \neq (x_\Delta, x_\Delta, x_\Delta)$.

We have $\zeta(\mathtt{b}) = (\mathtt{fin}, f_\mathtt{b})$ with $f_\mathtt{b}(x_\Delta) = x_\Delta$ and $f_\mathtt{b}(x_\bot) = f_\mathtt{b}(x_\infty) = x_\infty$, and $\zeta(\mathtt{c}) = x_\Delta$.

Then, after computing $\Phi^{\mathcal{G}}$, we obtain that:

- $\Phi^{\mathcal{G}}(S) = x_\infty$,
- $\Phi^{\mathcal{G}}(H) = (\mathtt{fin}, f_H)$ such that for all x, $f_H(x) = x_\bot$,
- $\Phi^{\mathcal{G}}(J) = (\mathtt{fin}, f_J)$ such that $f_J(x_\Delta) = x_\Delta$, and for all $x \neq x_\Delta$, $f_J(x) = x_\bot$,
- $\Phi^{\mathcal{G}}(F) = (\mathtt{fin}, f_F)$ such that for all $f \in X \to X$, $f_F\big((\mathtt{inf}, f)\big) = x_\bot$, and $f_F\big((\mathtt{fin}, f)\big) = f(x_\Delta)$.

As stated previously, for all term $t : o$, if $\Phi^{\mathcal{G}}(t) = x_\bot$, then $\|t\|_{\mathrm{IO}} = \bot$. For example,

$$\Phi^{\mathcal{G}}\big(J\ (H\ c)\big) = f_J\big(f_H(x_\Delta)\big) = f_J(x_\bot) = x_\bot.$$

After embedding the morphism, we obtain the following scheme. We just describe the rewrite rules, and we omit the annotated versions of F that do not appear in the productions of the rules (there are 54 different annotated versions of F).

$$S \ \to\ \mathtt{a}^{x_\bot, x_\bot, x_\Delta}\ (H^{x_\Delta}\ \mathtt{c})\ (J^{x_\bot}\ (H^{x_\Delta}\ \mathtt{c}))\ (F^{(\mathtt{fin}, f_\mathtt{b})}\ \mathtt{b}^{x_\bot}\ \mathtt{b}^{x_\infty}\ \mathtt{b}^{x_\Delta}),$$

$$\begin{array}{lcl}
H^{x_\bot}\ x & \to & F^{(\mathtt{fin}, f_H)}\ H^{x_\bot}\ H^{x_\infty}\ H^{x_\Delta}, \\
H^{x_\infty}\ x & \to & F^{(\mathtt{fin}, f_H)}\ H^{x_\bot}\ H^{x_\infty}\ H^{x_\Delta}, \\
H^{x_\Delta}\ x & \to & F^{(\mathtt{fin}, f_H)}\ H^{x_\bot}\ H^{x_\infty}\ H^{x_\Delta}, \\
 & & \\
J^{x_\bot}\ x & \to & \mathtt{c}, \\
J^{x_\infty}\ x & \to & \mathtt{c}, \\
J^{x_\Delta}\ x & \to & \mathtt{c}, \\
 & & \\
F^{(\mathtt{fin}, f_b)}\ \varphi^{x_\bot}\ \varphi^{x_\infty}\ \varphi^{x_\Delta} & \to & \varphi^{x_\Delta}\ \mathtt{c}, \\
F^{(\mathtt{fin}, f_H)}\ \varphi^{x_\bot}\ \varphi^{x_\infty}\ \varphi^{x_\Delta} & \to & \varphi^{x_\Delta}\ \mathtt{c}.
\end{array}$$

We operate then a second transformation, that do the following.

- It introduce new rewrite rules to remove the annotations on the terminals symbols (therefore the annotated terminal symbols becomes non-terminal).
- For all annotated non-terminal symbol $N^{d_1,\dots,d_k}$ such that $\Phi^{\mathcal{G}}(N)d_1,\dots,d_k = x_\perp$, the rewrite rules associated with $N^{d_1,\dots,d_k}$ becomes:

$$N^{d_1,\dots,d_k}\ \vec{x_1}\dots\vec{x_k} \to \texttt{bot}.$$

We show the rewrite rules of this new scheme, once again we only represent the rules associated with symbols that appear in the production of the rewrite rules.

$$S \to \texttt{a}^{x_\perp,x_\perp,x_\Delta}\ (H^{x_\Delta}\ \texttt{c})\ (J^{x_\perp}\ (H^{x_\Delta}\ \texttt{c}))\ (F^{(\texttt{fin},f_\texttt{b})}\ \texttt{b}^{x_\perp}\ \texttt{b}^{x_\infty}\ \texttt{b}^{x_\Delta}),$$

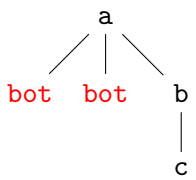
$$\begin{aligned}
H^{x_\perp}\ x &\to \texttt{bot},\\
H^{x_\infty}\ x &\to \texttt{bot},\\
H^{x_\Delta}\ x &\to \texttt{bot},
\end{aligned}$$

$$\begin{aligned}
J^{x_\perp}\ x &\to \texttt{bot},\\
J^{x_\infty}\ x &\to \texttt{bot},\\
J^{x_\Delta}\ x &\to \texttt{c},
\end{aligned}$$

$$\begin{aligned}
F^{(\texttt{fin},f_b)}\ \varphi^{x_\perp}\ \varphi^{x_\infty}\ \varphi^{x_\Delta} &\to \varphi^{x_\Delta}\ \texttt{c},\\
F^{(\texttt{fin},f_H)}\ \varphi^{x_\perp}\ \varphi^{x_\infty}\ \varphi^{x_\Delta} &\to \texttt{bot},
\end{aligned}$$

$$\begin{aligned}
\texttt{a}^{x_\perp,x_\perp,x_\Delta}\ \eta_1\ \eta_2\ \eta_3 &\to \texttt{a}\ \eta_1\ \eta_2\ \eta_3,\\
\texttt{b}^{x_\perp}\ \eta_1 &\to \texttt{b}\ \eta_1,\\
\texttt{b}^{x_\infty}\ \eta_1 &\to \texttt{b}\ \eta_1,\\
\texttt{b}^{x_\Delta}\ \eta_1 &\to \texttt{b}\ \eta_1.
\end{aligned}$$

The value tree and the IO-value tree of this new scheme (denoted $\mathcal{G}''$) are equals, and equals to $(\|\mathcal{G}\|_{\text{IO}})_{[\perp\mapsto\texttt{bot}]}$, as depicted below.



$$\|\mathcal{G}''\| = \|\mathcal{G}''\|_{\text{IO}} = (\|\mathcal{G}\|_{\text{IO}})_{[\perp\mapsto\texttt{bot}]}$$

3.5. Simulation of the OI evaluation policy

In this section, we solve the OI simulation problems. We use a scheme transformation inspired from the standard continuation passing style transformation of programming languages.

Fix a recursion scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$. Our goal is to define another scheme $\overline{\mathcal{G}} = \langle \overline{\mathcal{V}}, \overline{\Sigma}, \overline{\mathcal{N}}, \overline{\mathcal{R}}, I \rangle$, with $\Sigma \subseteq \overline{\Sigma}$, such that $\|\overline{\mathcal{G}}\|_{\mathrm{IO}} = \|\mathcal{G}\|$.

The idea is to add an extra argument $(\mathtt{p} : o)$ to each non-terminal F (hence the types are changed and the order is augmented by 1). We feed this argument to the outermost non-terminal, and duplicate it to subterms only if the head of the term is a terminal. Hence all derivations will be OI-derivations. Similar transformations can be found in other contexts (see for example [Plo75]). Let us illustrate this idea with the following example.

Example 3.5.1. Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ be the order-2 recursion scheme with

- $\Sigma = \{\mathtt{a} : o^2 \to o, \mathtt{b} : o\}$,
- $\mathcal{N} = \{S : o, F : o^2 \to o, H : o\}$,
- $\mathcal{V} = \{x, y : o\}$,
- and $\mathcal{R}$ contains the following rewrite rules:

$$S \;\to\; F\ H\ (\mathtt{a}\ \mathtt{b}\ \mathtt{b}), \qquad F\ x\ y \;\to\; y, \qquad H \;\to\; H.$$

Then we have $\|\mathcal{G}\| = \mathtt{a}\ \mathtt{b}\ \mathtt{b}$ while $\|\mathcal{G}\|_{\mathrm{IO}} = \bot$. Indeed, the only IO-derivation is:

$$S \;\to_{\mathrm{IO}}\; F\ H\ (\mathtt{a}\ \mathtt{b}\ \mathtt{b}) \;\to_{\mathrm{IO}}\; F\ H\ (\mathtt{a}\ \mathtt{b}\ \mathtt{b}) \;\to_{\mathrm{IO}} \cdots .$$

The order-3 recursion scheme $\overline{\mathcal{G}} = \langle \overline{\mathcal{V}}, \Sigma \uplus \{\mathtt{p} : o\}, \overline{\mathcal{N}}, \overline{\mathcal{R}}, I \rangle$ is given by the following.

- $\overline{\mathcal{N}} = \{I : o, \overline{S}, \overline{a} : (o \to o)^2 \to o \to o, \overline{\mathtt{b}} : o \to o, \overline{F} : (o \to o)^2 \to o \to o, \overline{H} : o \to o\}$
- $\overline{\mathcal{V}} = \{\ell : o, \overline{x}, \overline{y} : o \to o\}$,
- $\overline{\mathcal{R}}$ contains the following rewrite rules:

$$\begin{array}{lllllllll} I & \to & \overline{S}\ \mathtt{p}, & \overline{S}\ \ell & \to & \overline{F}\ \overline{H}\ (\overline{\mathtt{a}}\ \overline{\mathtt{b}}\ \overline{\mathtt{b}})\ \mathtt{p}, & \overline{F}\ \overline{x}\ \overline{y}\ \ell & \to & \overline{y}\ \mathtt{p} \\ \overline{H}\ \ell & \to & \overline{H}\ \mathtt{p} & \overline{\mathtt{b}}\ \ell & \to & \mathtt{b} & \overline{\mathtt{a}}\ \overline{x}\ \overline{y}\ \ell & \to & \mathtt{a}\ (\overline{x}\ \mathtt{p})\ (\overline{y}\ \mathtt{p}) \end{array}$$

Note that in the term $\overline{F}\ \overline{H}\ (\overline{\mathtt{a}}\ \overline{\mathtt{b}}\ \overline{\mathtt{b}})\ \mathtt{p}$, the subterm $\overline{H}$ is no longer a redex since it lacks its last argument, hence it cannot be rewritten. We have the following IO-derivation:

$$I \;\to_{\mathrm{IO}}\; \overline{S}\ \mathtt{p} \;\to_{\mathrm{IO}}\; \overline{F}\ \overline{H}\ (\overline{\mathtt{a}}\ \overline{\mathtt{b}}\ \overline{\mathtt{b}})\ \mathtt{p} \;\to_{\mathrm{IO}}\; \overline{\mathtt{a}}\ \overline{\mathtt{b}}\ \overline{\mathtt{b}}\ \mathtt{p} \;\to_{\mathrm{IO}} \mathtt{a}\ (\overline{\mathtt{b}}\ \mathtt{p})\ (\overline{\mathtt{b}}\ \mathtt{p}) \to_{\mathrm{IO}} \mathtt{a}\ \mathtt{b}\ (\overline{\mathtt{b}}\ \mathtt{p}) \to \mathtt{a}\ \mathtt{b}\ \mathtt{b}.$$

Therefore $\|\overline{\mathcal{G}}\|_{\mathrm{IO}} = \|\overline{\mathcal{G}}\| = \mathtt{a}\ \mathtt{b}\ \mathtt{b} = \|\mathcal{G}\|$.

Let us formalise this construction.

Types. For all type τ, we inductively define the type $\overline{\tau}$ by:

- $\overline{o} = o \to o$,
- and $\overline{\tau_1 \to \tau_2} = \overline{\tau_1} \to \overline{\tau_2}$.

In particular, if $\tau = \tau_1 \to \cdots \to \tau_k \to o$ then

$$\overline{\tau} = \overline{\tau_1} \to \cdots \to \overline{\tau_k} \to o \to o.$$

Note that for all τ, $\mathit{order}(\overline{\tau}) = \mathit{order}(\tau) + 1$ and $\mathit{arity}(\overline{\tau}) = \mathit{arity}(\tau) + 1$.

Symbols. For all $x : \tau \in \mathcal{V}$ let $\overline{x} : \overline{\tau} \in \overline{\mathcal{V}}$. Let $ar_{\max}$ be the maximum arity of the terminal symbols. Let $\eta_1, \ldots, \eta_{ar_{\max}} : o \to o$ and $\ell : o$ in $\overline{\mathcal{V}}$.

For all $a : \tau \in \Sigma$ define $\overline{a} : \overline{\tau}$ as a *non-terminal* of $\overline{\mathcal{N}}$. For all $F : \tau \in \mathcal{N}$ define $\overline{F} : \overline{\tau}$ as a non-terminal of $\overline{\mathcal{N}}$. Furthermore we add the nonterminal $I : o$ in $\overline{\mathcal{N}}$. Note that I is the only symbol in $\overline{\mathcal{N}}$ of type o.

Finally we define $\overline{\Sigma} = \Sigma \uplus \{\mathtt{p} : o\}$.

Term transformation. Let $t : \tau \in \mathcal{T}(\mathcal{V} \uplus \Sigma \uplus \mathcal{N})$, we define inductively the term $\overline{t} : \overline{\tau} \in \mathcal{T}(\overline{\mathcal{V}} \uplus \overline{\mathcal{N}})$:

- If $t = \alpha \in \mathcal{V} \uplus \Sigma \uplus \mathcal{N}$, we let $\overline{t} = \overline{\alpha} \in \overline{\mathcal{V}} \uplus \overline{\Sigma} \uplus \overline{\mathcal{N}}$,
- if $t = t_1\ t_2 : \tau$ then $\overline{t} = \overline{t_1}\ \overline{t_2}$.

For all $e \in \mathcal{T}(\mathcal{V} \uplus \Sigma \uplus \mathcal{N})$, $x \in \mathcal{V}$, and $t \in \mathcal{T}(\Sigma \uplus \mathcal{N})$, a straightforward induction on e shows that $\overline{e_{[x \mapsto t]}} = \overline{e}_{\left[\overline{x} \mapsto \overline{t}\right]}$.

Rewrite rules. Let $F\ x_1\ \ldots\ x_k \to e$ be a rewrite rule of $\mathcal{R}$. We define the rule

$$\overline{F}\ \overline{x_1}\ \ldots\ \overline{x_k}\ \ell\ \to\ \overline{e}\ \mathtt{p} \in \overline{\mathcal{R}}.$$

Let $a \in \Sigma$ of arity k, we define the rule

$$\overline{a}\ \eta_1\ \ldots\ \eta_k\ \ell\ \to\ a\ (\eta_1\ \mathtt{p})\ \ldots\ (\eta_k\ \mathtt{p}) \in \overline{\mathcal{R}}.$$

We also add the rule $I\ \to\ \overline{S}\ \mathtt{p}$ to $\overline{\mathcal{R}}$.

Proposition 3.5.1.

Any derivation of $\overline{\mathcal{G}}$ is an OI and an IO-derivation. Hence $\|\overline{\mathcal{G}}\|_{\mathrm{IO}} = \|\overline{\mathcal{G}}\|$.

Claim 3.5.2. For all term $t \in \mathcal{T}(\overline{\mathcal{N}} \uplus \overline{\mathcal{V}})$ that does not contain I, t has type $\overline{\tau}$ for some τ. In particular, t has not type o. ⌟

Proof. The proof proceeds by induction on the size of t. If $t = \overline{\alpha} \in \overline{\mathcal{N}} \uplus \overline{\mathcal{V}}$ then $t : \overline{\tau}$ with $\alpha : \tau$. If $t = t_1\ t_2$ then $t_1 : \overline{\tau}_1 \to \ldots \to \overline{\tau}_k \to o \to o$ for some $k \geq 0$. Assume that $k = 0$ (*i.e.* $t_1 : o \to o$). Then $t_2 : o$ but $o \neq \overline{\tau}$ for all τ. Therefore $k > 0$ and $t_1\ t_2 : \overline{\tau}_2 \to \cdots \to \overline{\tau}_k \to o \to o = \overline{(\tau_2 \to \cdots \to \tau_k \to o)}$. □

Claim 3.5.3. If $\overline{S}\ \mathtt{p} \to^* t$, then all redexes of t are of the form $\overline{\alpha}\ t_1 \ldots t_k\ \mathtt{p}$ such that for all i, $t_i \in \mathcal{T}(\overline{\mathcal{N}})$. ⌟

Proof. We prove this result by induction on the size of the derivation. It is true for the term $\overline{S}\ \mathtt{p}$. Assume that it is true for t and let t' such that $t \to t'$. Then either we have (1) $t = C[\overline{F}\ t_1 \ldots t_k\ \mathtt{p}]$ with $F \in \mathcal{N}$ and $t' = C[e_{[\forall i\ \overline{x_i} \mapsto t_i]}\ \mathtt{p}]$ with $F\ x_1 \ldots x_k \to e \in \mathcal{R}$, or (2) $t = C[\overline{a}\ t_1 \ldots t_k\ \mathtt{p}]$ with $a \in \Sigma$ and $t' = C[a\ (\overline{t}_1\ \mathtt{p}) \ldots (\overline{t}_k\ \mathtt{p})]$. We prove the result by induction on $C[\bullet]$, and since the induction step is straightforward, we just show the basis case.

(1) Assume that $t = \overline{F}\ t_1 \ldots t_k\ \mathtt{p}$ with $F \in \mathcal{N}$ and $t' = e_{[\forall i\ \overline{x_i} \mapsto t_i]}\ \mathtt{p}$ with $F\ x_1 \ldots x_k \to e \in \mathcal{R}$. Since $e \in \mathcal{T}(\overline{\mathcal{V}} \uplus \overline{\mathcal{N}})$ and for all i, $t_i \in \mathcal{T}(\overline{\mathcal{N}})$, $e_{[\forall i\ \overline{x_i} \mapsto t_i]} \in \mathcal{T}(\overline{\mathcal{N}})$. Therefore it does not contain any subterm of type o. The only redex that t' contains is t' itself which satisfies the property.

(2) Assume $t = \overline{a}\ t_1 \cdots t_k\ \mathtt{p}$ with $a \in \Sigma$ and $t' = a\ (t_1\ \mathtt{p}) \ldots (t_k\ \mathtt{p})$. For all i, $t_i \in \mathcal{T}(\overline{\mathcal{N}})$ therefore the only redexes of t' are $\overline{t}_i\ \mathtt{p}$ for all i, which satisfy the property.

□

Proof of Proposition **3.5.1**. We show that if $\overline{S}\ \mathtt{p} \to^* t$ then if $t = C[r]$ with r a redex, then r is an IO redex and $C[\bullet]$ is an OI context.

Since $r = \overline{\alpha}\ t_1 \ldots t_k\ \mathtt{p}$ such that for all i, then by Claim **3.5.3** $t_i \in \mathcal{T}(\overline{\mathcal{N}})$, therefore by Claim **3.5.2** t_i does not contain any redex, thus r is an IO redex.

Assume by contradiction that $C[\bullet]$ is not an OI context, therefore $C[\bullet] = C_1[F\ t_1 \ldots C_2[\bullet] \ldots t_k]$ such that the term $r' = F\ t_1 \ldots C_2[r] \ldots t_k$ is a redex. That would mean that $t = C_1[r']$ with r' a redex that is not an IO redex, this is not possible due to the previous statement. □

Proposition 3.5.4.

$$\|\mathcal{G}\| = \|\overline{\mathcal{G}}\|.$$

We introduce arelation R between ground typed terms of $\mathcal{G}$ and of $\overline{\mathcal{G}}$ such that, intuitively, tRt' if t' is obtained from $\overline{t}\ \mathtt{p}$ by rewriting some non terminals $\overline{a}$ with $a \in \Sigma$.

Let $t : o \in \mathcal{T}(\Sigma \uplus \mathcal{N})$, then:

- $t\ R\ (\overline{t}\ \mathtt{p})$,

- let $t = a\ t_1 \dots t_k$ with $a \in \Sigma$, then for all $t'_1, \dots, t'_k$, such that $t_i\ R\ t'_i$ for all i, $t\ R\ (a\ t'_1 \dots t'_k)$.

Observe that for all t_1, t_2, t', if $t_1\ R\ t'$ and $t_2\ R\ t'$ then $t_1 = t_2$. Indeed, first remark that for all term t of $\mathcal{G}$, $\overline{t}$ does not contain any terminal symbol. Then we prove this result by induction on the structure of t'. Either $t' = (\overline{t}\ \mathtt{p})$ in which case $\overline{t} = \overline{t}_1 = \overline{t}_2$ which implies $t_1 = t_2$. Or $t' = a\ t'_1 \dots t'_k$ then $t_1 = a\ t^1_1 \dots t^1_k$, $t_2 = a\ t^2_1 \dots t^2_k$ such that for all i, $t^1_i\ R\ t'_i$ and $t^2_i\ R\ t'_i$. By induction hypothesis $t^1_i = t^2_i$, therefore $t_1 = t_2$.

For all term t, we write $R_t = \{t' \mid t\ R\ t'\}$.

Example 3.5.2. Take the term $t = \mathtt{a\ b}$ with $\mathtt{a} : o \to o$ and $\mathtt{b} : o$ in Σ. R_t contains the following three terms:

$$\overline{\mathtt{a}}\ \overline{\mathtt{b}}\ \mathtt{p}, \qquad \mathtt{a}\ (\overline{\mathtt{b}}\ \mathtt{p}), \qquad \mathtt{a\ b}.$$

Claim 3.5.5. Let $t : o$ a term of $\mathcal{G}$, $t' \in R_t$, and s' be such that $t' \to_{\mathrm{OI}} s'$ in $\overline{\mathcal{G}}$.
If $t' = C[\overline{a}\ t'_1 \dots t'_k\ \mathtt{p}]$ and $s' = C[a\ (t'_1\ \mathtt{p}) \dots (t'_k\ \mathtt{p})]$, then $s' \in R_t$. ⌟

Proof. The proof proceeds by induction on $C[\bullet]$. If $t' = \overline{a}\ t'_1 \dots t'_k\ \mathtt{p}$ then $t' = \overline{t}$, therefore $t = a\ t_1 \dots t_k$ with for all i $t'_i = \overline{t}_i$. Since $s' = a\ (t'_1\ \mathtt{p}) \dots (t'_k\ \mathtt{p})$, we have $t\ R\ s'$.

If $C[\bullet] = b\ r'_1 \dots r'_{i-1} C'[\bullet] r'_{i+1} \dots r'_k$, then since s' has been obtained by an OI rewrtie, $b \in \Sigma$. Therefore $t = b\ r_1 \dots r_k$ with $r_j\ R\ r'_j$ for all $j \neq i$, and $r_i\ R\ C'[\overline{a}\ t'_1 \dots t'_k\ \mathtt{p}]$. By induction hypothesis, $r_i\ R\ C'[a\ (t'_1\ \mathtt{p}) \dots (t'_k\ \mathtt{p})]$. Since $s' = b\ r'_1 \dots r'_{i-1} C'[a\ (t'_1\ \mathtt{p}) \dots (t'_k\ \mathtt{p})] r'_{i+1} \dots r'_k$, we have $s' \in R_t$. □

Claim 3.5.6. Let $t : o$ a term of $\mathcal{G}$, $t' \in R_t$, and s' be such that $t' \to_{\mathrm{OI}} s'$ in $\overline{\mathcal{G}}$.
If $t' = C[\overline{F}\ t'_1 \dots t'_k\ \mathtt{p}]$ and $s' = C[e_{[\forall i\ \overline{x}_i \mapsto t'_i]}\ \mathtt{p}]$, with $F\ x_1 \dots x_k \to e \in \mathcal{R}$, then there exists s such that $t \to_{\mathrm{OI}} s$ and $s' \in R_s$. ⌟

Proof. The proof proceeds by induction of $C[\bullet]$. If $C[\bullet] = \bullet$. Then $t' = \overline{F}\ t'_1 \dots t'_k\ \mathtt{p}$ therefore $t = F\ t_1 \dots t_k$ such that for all i, $t'_i = \overline{t_i}$. We have $t \to e_{[\forall i\ x_i \mapsto t_i]}$. If we let $s = e_{[\forall i\ x_i \mapsto t_i]}$, we have $s' = \overline{s}\ \mathtt{p}$, therefore $s' \in R_s$.

If $C[\bullet] = a\ r'_1 \dots r'_{i-1} C'[\bullet] r'_{i+1} \dots r'_k$ with $a \in \Sigma$. Then $t = a\ r_1 \dots r_k$ with $r_j\ R\ r'_j$ for all $j \neq i$ and $r_i\ R\ C'[\overline{F}\ t'_1 \dots t'_k\ \mathtt{p}]$. By induction hypothesis, there exists s_i such that $r_i \to_{\mathrm{OI}} s_i$ and $s_i\ R\ C'[e_{[\forall i\ x_i \mapsto t'_i]}]$. Since $s' = a\ r_1 \dots r_{i-1} C'[e_{[\forall i\ x_i \mapsto t'_i]}] r_{i+1} \dots r_k$, if we set $s = a\ r_1 \dots r_{i-1}\ s_i\ r_{i+1} \dots r_k$, we have $s' \in R_s$. □

Let $Acc^{\mathrm{OI}}_{\mathcal{G}} = \{t \mid S \to^*_{\mathrm{OI}} t\}$ and $Acc^{\mathrm{OI}}_{\overline{\mathcal{G}}} = \{t' \mid \overline{S}\ \mathtt{p} \to^*_{\mathrm{OI}} t'\}$ (we omit the term I for ease of exposition). The following claim is a corollary of the two previous claims.

Claim 3.5.7. If $\overline{S}\ \mathtt{p} \to^*_{\mathrm{OI}} t'$ in $\overline{\mathcal{G}}$ then there exists t such that $S \to^*_{\mathrm{OI}} t$ in $\mathcal{G}$ such that $t' \in R'_t$, which is equivalent to:

$$Acc^{\mathrm{OI}}_{\overline{\mathcal{G}}} \subseteq \bigcup_{t \in Acc^{\mathrm{OI}}_{\mathcal{G}}} R_t.$$

⌟

For all $t : o \in \mathcal{T}(\Sigma \uplus \mathcal{N})$, we define the term $t^\square$ of $\overline{\mathcal{G}}$ as follows.

- If $t = F\ t_1 \dots t_k$ with $F \in \Sigma$, $t^\square = \overline{t}\ \mathtt{p}$.
- If $t = a\ t_1 \dots t_k$ with $a \in \Sigma$, then $t^\square = a\ t_1^\square \dots t_k^\square$.

Observe that $t^\square \in R_t$, and $(t^\square)^\perp = t^\perp$. Furthermore a simple induction shows that for all $t' \in R_t$, $(t')^\perp \sqsubseteq (t^\square)^\perp$. Intuitively $t^\square$ corresponds to the term of R_t where the maximum of symbols of the form $\overline{a}$, with $a \in \Sigma$, have been rewritten.

Claim 3.5.8. For all $t : o \in \mathcal{T}(\Sigma \uplus \mathcal{N})$, for all $t' \in R_t$, $t' \to_{\mathrm{OI}}^* t^\square$.

⌟

Proof. We prove this result by induction on t. If t is a redex, then $t' = t^\square = \overline{t}\ \mathtt{p}$. If $t = a\ t_1 \dots t_k$ with $a \in \Sigma$.

- Either $t' = a\ t'_1 \dots t'_k$ such that $t_i\ R\ t'_i$ for all i, by induction hypothesis $t'_i \to_{\mathrm{OI}}^* t_i\square$ therefore $t' \to_{\mathrm{OI}}^* t^\square$.
- Or $t' = \overline{t}\ \mathtt{p} = \overline{a}\ \overline{t_1} \dots \overline{t_k}\ \mathtt{p}$, in which case $t' \to_{\mathrm{OI}} a(\overline{t_1}\ \mathtt{p}) \dots (\overline{t_k}\ \mathtt{p})$. Therefore, if for all i we let $t'_i = \overline{t_i}\ \mathtt{p}$, we have $t_i\ R\ t'_i$ for all. By induction hypothesis $t'_i \to_{\mathrm{OI}}^* t_i\square$ therefore $t' \to_{\mathrm{OI}}^* t^\square$.

□

Claim 3.5.9. For all $t : o \in \mathcal{T}(\Sigma \uplus \mathcal{N})$, if $t \to_{\mathrm{OI}} s$ then there exists $s' \in R_s$ such that $t^\square \to_{\mathrm{OI}} s'$.

⌟

Proof. If $t = F\ t_1 \dots t_k$ is a redex, then $s = e_{[\forall i\ x_i \mapsto t_i]}$ with $F\ x_1 \dots x_k \to e \in \mathcal{R}$. We have $t^\square = \overline{F}\ \overline{t_1} \dots \overline{t_k}\ \mathtt{p}$, therefore $t^\square \to_{\mathrm{OI}} \overline{e}_{\left[\forall i\ \overline{x_i} \mapsto \overline{t_i}\right]}\ \mathtt{p} = \overline{s}\ \mathtt{p} \in R_s$.

If $t = a\ t_1 \dots t_k$ with $a \in \Sigma$, There exists i such that $t' = at_1 \dots s_i \dots t_k$ with $t_i \to_{\mathrm{OI}} s_i$. We have $t^\square = a\ t_1^\square \dots t_k^\square$, by induction hypothesis, $t_i^\square \to_{\mathrm{OI}} s'_i \in R_{s_i}$, therefore $t^\square \to_{\mathrm{OI}} a\ t_1^\square \dots s'_i \dots t_k^\square \in R_s$. □

The following is a consequence of those claims.

Claim 3.5.10. for all $t : o \in \mathcal{T}(\Sigma \uplus \mathcal{N})$ such that $S \to^* t$ in $\mathcal{G}$, $\overline{S} \to^* t^\square$ in $\overline{\mathcal{G}}$:

$$\{t^\square \mid t \in Acc_{\mathcal{G}}^{\mathrm{OI}}\} \subseteq Acc_{\overline{\mathcal{G}}}^{\mathrm{OI}}.$$

⌟

Proof of Proposition **3.5.4**. For all set X of terms of type o, we write $(X)^\perp = \{t^\perp \mid t \in X\}$.

Recall that $\|\mathcal{G}\|$ is the supremum of $(Acc_{\mathcal{G}}^{\mathrm{OI}})^\perp$ and $\|\overline{\mathcal{G}}\|$ is the supremum of $(Acc_{\overline{\mathcal{G}}}^{\mathrm{OI}})^\perp$. Since for all t, $(t^\square)^\perp = t^\perp$, we have $(Acc_{\mathcal{G}}^{\mathrm{OI}})^\perp = (\{t^\square \mid t \in Acc_{\mathcal{G}}^{\mathrm{OI}}\})^\perp$. Therefore, Claim **3.5.10** implies that $\|\mathcal{G}\| \sqsubseteq \|\overline{\mathcal{G}}\|$. Furthermore, since for all t, $(t^\square)^\perp$ is the maximum of $(R_t)^\perp$, the supremum of $\left(\bigcup_{t \in Acc_{\mathcal{G}}^{\mathrm{OI}}} R_t\right)^\perp$ is also the supremum of $(\{t^\square \mid t \in Acc_{\mathcal{G}}^{\mathrm{OI}}\})^\perp$. Hence, from Claim **3.5.7**, one gets that $\|\overline{\mathcal{G}}\| \sqsubseteq \|\mathcal{G}\|$. Therefore $\|\overline{\mathcal{G}}\| = \|\mathcal{G}\|$. □

As a corollary one can state the following.

Theorem 3.5.11 (OI Simulation)**.**

$$\|\mathcal{G}\| = \|\overline{\mathcal{G}}\|_{\mathrm{IO}}.$$

In Example 3.5.1 the value tree of the original scheme, was a finite tree `a b b`, it could have been done with an order 0 scheme with one non-terminal: $S \to \mathtt{a}\ \mathtt{b}\ \mathtt{b}$. It is actually easy to show that the set of trees produced by order 0 scheme is the set of all ***regular trees*** [CDG+07] (a tree is regular if it contains only finitely many subtrees).

We show in Example 3.5.3 an order-1 scheme whose OI value tree cannot be produced by any order 1 scheme using only IO-derivations.

Example 3.5.3. Let $\mathcal{G}_0 = \langle \Sigma_0, \mathcal{N}_0, \mathcal{V}_0, \mathcal{R}_0, S_0 \rangle$ be the order 1 scheme such that:

- $\Sigma_0 = \{\mathtt{a} : o \to o, \mathtt{b} : o^2 \to o\}$,
- $\mathcal{N}_0 = \{S_0, B : o, F : o \to o\}$,
- $\mathcal{V}_0 = \{x : o\}$,
- and $\mathcal{R}_0$ contains the following rewrite rules:

$$S_0 \ \to \ F\ B, \qquad F\ x \ \to \ \mathtt{b}\ x\ (F\ (\mathtt{a}\ x)), \qquad B \ \to \ \mathtt{b}\ B\ B.$$

The tree $\|\mathcal{G}_0\|$ is depicted in Figure 3.3. The tree is not regular, then it cannot be generated by an order 0 scheme. We can show the following proposition.

Proposition 3.5.12.

Let T be the value tree of the order-1 scheme $\mathcal{G}_0$ defined above. There exists no order-1 scheme $\mathcal{G}$ such that $\|\mathcal{G}\|_{\mathrm{IO}} = T$.

We reason by contradiction, and assume there exists an order-1 scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ such that $\|\mathcal{G}_0\| = \|\mathcal{G}\|_{\mathrm{IO}}$. We will then transform $\mathcal{G}$ into a scheme $\mathcal{G}'$ of order 0 whose IO value tree is $\|\mathcal{G}_0\|$ which will raise a contradiction.

Intuitively, $\mathcal{G}'$ is obtained from $\mathcal{G}$ by turning all non-terminal symbols $F : o^k \to o$ into an order 0 symbol $F' : o$. The rewrite rule associated with F' is obtained from the one of F by removing the argument to the nonterminals and replacing the variables by a symbol `c`.

Formally, we define $\mathcal{N}' = \{F' : o \mid F \in \mathcal{N}\}$ and $\Sigma' = \Sigma \uplus \{\mathtt{c} : o\}$, `c` being a fresh symbol. Given a ground term $t : o \in \mathcal{T}(\mathcal{N} \uplus \Sigma \uplus \mathcal{V})$ we define the term $t' : o \in \mathcal{T}(\Sigma' \uplus \mathcal{N}')$

by induction: $x' = \mathsf{c}$ for all $x : o \in \mathcal{V}$, $(F\ t_1 \dots t_k)' = F'$ for all $F \in \mathcal{N}$, and $(a\ t_1 \dots t_k)' = a\ t'_1 \dots t'_k$ for all $a \in \Sigma$.

We define $\mathcal{R}' = \{F' \to e' \mid F\ x_1 \dots x_k \to e \in \mathcal{R}\}$ and $\mathcal{G}' = \langle \Sigma', \mathcal{N}', \{\ \}, \mathcal{R}', S' \rangle$. Note that $\mathcal{G}'$ is an order-0 scheme.

We prove that when the order-1 scheme $\mathcal{G}$ produces a tree whose subtrees are all infinite with an IO derivation, it never uses the argument of the non-terminal symbols. Therefore the associated order-0 scheme $\mathcal{G}'$, obtained by getting rid of the arguments of the non-terminal symbols, produces the same tree.

Let $\mathcal{I}$ be the set of ground terms t of $\mathcal{T}(\Sigma \uplus \mathcal{N})$ such that all subtrees of $\|\mathcal{G}_t\|_{\mathrm{IO}}$ are infinite.

Claim 3.5.13. Let $t_1 : o, \dots, t_k : o$ be some terms containing only terminals (*i.e.* finite trees), $F : o^k \to o \in \mathcal{N}$ and $F\ x_1 \dots x_k \to e \in \mathcal{R}$.

If $t = F\ t_1 \dots t_k \in \mathcal{I}$ then all occurrences of x_i in e appear in a subterm whose head is a non terminal. ⌟

Proof. Indeed, otherwise,$e^{\perp}_{[\forall i\ x_i \mapsto t_i]}$ would contain a finite subtree that does not contain $\perp$, and then so would do $\|\mathcal{G}_t\|_{\mathrm{IO}}$. □

Observes that for all order-0 scheme the relations $\to_{\mathrm{IO}}$, $\to_{\mathrm{OI}}$ and $\to$ are the same.

Claim 3.5.14. Given $t_1, t_2 \in \mathcal{I}$ such that $t_1 \to_{\mathrm{IO}} t_2$, then $t'_1 \to^*_{\mathrm{IO}} t'_2$. ⌟

Proof. We prove this result by induction on the structure of the context of t_1 where the rewrite takes place. Assume that t_1 is an IO-redex. Then $t_1 = F\ s_1 \dots s_k$ with $s_1 : o, \dots, s_k : o$ be some terms containing only terminals. Let $F\ x_1 \dots x_k \to e \in \mathcal{R}$. We have $t_2 = e_{[\forall i\ x_i \mapsto s_i]}$. Claim **3.5.13** implies that $e' = (e_{[\forall i\ x_i \mapsto s_i]})' = t'_2$, and since $t'_1 \to e'$, we have $t'_1 \to_{\mathrm{IO}} t'_2$.

If $t_1 = F\ s_1 \dots s_k$ with $F \in \Sigma$ is not an IO redex, then $t_2 = F\ s_1 \dots r_j \dots s_k$ with $s_j \to_{\mathrm{IO}} r_j$. Therefore $t'_1 = F \to^*_{\mathrm{IO}} F = t'_2$.

If $t_1 = a\ s_1 \dots s_k$ with $a \in \Sigma$, then $t_2 = a\ s_1 \dots r_j \dots s_k$ with $s_j \to_{\mathrm{IO}} r_j$. By induction hypothesis, $s'_j \to^*_{\mathrm{IO}} r'_j$, then $t'_1 = a\ s'_1 \dots s'_k \to^*_{\mathrm{IO}} a\ s'_1 \dots r'_j \dots s'_k = t'_2$. □

Claim 3.5.15. For all term $t : o \in \mathcal{T}(\Sigma \uplus \mathcal{N})$, if $t \in \mathcal{I}$ then $\|\mathcal{G}'_{t'}\|_{\mathrm{IO}} = \|\mathcal{G}_t\|_{\mathrm{IO}}$. ⌟

Proof. First notice that for all $t \in \mathcal{T}(\Sigma \uplus \mathcal{N})$, $t^{\perp} = (t')^{\perp}$. Then take an IO derivation $t \to_{\mathrm{IO}} t_1 \to_{\mathrm{IO}} t_2 \to_{\mathrm{IO}} \cdots$ in $\mathcal{G}$ that leads to $\|\mathcal{G}_t\|_{\mathrm{IO}}$; we have a derivation $t' \to^*_{\mathrm{IO}} t'_1 \to^*_{\mathrm{IO}} t'_2 \to^*_{\mathrm{IO}} \cdots$ in $\mathcal{G}'$ that leads to the same tree, hence $\|\mathcal{G}_t\|_{\mathrm{IO}} \sqsubseteq \|\mathcal{G}'_{t'}\|_{\mathrm{IO}}$. Since all subtrees of $\|\mathcal{G}_t\|$ are infinite, $\|\mathcal{G}_t\|$ does not contains $\perp$, therefore $\|\mathcal{G}_t\|_{\mathrm{IO}} = \|\mathcal{G}'_{t'}\|_{\mathrm{IO}}$ □

Proof of Proposition **3.5.12**. From the previous claim we can conclude that if there exists an order-1 scheme $\mathcal{G}$ such that $\|\mathcal{G}_0\| = \|\mathcal{G}\|_{\mathrm{IO}}$ then there exists an order-0 scheme $\mathcal{G}'$ such that $\|\mathcal{G}_0\| = \|\mathcal{G}'\|_{\mathrm{IO}}$, therefore $\|\mathcal{G}_0\|$ would be regular, which raises a contradiction. □

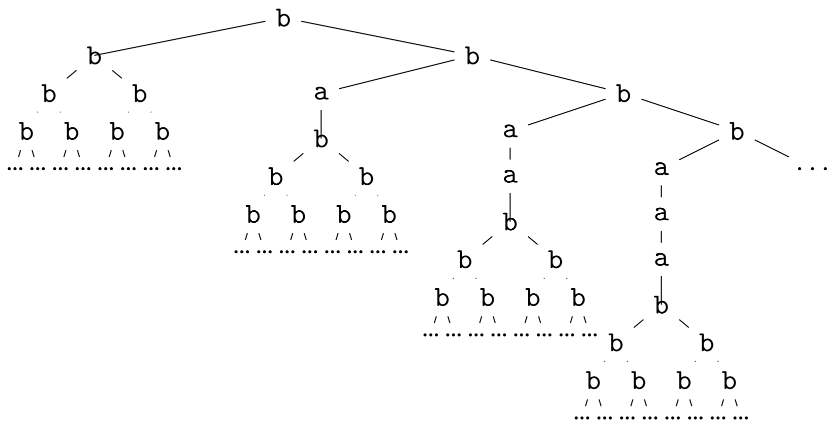


Figure 3.3.: The tree $\|\mathcal{G}_0\|$

4. Reflection and Selection

4.1. Introduction . . . 125
4.2. Generalising the notion of runs to higher order terms . . . 127
4.2.1. Parity automata revisited . . . 127
4.2.2. An example of transitions on nonterminal . . . 130
4.2.3. Runs over terms . . . 135
4.3. Reflection . . . 139
4.3.1. Automata reflection . . . 139
4.3.2. MSO reflection . . . 141
4.4. Kobayashi-Ong Result . . . 147
4.4.1. Occurrences . . . 147
4.4.2. Parity games . . . 152
4.4.3. Adding colours to annotations . . . 154
4.4.4. Kobayashi-Ong game . . . 157
4.4.5. Solving problem 8 . . . 165
4.5. Selection . . . 167
4.5.1. Constructing the decorated ARS . . . 167
4.5.2. A winning condition for derivations . . . 171
4.5.3. Solving the selection problem . . . 180

4.1. Introduction

In this chapter we study the reflection and selection problems as stated in Chapter 2. We recall these problems, both have two versions whether the input tree property is expressed as an automaton or an MSO formula.

- The *automata reflection problem* requires, given a scheme $\mathcal{G}$ and an automaton $\mathcal{A}$ to produce another scheme $\mathcal{G}'$ whose value tree is annotated by sets of states, such that each node u is annotated by the set of states from which the automaton accepts the subtree of $\|\mathcal{G}\|$ rooted on u.

- The *MSO reflection problem* requires, given a scheme $\mathcal{G}$ and an MSO formula $\varphi[x]$, to produce another scheme whose value tree is annotated by $\{0, 1\}$, such that each node u is annotated by 1 if and only if the formula holds on u in $\|\mathcal{G}\|$.

- The *automata selection problem* requires, given an automaton $\mathcal{A}$ and a scheme $\mathcal{G}$ whose value tree accepted by the automaton, to produce another scheme whose value tree is an accepting run of the automaton over $\|\mathcal{G}\|$.

- Finally, the *MSO selection problem* requires, given a scheme $\mathcal{G}$ and a formula of the form $\varphi = \exists X \psi[X]$ such that $\mathcal{G} \models \varphi$, to produce another scheme whose value tree is obtained from $\|\mathcal{G}\|$ by marking a set of nodes S such that $\psi[X]$ holds on S in $\|\mathcal{G}\|$.

We present three synthesis procedures for the automata reflection, MSO reflection and the automata selection problems. There is a straightforward equivalence between the automata selection and MSO selection, therefore the procedure for the automata selection problem induces a procedure for the MSO selection problem.

The algorithms we introduce are based on a decision procedure for automata model checking, which is a reformulation of the results of Kobayashi and Ong [KO09]. Intuitively, the idea here is, given an automaton and a scheme, to define *higher-order* transitions on non-terminal symbols allowing us to define runs of the automaton on terms, and to find a set of transitions associated with the non-terminal symbols such that for all term $t : o$, there exists an accepting run of t using the new transitions, if there exists an accepting run on the value tree of t. In order to find this set of transitions, we use a two player game, in which Eve tries to find some higher-order transitions, and Adam tries to show that the choices of Eve are not consistent with the rewrite rules.

More precisely, we proceed the following way.

1. First given a set of states Q, and a set of colours M, we consider transitions of tree automata with states Q and colours M as transitions related with types of order 0 or 1. Then we generalise the notion of transition for any higher-order type yielding the set Θ.

2. Given a typed alphabet Γ, we extend Γ to Γ_Θ that consists of the symbols of Γ annotated with transitions of Θ. While the type τ of a symbol and its associated transition must be identical, the annotated symbol has a new type τ^+ inductively defined.

3. Let t be a term on $\mathcal{T}(\Gamma)$, we specify a set $CA(t)$ of terms of $\mathcal{T}(\Gamma_\Theta)$ by structural and local constraints depending on t and Θ.

4. Given an automaton $\mathcal{A}$ with states Q and colours M, and a scheme $\mathcal{G}$, we design a procedure, based on a two-player game, that produce a subset $\delta_{\mathcal{A},\mathcal{G}}$ of annotated symbols, such that for any term $t : o$, $\mathcal{A}$ accepts the value tree of t if and only if there exists a term in $CA(t)$ that uses only symbols of $\delta_{\mathcal{A},\mathcal{G}}$.

We show that from $\delta_{\mathcal{A},\mathcal{G}}$ one can construct a morphism, such that, once embedded in the scheme, one solves the automata reflection problem. Then we use a result of [BCOS10] to construct a solution for the MSO reflection problem. Finally, we show that from a positional winning strategy of the game, one can compute a scheme that solves the automata selection problem.

These problems were solved in [BCOS10] and [CS12] using the equivalence between HORS and CPDA, therefore their procedure loses the structure of the scheme, thus of the functional program from which the scheme is obtained. The procedures presented here are shape-preserving scheme transformations and do not involve CPDA.

4.2. Generalising the notion of runs to higher order terms

The goal of this section is to extend the notion of run of a parity automaton to higher order terms. We first give a new definition of parity automata equivalent to the previous one that intuitively corresponds to putting colours in the transitions instead of the states. Then we see on an example how we should enlarge the definition of transitions in order to describe the behaviour of non-terminal symbols of a scheme. Finally we formalise this definition and state a problem that slightly generalises the model checking problem.

4.2.1. Parity automata revisited

In order to perform model checking over HORS, we propose an alternative definition of parity automata. Here we associate a tuple of colours with a transition. In the following, a ***set of colours*** is a set $\{0, \dots, m\}$ for some $m \in \mathbb{N}$. We give a general form for the transitions of the such automata.

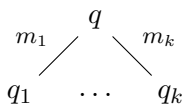
Definition 4.2.1 (Simple Transitions)**.**

Given a set of states Q and a set of colours M, for all type τ of order 0 or 1 the set $\Theta_s^\tau(Q, M)$ of ***simple transitions of type*** τ is defined by:

- $\Theta_s^o(Q, M) = Q$,
- For all $k > 0$,

$$\Theta_s^{o^k \to o}(Q, M) = \{q \to \big((q_1, m_1), \dots, (q_k, m_k)\big) \mid q, \in Q, \forall i\ q_i \in Q, m_i \in M\}.$$

When Q and M are clear from the context we simply write $\Theta_s = \Theta_s(Q, M)$. We may graphically represent the simple transition $q \to \big((q_1, m_1), \dots, (q_k, m_k)\big)$ as follows.



Given a simple transition $\theta = q \to \big((q_1, m_1), \dots, (q_k, m_k)\big)$ we define

- the ***source*** of θ as $source(\theta) = q$,
- the ***arity*** of θ as $arity(\theta) = k$,
- for all $j \leq k$, the j***-th destination*** of θ as $dst_j(\theta) = q_j$,
- for all $j \leq k$, the j***-th colour*** of θ as $colour_j(\theta) = m_j$.

In the standard definition of automata, transitions contained symbols. Therefore we annotate symbols of a ranked alphabet with transition, obtaining a new ranked alphabet. Although this definition is the same as the one we introduced in Chapter 2 about annotations, we give the full definition, as we extend it in the next section.

Definition 4.2.2 (Annotated symbols)**.**

Given a set of states Q, a set of colours M and a ranked alphabet Σ, the ranked alphabet Σ_{Θ_s} is defined as follows.

$$\Sigma_{\Theta_s} = \left\{ a^\theta : \tau \mid a : \tau \in \Sigma, \theta : \tau \in \Theta_s \right\}$$

We introduce a special kind of trees called transition trees appropriate for transitions we introduced before. A transition tree is a tree where each node is labelled by an annotated symbol. Intuitively a node labeled by $a^{q \to ((q_1,m_1),(q_2,m_2))}$ requires the sources of its sons to be q_1 and q_2 respectively, *i.e.* if θ_1 and θ_2 are the annotations of its sons, we want that $source(\theta_1) = q_1$ and $source(\theta_2) = q_2$.

Definition 4.2.3 (Transition tree)**.**

Let Σ be a ranked alphabet, Q a finite set, and M a set of colours. Given a tree R on Σ_{Θ_s}, we say that R is a ***transition tree*** from state $q_0 \in Q$ if the following holds.

- $R(\varepsilon) = a_0^{\theta_0}$ for some $a_0 \in \Sigma$ and $\theta_0 \in \Theta_s$ such that $source(\theta_0) = q_0$.
- For all node u, if $R(u) = a^\theta$ with $\theta = q \to ((q_1, m_1), \ldots, (q_k, m_k))$, then for all $i \leq k$, $R(u \cdot i) = b^{\theta_i}$ for some $b \in \Sigma$ and $\theta_i \in \Theta_s$ such that $source(\theta_i) = q_i$.

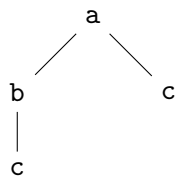
Given a transition tree R, the set $\delta_R \subseteq \Sigma_{\theta_s}$ is defined as the set of all symbols occurring in R, *i.e.* $\delta_R = \{R(u) \mid u \in dom(R)\}$.

The following definition matches the idea that a transition tree on Σ_{Θ_s} can be seen as a tree on Σ to which we have added some annotations.

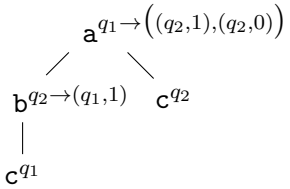
Definition 4.2.4 (Correct annotation)**.**

Let T be a tree on Σ and R be a tree on Σ_{Θ_s}. R is a ***correct annotation*** of T if R is a transition tree, $dom(R) = dom(T)$, and for all node u, if $T(u) = a \in \Sigma$, then $R(u) = a^\theta$ for some $\theta \in \Theta_s$.

Example 4.2.1. Let $Q = \{q_1, q_2\}$, $M = \{0, 1\}$, and $\Sigma = \{a : o^2 \to o, b : o \to o, c : o\}$. Finally let T be a tree on Σ, and R be a tree on Σ_{Θ_s}, both depicted below.

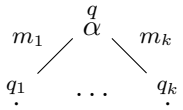


The tree T.

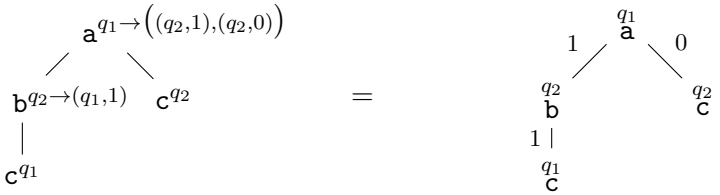


A correct annotation R of the tree T.

Then R is a correct annotation of T. We use the following graphical representation of a symbol α^θ with $\theta = q \to \big((q_1, m_1), \ldots, (q_k, m_k)\big)$.



And following this graphical representation we obtained for transition trees an alternative graphical representation described below:



As in the standard definition, one defines the colouring of a branch corresponding to the sequence of colours appearing along the branch in a transition tree.

Definition 4.2.5 (Colouring of a branch)**.**

Given a correct annotation R, and a branch $b = b_1 \cdot b_2 \cdots$ of R. The colouring of b, written $\mathit{colour}(b)$ is the sequence $m_1, m_2, \ldots$ such that for all i, if $R(b_1 \cdots b_i) = (a, \theta)$ then $m_i = \mathit{colour}_{b_i}(\theta)$.

We are now in position to introduce the new definition of a parity automaton where colours label transitions and not the states.

Definition 4.2.6 (New parity automaton)**.**

A ***parity automaton*** is a tuple $\langle \Sigma, M, Q, \delta \rangle$ where:

- Σ is a ranked alphabet,
- M is a set of colours,
- Q is a finite set of ***states***,
- δ is a subset of $\Sigma_{\Theta_s(Q,M)}$.

Accordingly to the intuition, a run of the automaton over a tree is a correct annotation of that tree whose symbols are in δ. As previously it is accepting if the maximum colour seen along every infinite branch is even.

Definition 4.2.7 (Run of an automaton)**.**

Given a new parity automaton $\mathcal{A} = \langle \Sigma, M, Q, \delta \rangle$ a tree T on Σ, and a state $q \in Q$.
A ***run*** of $\mathcal{A}$ on T from state q is a correct annotation R of T from state q such that $\delta_R \subseteq \delta$.
A run R is ***accepting*** if for all infinite branch b of R, the greatest colour seen infinitely often in $\mathit{colour}(b)$ is even.

Definition 4.2.8 (Acceptance by an automaton)**.**

Given a new parity automaton $\mathcal{A} = \langle \Sigma, M, Q, \delta \rangle$ a tree T on Σ, and a state $q \in Q$.
The tree T is ***accepted by $\mathcal{A}$ from state*** q if there exists an accepting run of $\mathcal{A}$ on T from state q.

We let the reader check that the family of languages defined by the two kinds of parity automata coincide. Furthermore there are effective polynomial time transformations from one kind of automata to the other.

In the following, we adopt the new definition of tree automata and refer to them as tree automata.

4.2.2. An example of transitions on nonterminal

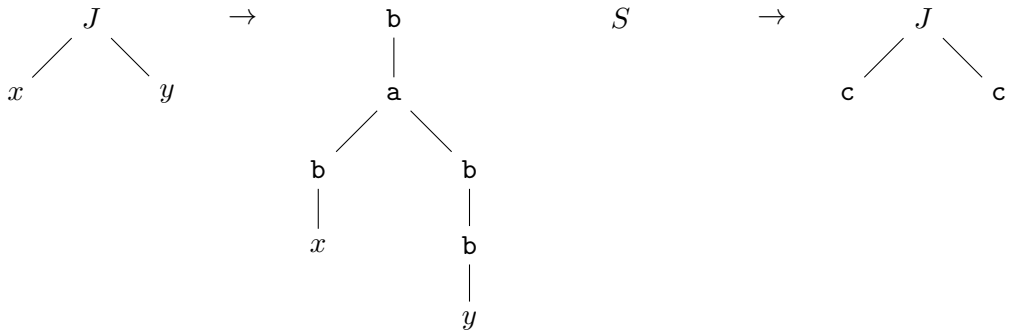
We want now to define annotations of terms by the transitions. In order to do so, we enlarge the definition of transitions. We already know how to annotate terminal symbols

by transitions. Intuitively, we try to give to a nonterminal symbol some transition corresponding to the annotations of the "unfolding" of its rewrite rule.

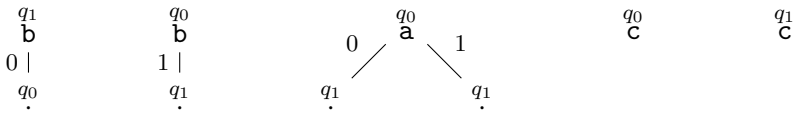
In order to explain how to associate transitions with nonterminal symbols, we illustrate the intuitions in the following example.

Example 4.2.2. Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be a scheme where:

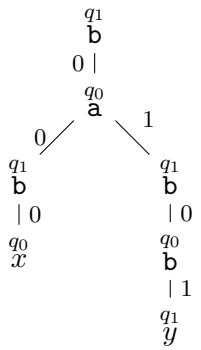
- $\Sigma = \{\mathtt{a} : o^2 \to o, \mathtt{b} : o \to o, \mathtt{c} : o\}$,
- $\mathcal{V} = \{x : o, y : o\}$,
- $\mathcal{N} = \{S : o, J : o^2 \to o\}$,
- $\mathcal{R}$ contains the following rewrite rules (the terms are depicted by their syntactical trees).



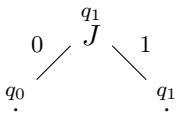
And let $\mathcal{A} = \langle \Sigma, M, Q, \delta \rangle$ be an automaton with $M = \{0, 1\}$, $Q = \{q_0, q_1\}$, and δ contains the following annotated symbols:



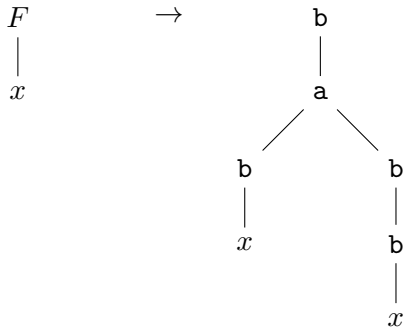
Since x and y are order-0 variables, we annotate them as constants, *i.e.* with simple states. First, observe that there is only one correct annotation of the production of the rule associated with J whose transitions of Σ are picked in δ, as depicted below.



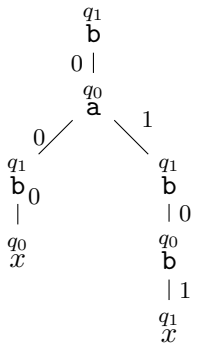
This annotation induces a transition associated with J. Indeed if we start with a labelling of J with state q_1 then we have to annotate x with state q_0 and y with state q_1. Furthermore the greatest colour seen along the way to x (resp. y) is 0 (resp. 1). This leads to the following transition.



Let us add the non-terminal symbol $F : o \to o$ with the following rewrite rule.

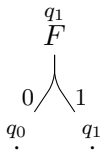


Once again there is only one correct annotation of the production of the rule associated with F whose transitions of Σ are picked in δ which is the same as previously.

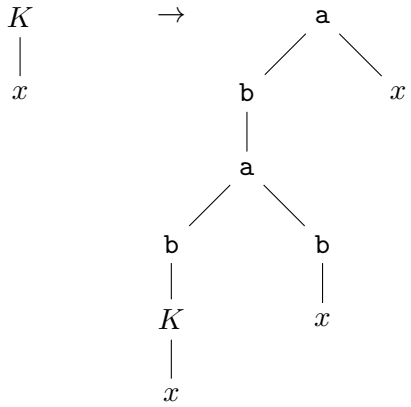


Due to the fact that x is duplicated in this rewrite rule, we cannot associate a transition with F as we did it with J. The appropriate transition should state that *"labelling F with state q_1, we need to label the first occurrence of x with state q_0 and the second with state q_1"*. The order of the occurrences is irrelevant for our purpose, since when checking the acceptance by the automaton, applying requires that the term substituted for variable x should be accepting from state q_0 and q_1.

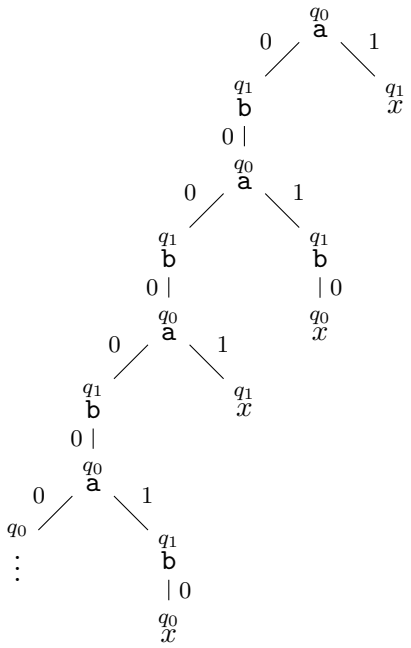
We graphically represent it as follows.



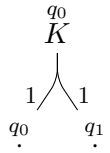
Consider the non terminal $K : o \to o$ with the following rewrite rule.



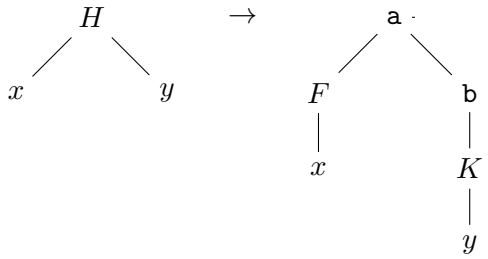
Observe that the rewriting rule includes recursion. The "unfolding" of this rewrite rule is depicted below, along with a correct annotation on this unfolding. In the previous example there was two occurrences of x therefore we needed to keep two different informations about x. Now there are infinitely many occurrences of x in the unfolding of the rule. However, x appears in only two configurations, either labelled with state q_0 or with state q_1 and in each case the maximum colour seen from the root is 1.



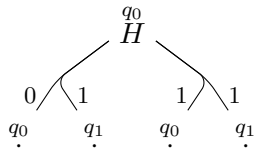
Therefore we only need to duplicate the argument in the transition associated with K.



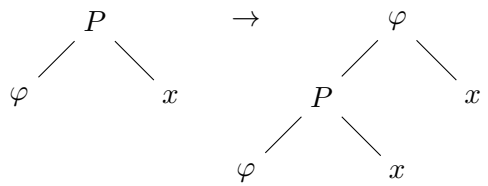
Take a nonterminal $H : o^2 \to o$ with the following rewrite rule.



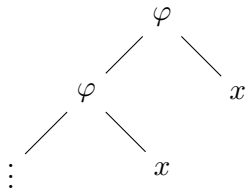
Again looking at the unfolding of this rule leads to the following transition, meaning that the first argument appears in two different configurations (state q_0 with maximum colour 0, and state q_1 with maximum colour 1) and the second argument too (state q_0 with maximum colour 1, and state q_1 with maximum colour 1).



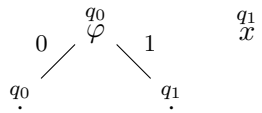
Finally, take the order-2 non-terminal symbol $P : (o^2 \to o) \to o \to o$ with the following rewrite rule (we add the variable $\varphi : o^2 \to o$).



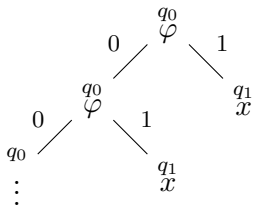
The "unfolding" of its rewrite rule is the following.



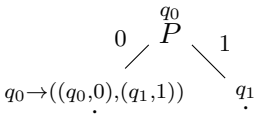
We are now looking for an annotation of this unfolding that could be obtained by one (or some) order-1 transition(s) associated with φ and one (or some) order-0 transition(s) associated with x (*i.e.* states). Looking at the figure below, if we chose transitions as depicted on the left, we get a correct annotation of the unfolding depicted on the right.



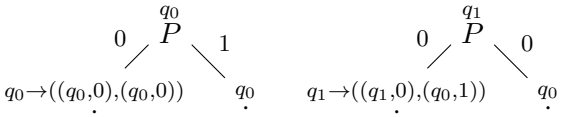
Transitions for φ and x.



To describe this possible run, we give to P an "order-2 transition": a transition whose sons are either order-0 transitions (states) or order-1 transitions. We give to P the following transition.



In general, there may exists more than one transition. The non-terminal symbol P have several possible rewrite rules, here are two examples.



If we add the non-terminal $D : o \to o$, with the rewrite rule $D\ x \to \mathtt{b}\ (\mathtt{b}\ x)$. Both following transitions are valid.

4.2.3. Runs over terms

Gathering all the observations of the previous example, we obtain the following definition of higher order transitions. Intuitively an order-0 transition is a state and a transition of type $\tau_1 \to \cdots \to \tau_k \to o$ is a pair $q \to (\sigma_1, \ldots, \sigma_k)$ of a state q and a tuple of sets, one for each argument, such that for all i, the elements of σ_i are pairs of a transition of type τ_i and a colour.

Following the ideas presented in the previous example, ideally we would like that if we associate a transition $q \to (\sigma_1, \dots, \sigma_k)$ with a non-terminal symbol $F : \tau_1 \to \cdots \to \tau_k \to o$, then there exist a run on the "unfolding" of the rewrite rule $F\ x_1 \dots x_k \to e$ associated with e, such that for all i, σ_i is the set of "configurations" of x_i in this unfolding, *i.e.* $(\theta, m) \in \sigma_i$ if there is an occurrence of x_i in the unfolding annotated with θ and such that the greatest colour seen from the root of this unfolding to the occurrence is m.[1]

Definition 4.2.9 (Higher order transitions)**.**

Given a set Q and a set of colours M, we define for all type τ the set $\Theta^\tau(Q, M)$ of ***(higher order) transitions of type*** τ:

- $\Theta^o(Q, M) = Q$,
- For all type $\tau = \tau_1 \to \cdots \to \tau_k \to o$,

$$\Theta^\tau(Q, M) = \big\{ q \to (\sigma_1, \dots, \sigma_k) \mid q \in Q, \forall i\ \sigma_i \subseteq \Theta^{\tau_i}(Q, M) \times M \big\}.$$

Given a typed alphabet Γ, we are now in position to define the alphabet Γ_Θ of symbols annotated with transitions. As previously an annotated symbol in Γ_Θ is a symbol of type τ along with a transition of type τ.

Let $\alpha : \tau$ be a symbol and θ and θ' be two transitions of type τ, we want to produce two annotated symbols α^θ and $\alpha^{\theta'}$ of the same type τ^+ to be defined. However the structure of θ and θ' may be different as shown in the following example.

Example 4.2.3. Take the symbol $F : o \to o$ and the transitions $\theta = q \to (\{(q_0, 0), (q_1, 1)\})$ and $\theta' = q_0 \to (\{(q_0, 0)\})$, how do we type the annotated symbols F^θ and $F^{\theta'}$ depicted below?

Fortunately, we know that for all transition $q \to \sigma : o \to o$, the cardinal of σ is bounded by the cardinal of $Q \times M$. Therefore we require that all annotated versions of F have arity $n = Q \times M$, each argument corresponding to one element of $Q \times M$. So $F^\theta : o^n \to o$ and $F^{\theta'} : o^n \to o$.

Let us generalise this example.

[1] Higher order transitions are exactly strict intersection type in the sense of [KO09].

Definition 4.2.10 (The set Γ_Θ)**.**

Let Q be a set of states and M be a set of colours. For all type τ we write $Pairs^\tau = \Theta^\tau(Q, M) \times M$. Let $|\tau| = Card(Pairs^\tau)$, and let $p_1^\tau, \dots, p_{|\tau|}^\tau$ be a total ordering on $Pairs^\tau$, *i.e.* $Pairs^\tau = \{p_1^\tau, \dots, p_{|\tau|}^\tau\}$.
With all type $\tau = \tau_1 \to \cdots \to \tau_k \to o$, we associate the type τ^+, inductively defined by:

$$\tau^+ = (\tau_1^+)^{|\tau_1|} \to \cdots \to (\tau_k^+)^{|\tau_k|} \to o.$$

Given a typed alphabet Γ, the typed alphabet Γ_Θ is defined by:

$$\Gamma_\Theta = \{\alpha^\theta : \tau^+ \mid \alpha : \tau \in \Gamma, \theta \in \Theta^\tau(Q, M)\}.$$

Given $\alpha^\theta \in \Gamma_\Theta$ such that $\alpha^\theta : (\tau_1^+)^{|\tau_1|} \to \cdots \to (\tau_k^+)^{|\tau_k|} \to o$. For all $i \leq k$ let $\vec{t_i}$ be a sequence of $|\tau_i|$ terms of type τ_i^+, *i.e.* $\vec{t_i} = (t_i^1, \dots, t_i^{|\tau_i|})$ such that for all $j \leq |\tau_i|$, $t_i^j : \tau_i^+$. We use the notation $\alpha^\theta\ \vec{t_1} \dots \vec{t_j}$ as a shorthand to denote the term

$$\alpha^\theta\ t_1^1 \dots t_1^{|\tau_1|}\ \dots\ t_k^1 \dots t_k^{|\tau_k|}.$$

In the following, the notation $\vec{s} : \tau^+$ will always refer to a sequence of $|\tau|$ terms of type τ^+ and we write s^i the i-th element of this sequence. Furthermore, if $p_i^\tau = (\theta, m)$, we may write $s^{\theta,m} = s^i$.

We are now in position to define, as in the previous section, a correct annotation of a term. We require again that, if a symbol is annotated with $q \to (\sigma_1, \dots, \sigma_k)$ then for all (θ, m) in σ_i for some i, it has a son in its i-th tuple of arguments labelled with θ.

For every type τ, we define the fresh symbol $\Box^\tau : \tau$, and we let *Boxes* be the set of all these symbols. Intuitively $\Box^\tau$ will be used to fill the missing arguments in an term as described in the following example.

Example 4.2.3 continued. Assume that the arity of the annotated versions of F is 2. Therefore if we want to annotate the term $F\ \mathtt{a}$ with $\mathtt{a} : o$, with the transition $\theta = q \to (\{(q_0, 0), (q_1, 1)\})$ on F, F^θ will have two arguments, $\mathtt{a}^{q_0}$ and $\mathtt{a}^{q_1}$:

$$F^{q \to (\{(q_0,0),(q_1,1)\})}\ \mathtt{a}^{q_0}\ \mathtt{a}^{q_1}.$$

However if F is annotated with $\theta' = q \to (\{(q_0, 0)\})$, also $F^{\theta'}$ expects two arguments, the transition θ' expects only one configuration for the argument, therefore we use $\Box^o$ to replace the missing arguments.

$$F^{q \to (\{(q_0,0)\})}\ \mathtt{a}^{q_0}\ \Box^o.$$

Definition 4.2.11 (Correct annotation)**.**

Let $0 \leq k' \leq k$ be two integers. Let $t = \alpha\ t_1 \dots t_{k'} : \tau \in \mathcal{T}(\Gamma)$ with

$$\alpha : \tau_1 \to \dots \to \tau_k \to o \quad \text{and} \quad \tau = \tau_{k'+1} \to \dots \to \tau_k \to o.$$

Let $\theta = q \to (\sigma_{k'+1}, \dots, \sigma_k) \in \Theta^\tau(Q, M)$. The set of ***θ-annotations of*** t is inductively defined on the structure of t as follows. Given $\sigma_1 \subset \Theta^{\tau_1}, \dots, \sigma_{k'} \subset \Theta^{\tau_{k'}}$, the term

$$t' = \alpha^{q \to (\sigma_1, \dots, \sigma_{k'}, \sigma_{k'+1}, \dots, \sigma_k)}\ \vec{t}_1 \dots \vec{t}_{k'}$$

is a θ-annotation of t if for all $t_i^{\theta', m'}$:

- if $(\theta', m') \in \sigma_i$ then $t_i^{\theta', m'}$ is a θ'-annotation of t_i,
- if $(\theta', m') \notin \sigma_i$ then $t_i^{\theta', m'} = \Box^{\tau_i^+}$.

In the following, we say that s is a ***correct annotation*** of t if it is a θ-annotation of t for some θ, furthermore we may not precise t, *i.e.* we simply say that that s is a correct annotation, if it is a correct annotation of some term t and we denote by $CA_{Q,M}(\Gamma)$ (or simply $CA(\Gamma)$ when Q and M are clear from the context) the set of all correct annotations of some terms in $\mathcal{T}(\Gamma)$. Finally we say that a context $C[\bullet]$ is a ***correct context*** if there exist two correct annotations s and s' such that $C[s'] = s$ (basically, it is a correct annotation where one subterm is replaced by $\bullet$). We denote $CC(\Gamma)$ the set of correct contexts.

Since we have, using the maximal possible set of pairs, unified transitions with different cardinalities of pairs, the previous definition recovers the intuition that the pairs absent in some subset do not have to be checked and thus the corresponding subterm is a box.

Given an annotation t' of a term t, we write $\delta_{t'} \subseteq \Gamma_\Theta$ the set of symbols, different from $\Box^\tau$, occurring in t'. Given Γ' a subset of Γ and $\delta \subseteq \Gamma'_\Theta$, t' ***conforms to*** δ ***on*** Γ' if $\delta_{t'} \cap \Gamma'_\Theta \subseteq \delta$.

Furthermore, for all simple transition $\theta = q \to ((q_0, m_1), \dots, (q_k, m_k))$, we let $\tilde{\theta} = q \to (\{(q_0, m_1)\}, \dots, \{(q_k, m_k)\}) \in \Theta$, and, given a ranked alphabet Σ, for all $\delta \subseteq \Sigma_{\Theta_s}$ we let $\tilde{\delta} = \{a^{\tilde{\theta}} \mid a^\theta \in \delta\}$.

We define a run of an automaton over a term as follows.

Definition 4.2.12 (Run over terms)**.**

Given a ranked alphabet Σ and a typed alphabet Γ, a term $t = \mathcal{T}(\Sigma \uplus \Gamma)$ and an automaton $\mathcal{A} = \langle \Sigma, M, Q, \delta \rangle$, a ***θ-run of*** $\mathcal{A}$ ***over*** t is a θ-annotation r of t that conforms to $\tilde{\delta}$ on Σ.

A run of $\mathcal{A}$ over some term is simply a q-run for some q. We denote by $\mathcal{A}(\Gamma)$ the set of runs of $\mathcal{A}$ over some terms in $\mathcal{T}(\Gamma)$.

Ideally, we would like to find a definition of *accepting run* over a term that satisfies a property of the kind of "given a scheme, an automaton and a term t on the scheme, there exists an accepting q-run of the automaton on t if the automaton accepts the value tree of t from state q".

We approach this problem from another angle. We look at some conditions on terms, and wish to find a condition $\mathcal{C}$ such that a term $t : o$ satisfies $\mathcal{C}$ if and only its value tree is accepted by the automaton from some states. Then we will try to construct such a condition. The conditions are described as sets of annotated non terminals and require terms to conforms to these sets on $\mathcal{N}$.

Formally, here is the problem we want to solve.

Problem 8.
Input: *A scheme* $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$*, an automaton* $\mathcal{A} = \langle \Sigma, M, Q, \delta \rangle$*.*
Output: *A set* $\delta_{\mathcal{N}} \subseteq \mathcal{N}_\Theta$ *such that:*
for all term $t : o \in \mathcal{T}(\Sigma \uplus \mathcal{N})$*, there exists a* q*-run* r *of* $\mathcal{A}$ *over* t *that conforms to* $\delta_{\mathcal{N}}$ *on* $\mathcal{N}$ *if and only if* $\mathcal{A}$ *accepts* $\|t\|_{\mathcal{G}}$ *from state* q*.*

As we will see in Section 4.4, this problem is solved by an algorithm introduced by Kobayashi and Ong [KO09] to solve the automata model checking problem.

4.3. Reflection

Before solving this problem, we see how, once it is solved, one can construct an algorithm for the automata reflection problem, and then for the MSO reflection problem. Therefore in this section, we assume that we have an algorithm that solves Problem 8.

As we will see, the MSO reflection problem is decomposed into the automata reflection and the so-called *regular path reflection* problem. First we solve the automata reflection.

4.3.1. Automata reflection

The goal of this section is to find an algorithm that solves the automata reflection problem. Recall that given a tree T on a ranked alphabet Σ, and an automaton $\mathcal{A}$ on Σ the $\mathcal{A}$-reflection on T is the tree $T_{\mathcal{A}}$ on the ranked alphabet $\Sigma \times \mathcal{P}(Q)$ (Q stands for the set of states of $\mathcal{A}$), such that $dom(T_{\mathcal{A}}) = dom(T)$ and for all $u \in dom(T_{\mathcal{A}})$, $T_{\mathcal{A}}(u) = (T(u), \mathcal{A}(T(u)))$ where $\mathcal{A}(T[u])$ denotes the set of states q such that $\mathcal{A}$ accepts the subtree of T rooted on u from q.

The automata reflection problem asks given a scheme $\mathcal{G}$ and an automaton $\mathcal{A}$ to produce another scheme $\mathcal{G}'$ whose value tree is the reflection of the automaton on the value tree of the initial scheme, *i.e.* $\|\mathcal{G}'\| = \|\mathcal{G}\|_{\mathcal{A}}$.

In the following, we fix a scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ and an automaton $\mathcal{A} = \langle \Sigma, M, Q, \delta \rangle$, and we assume without loss of generality that for any ground typed term $t : o$ of $\mathcal{G}$, $\|t\|_{\mathcal{G}}$ does not contain any occurrence of $\perp$; indeed one can always obtain this

restriction by applying the $\bot$-elimination[2] algorithm on $\mathcal{G}$. In the sequel we make this assumption for every scheme we introduce.

We assume that one has computed a set $\delta_{\mathcal{N}} \subseteq \mathcal{N}_\Theta$ such that for all term $t : o$, there exists a q-run r of $\mathcal{A}$ over t that conforms to $\delta_{\mathcal{N}}$ on $\mathcal{N}$ if and only if $\mathcal{A}$ accepts $\|t\|_{\mathcal{G}}$ from state q.

We define the typed application system $\langle \mathbb{S}, \cdot \rangle$ as follows.

- For all type τ, $\mathbb{S}^\tau = \mathcal{P}(\Theta^\tau)$,
- Given $S : \tau' \to \tau$ and $S' : \tau'$,

$$S \cdot S' = \{q \to (\sigma_1, \ldots, \sigma_k) \mid \exists q \to (\sigma_0, \sigma_1, \ldots, \sigma_k) \in S \ \forall (\theta, m) \in \sigma_0 \ \theta \in S'\}.$$

We define the morphism $\varphi_{\mathcal{A}} \in \mathbf{Morph}(\mathcal{G}, \mathbb{S})$ as for all $a \in \Sigma$, $\varphi_{\mathcal{A}}(a) = \{\theta \mid a^\theta \in \tilde{\delta}\}$, for all $F \in \mathcal{N}$, $\varphi_{\mathcal{A}}(F) = \{\theta \mid F^\theta \in \delta_{\mathcal{N}}\}$. We want this morphism to describe the set of transitions from which one can annotate a given term, as stated in the following proposition.

Proposition 4.3.1.

For all term $t : \tau$ and transition $\theta : \tau$, there exists a θ-run t' of t that conforms to $\delta_{\mathcal{N}}$ on $\mathcal{N}$ if and only if $\theta \in \varphi_{\mathcal{A}}(t)$.

Proof. We prove this result by induction of the structure of t. Let $\theta = q \to (\sigma_1, \ldots, \sigma_k)$.

- Let $t = a \in \Sigma$. If there exists a correct annotation t' from θ that conforms to $\tilde{\delta} \cup \delta_{\mathcal{N}}$, $t' = a^\theta$ with $a^\theta \in \tilde{\delta}$ therefore $\theta \in \varphi_{\mathcal{A}}(t)$. On the other hand if $\theta \in \varphi_{\mathcal{A}}(t)$ then $a^\theta \in \tilde{\delta}$, therefore it is a correct annotation of t from θ that conforms to $\tilde{\delta} \cup \delta_{\mathcal{N}}$.
- Let $t = F \in \mathcal{N}$. If there exists a correct annotation t' from θ that conforms to $\tilde{\delta} \cup \delta_{\mathcal{N}}$, then $t' = F^\theta$ with $F^\theta \in \delta_{\mathcal{N}}$ therefore $\theta \in \varphi_{\mathcal{A}}(t)$. On the other hand if $\theta \in \varphi_{\mathcal{A}}(t)$ then $F^\theta \in \delta_{\mathcal{N}}$, therefore it is a correct annotation of t from θ that conforms to $\tilde{\delta} \cup \delta_{\mathcal{N}}$.
- Let $t = t_1\ t_2$ with $t_1 : \tau' \to \tau$ and $t_2 : \tau$. If there exists a correct annotation t' from θ that conforms to $\tilde{\delta} \cup \delta_{\mathcal{N}}$, then $t' = t'_1\ \vec{t}'_2$, with t'_1 a $q \to (\sigma, \sigma_1, \ldots, \sigma_k)$-annotation of t_1 and for all $(\theta', m') \in \sigma$ $t'^{\theta', m'}_2$ is a correct annotation of t_2 from θ'.

  Therefore, by induction hypothesis $q \to (\sigma, \sigma_1, \ldots, \sigma_k) \in \varphi_{\mathcal{A}}(t_1)$ and for all $(\theta', m') \in \sigma$, $\theta' \in \varphi_{\mathcal{A}}(t_2)$. Therefore $q \to (\sigma_1, \ldots, \sigma_k) \in \varphi_{\mathcal{A}}(t)$.

  On the other hand assume that $q \to (\sigma_1, \cdots, \sigma_k) \in \varphi_{\mathcal{A}}(t)$. Then there exists σ such that $\theta' = q \to (\sigma, \sigma_1, \cdots, \sigma_k) \in \varphi_{\mathcal{A}}(t_1)$ and for all $(\theta'', m) \in \sigma$, $\theta'' \in \varphi_{\mathcal{A}}(t_2)$. Therefore by induction hypothesis:

[2] as we will see, this does not increase the theoretical complexity of the algorithms we present.

- there exists a correct annotation t'_1 of t_1 from $q \to (\sigma, \sigma_1, \ldots, \sigma_k)$,
- for all θ'' such that $(\theta'', m) \in \sigma$ for some m, there exists a correct annotation $t_2^{\theta''}$ of t_2 from θ''.

Then $t'_1\ \vec{t'}_2$ is a correct annotation of t from θ, where $\vec{t'_2} : \tau'^+$ is define by:

- for all $(\theta'', m) \in \sigma$, $t'^{\theta'',m}_2 = t_2^{\theta''}$,
- for all $(\theta'', m) \notin \sigma$, $t'^{\theta'',m}_2 = \Box^{\tau'^+}$.

□

Therefore, for all $t : o$, $\varphi_{\mathcal{A}}(t)$ is the set of states q such that $\mathcal{A}$ accepts $\|t\|$ from q. Thus, $\varphi_{\mathcal{A}}$ describes a tree property and it can be embedded into $\mathcal{G}$. As a consequence, we obtain an algorithm that solves the automata reflection problem.

Theorem 4.3.2 (Automata Reflection)**.**

Given a scheme $\mathcal{G}$ and an automaton $\mathcal{A}$ one can construct a scheme $\mathcal{G}'$ whose value tree is the $\mathcal{A}$-reflection of $\|\mathcal{G}\|$.

Proof. The algorithm is the following.

Input: An order-n scheme $\mathcal{G}$, a parity automaton $\mathcal{A}$.

1. Solve Problem 8 and computes $\delta_{\mathcal{N}}$.
2. Construct the morphism $\varphi_{\mathcal{A}}$.
3. Construct the embedding $\mathcal{G}'$ of $\varphi_{\mathcal{A}}$ in $\mathcal{G}$.
4. Transform $\mathcal{G}'$ into $\mathcal{G}''$ where the symbols $a^{h_1,\ldots,h_k}$ are replaced by $(a, \varphi_{\mathcal{A}}(a)\ h_1 \ldots h_k)$ for all $a \in \Sigma$.

Output: The scheme $\mathcal{G}''$ such that $\|\mathcal{G}''\|$ is the reflection of $\mathcal{A}$ in $\|\mathcal{G}\|$.

□

4.3.2. MSO reflection

Now, we extend this result by showing that one can construct an algorithm to solve the MSO reflection problem. Recall that given a tree T on a ranked alphabet Σ and an MSO formula $\varphi[x]$, the $\varphi[x]$-reflection of T, denoted $T_{\varphi[x]}$ is the tree on the ranked alphabet $\Sigma \times \{0, 1\}$ such that $dom(T_{\varphi[x]}) = dom(T)$ and for all $u \in dom(T)$, $T_{\varphi[x]}(u) = (T(u), b)$ such that $b = 1$ if and only if $T \models_{\{x \mapsto u\}} \varphi[x]$.

The MSO reflection problem asks given a scheme $\mathcal{G}$ and an MSO formula $\varphi[x]$, to produce another scheme $\mathcal{G}'$ whose value tree is the reflection of the formula on the value tree of the initial scheme, *i.e.* $\|\mathcal{G}'\| = \|\mathcal{G}\|_{\varphi[x]}$.

The MSO reflection algorithm uses two ingredients, the first one is the automata reflection, the second one is the *regular path language reflection*. A regular path language is a regular language of words whose letters alternates between a ranked alphabet and arities of this alphabet. Given a tree, a regular path language describes a set of nodes in the tree.

Definition 4.3.1.

Let Σ be a ranked alphabet, and let $arity_{\max}$ be the maximum arity of symbols of Σ. A ***regular path language (RPL)*** L over Σ, is a regular languages over words of $(\Sigma \cdot \{1, \dots, arity_{\max}\})^* \cdot \Sigma$.
Given a tree T of Σ, and a node $u = j_0 \cdots j_k$ in T, we say that u is in the RPL L if

$$T(\varepsilon) \cdot j_0 \cdot T(j_0) \cdot j_1 \cdot T(j_0 \cdot j_1) \cdots j_k \cdot T(u) \in L.$$

In the following, we denote $Path(u)$ the word $T(\varepsilon) \cdot j_0 \cdot T(j_0) \cdot j_1 \cdot T(j_0 \cdot j_1) \cdots j_k \cdot T(u)$. Given a RPL L and a tree T, the *L-reflection* on T is the tree obtained from T by marking the nodes that are in L.

Definition 4.3.2 (L-Reflection)**.**

Given a tree T of a ranked alphabet Σ, and a RPL L over Σ, the ***L-reflection*** of T is the S-marking T_L of T, with S being the set of nodes of T in L.

Finally the RPL reflection problem asks given a scheme and an RPL to produce a scheme whose value tree is the reflection of the RPL in the value tree of the initial scheme.

Problem 9 (RPL reflection problem).
Input: *A scheme $\mathcal{G}$, a finite word automaton $\mathcal{B}$ whose language $L_{\mathcal{B}}$ is a RPL.*
Output: *A scheme $\mathcal{G}'$ such that $\|\mathcal{G}\|$ is the $L_{\mathcal{B}}$-reflection of $\|\mathcal{G}\|$.*

Once solved the RPL reflection problem, one can use the following theorem, from [BCOS10], that reduce the MSO reflection into automata and RPL reflection.

Theorem 4.3.3 (Broadbent, Carayol, Ong, Serre)**.**

There exists an algorithm with the following specification:
Input: An MSO formula $\varphi[x]$ on the ranked alphabet Σ,
Output: A parity tree automaton $\mathcal{A} = \langle \Sigma, M, Q, \delta \rangle$ and a finite word automaton $\mathcal{B}$ on the ranked alphabet $\Sigma_{\Theta_s(Q,M)} \cup \{1, \dots, arity_{\max}\}$ whose language L is a RPL, such that the following holds.

For all tree T, let $T_{\mathcal{A}}$ be the $\mathcal{A}$-reflection of T and $(T_{\mathcal{A}})_L$ be the L-reflection of $T_{\mathcal{A}}$, then $(T_{\mathcal{A}})_L$ is the $\varphi[x]$-reflection of T.

First design an algorithm to solve the RPL reflection reflection problem. Take a deterministic and complete[3] finite word automaton $\mathcal{B} = \langle \Sigma \uplus \{1, \dots, arity_{\max}\}, Q, q_0, F \subseteq Q, \delta_{\mathcal{B}} \rangle$ whose RPL language is denoted by L.

The algorithm follows the following steps. First, we construct a deterministic and complete TAC automaton (*i.e.* we have only one colour equal to 0) $\mathcal{A} = \langle \Sigma, Q, \delta \rangle$ such that $Path(u) \in L$ if and only if the node u in the run of $\mathcal{A}$ over T from q_0 is labelled by some state in F (there exists a unique run because the automaton is deterministic and complete). Then we present a scheme transformation that transforms a scheme into a new one whose value tree is the run of $\mathcal{A}$ starting from q_0 on the value tree of the initial scheme. Finally we slightly transform this scheme such that the nodes of its value tree are simply marked by 0 or 1 whether the source of its transition in the previous tree is in F or not.

For all $a \in \Sigma$ and $q \in Q$ we defined the unique transition θ such that a^θ in δ and $source(\theta) = q$ as follows. Let $q' \in Q$ be the state such that $q \xrightarrow{a} q' \in \delta_{\mathcal{B}}$ and for all $i \leq arity(a)$ let q_i be the state such that $q \xrightarrow{i} q' \in \delta_{\mathcal{B}}$,then $\theta = q \to (q_1, \dots, q_k)$.

Let T be a tree and R the run of $\mathcal{A}$ on T from q_0. For all node u, a simple induction on the structure of u shows that shows that if $R(u) = a^\theta$, then $source(\theta)$ is the state reached by $\mathcal{B}$ after reading $Path(u)$.

Now we show that given a scheme, we can embed in the scheme the reached states of the run of $\mathcal{A}$ over the value tree of the scheme. We fix a scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$. We create a scheme $\mathcal{G}'$ on Σ_{Θ_s} such that $\|\mathcal{G}'\|$ is the run of $\mathcal{A}$ over $\|\mathcal{G}\|$ from state q_0. This transformation resemble the embedding transformation of Chapter 3.

Types. In the following let $N = Card(Q)$, and let $q_1, \dots, q_N$ be an enumeration of Q. For all type $\tau = \tau_1 \to \cdots \to \tau_k \to o$, we inductively define the type

$$[\tau] = [\tau_1]^N \to \cdots \to [\tau_k]^N \to o.$$

Symbols. With a terminal $a : o^k \to o$ and a states q, we associate the non-terminal symbol $\overline{a}^q : [o^k \to o] \in \mathcal{N}'$.

[3] *i.e.* for every letter a and state q there exists exactly one transition from q on a.

With a non-terminal symbol $F : \tau$ (*resp.* a variable $x : \tau$) and a states q, we associate the non-terminal symbol $F^q : [\tau] \in \mathcal{N}'$ (*resp.* the variable $x^q : [\tau] \in \mathcal{V}'$).

Furthermore, let $arity_{\max}$ be the maximum arity of Σ, for all $i \leq arity_{\max}$ and for all state q, we define the variable $\eta_i^q \in \mathcal{V}'$.

Term transformation. Given a term $t : \tau \in \mathcal{T}(\mathcal{V}' \uplus \Sigma' \uplus \mathcal{N}')$ and a state q , we inductively define the term $t^q : [\tau] \in \mathcal{T}(\mathcal{V}' \uplus \mathcal{N}')$.

- If $t = a \in \Sigma$, then $t^q = \overline{a}^q$.
- If $t = \alpha \in \mathcal{N} \uplus \mathcal{V}$, then $t^q = \alpha^q$.
- If $t = t_1\ t_2$ we define
$$(t_1\ t_2)^q = {t_1}^q\ {t_2}^{q_1} \dots\ {t_2}^{q_N}.$$

Note that since this transformation is only duplicating and annotating, given a term t^q we can find the unique term t associated to it.

Rewrite rules. Let $F : \tau_1 \to \cdots \to \tau_k \to o \in \mathcal{N}$, $q \in Q$. If $F\ x_1 \dots x_k\ \to\ e \in \mathcal{R}$, we define in $\mathcal{R}'$ the rule

$$F^q\ x_1^{q_1} \dots\ x_1^{q_N}\ \dots\ x_k^{q_1} \dots\ x_k^{q_N}\ \to\ e^q.$$

Let $a : o^k \to o \in \Sigma$, $q \in Q$ and let $(q_{i_1}, \dots, q_{i_k})$ be the unique tuple of states such that $a^{q \to (q_{i_1}, \dots, q_{i_k})} \in \delta$. We define in $\mathcal{R}'$ the rule

$$\overline{a}^q\ \eta_1^{q_1} \dots\ \eta_1^{q_N}\ \dots\ \eta_k^{q_1} \dots\ \eta_k^{q_N}\ \to\ a^{q \to (q_{i_1}, \dots, q_{i_k})}\ \eta_1^{q_{i_1}} \dots \eta_k^{q_{i_k}}.$$

Finally, We let $\mathcal{G}' = \langle \mathcal{V}', \Sigma_{\Theta_s}, \mathcal{N}', \mathcal{R}', S^{q_0} \rangle$.

Lemma 4.3.4.

Given two terms $t, t' : o \in \mathcal{T}(\Sigma \uplus \mathcal{N})$, if $t \Rightarrow_{\mathcal{G}} t'$, then $t^q \Rightarrow_{\mathcal{G}'} t'^q$.

Proof. Straightforward induction on t. □

Since we assume that for all term $t : o$, $\|t\|$ does not contain $\bot$, modifying the transitions associated with $\bot$ in the automaton does not affect the results, therefore we assume that $\bot^q \in \delta$ for all $q \in Q$.

Lemma 4.3.5.

Given $t : o \in \mathcal{T}(\Sigma \uplus \mathcal{N})$ and $q \in Q$, then there exists t' in $\mathcal{G}'$ such that

(1) $t^q \to^* t'$,

(2) $dom((t')^\perp) = dom(t^\perp)$,

(3) for all $u \in dom(t^\perp)$ such that $t^\perp(u) \in \Sigma$, if we let R be the run of $\mathcal{A}$ on $t^\perp$ from q, $(t')^\perp(u) = R(u)$.

(4) for all $u \in dom(t^\perp)$ such that $t^\perp(u) = \perp$, $t'^\perp(u) = \perp$.

Proof. We prove this result by induction on the structure of t. If t is a redex $t = F\ t_1 \dots t_k$, then $t^\perp = \perp$ and $(t^q)^\perp = \perp$. If $t = a\ t_1 \dots t_k$ with $a \in \Sigma$ and for all i, $t_i : o$. Let $(q_{i_1}, \dots, q_{i_k})$ be the unique tuple of states such that $a^{q \to (q_{i_1}, \dots, q_{i_k})} \in \delta$. Then

$$t^q = \overline{a}^q\ t_1^{q_1} \dots\ t_1^{q_N}\ \dots\ t_k^{q_1} \dots\ t_k^{q_N}\ \to\ a^{q \to (q_{i_1}, \dots, q_{i_k})}\ t_1^{q_{i_1}} \dots t_k^{q_{i_k}}.$$

By induction hypothesis, for all $j \leq k$, there exists t'_i such that:

(1) $t_j^{q_j} \to^* t'_j$,

(2) $dom((t'_j)^\perp) = dom(t_j^\perp)$,

(3) for all $u \in dom(t_j^\perp)$ such that $t^\perp(u) \in \Sigma$, for all $u \in dom(t^\perp)$ such that $t^\perp(u) \in \Sigma$, if we let R_j be the run of $\mathcal{A}$ on $t_j^\perp$ from q_{i_j}, $(t'_j)^\perp(u) = R_j(u)$,

(4) for all $u \in dom(t_j^\perp)$ such that $t_j^\perp(u) = \perp$, $(t'_j)^\perp(u) = \perp$.

Therefore, if we let $t' = a^{q \to (q_{i_1}, \dots, q_{i_k})}\ t'_1 \dots t'_k$, t' satisfies the requirement. Indeed $t^q \to^* t'$, $dom(t) = \{\varepsilon\} \cup \bigcup_i dom(t_i) = \{\varepsilon\} \cup \bigcup_j dom(t'_j) = dom(t')$, and to have (3) and (4) it suffices to remark that if R is the run of $\mathcal{A}$ on $t^\perp$, $R = a\ R_1 \dots R_k$. □

Lemma 4.3.6.

For all trees $T \sqsubseteq T'$, for all state q, if R and R' denotes respectively the runs of $\mathcal{A}$ on T and T' from q, for all $u \in dom(T)$, if $T(u) \in \Sigma$, then $R(u) = R'(u)$.

Proof. Straightforward induction on the size of u. □

Corollary 4.3.7.

$\|\mathcal{G}'\|$ is equal to the run of $\mathcal{A}$ on $\|\mathcal{G}\|$ from state q_0.

Proof. Let $t_0 = S$ and $t_0 \Rightarrow t_1 \Rightarrow t_2 \Rightarrow \cdots$ be the parallel derivation from S in $\mathcal{G}$. Then For all t_i, Lemma **4.3.4** states that $S^{q_0} \Rightarrow^* t_i^{q_0}$. Furthermore, Lemma **4.3.5** states that there exists t'_i that satisfies:

(1) $t_i^{q_0} \rightarrow^* t'_i$,

(2) $dom((t'_i)^\perp) = dom(t_i^\perp)$,

(3) for all $u \in dom(t_i^\perp)$ such that $t^\perp(u) \in \Sigma$, for all $u \in dom(t^\perp)$ such that $t^\perp(u) \in \Sigma$, if we let R_i be the run of $\mathcal{A}$ on $t_i^\perp$ from q_0, $(t'_i)^\perp(u) = R_i(u)$,

(4) for all $u \in dom(t_i^\perp)$ such that $t_i^\perp(u) = \perp$, $(t'_i)^\perp(u) = \perp$.

For all i, let $T_i = (t'_i)^\perp$. We show that the sequence of the T_i is monotonous, then we show that its limit is the the of $\mathcal{A}$ over $\|\mathcal{G}\|$.

For all i, we have $dom(T_i) = dom(t_i^\perp) \subseteq dom(t_{i+1}^\perp) = dom(T_{i+1})$. Furthermore since $t_i^\perp \sqsubseteq t_{i+1}^\perp$ we have, using Lemma **4.3.6**, that for all $u \in dom(t_i^\perp)$, if $t_i^\perp(u) \neq \perp$, then $R_i(u) = R'_i(u)$. Therefore, for all $u \in dom(T_i)$, if $T_i(u) \neq \perp$, then $T_i(u) = R_i(u) = R'_i(u) = T_{i+1}(u)$. Therefore $T_i \sqsubseteq T'_i$.

Let R be the run of $\mathcal{A}$ over $\|\mathcal{G}\|$ from q_0. We have $dom(R) = dom(\|\mathcal{G}\|) = \bigcup_i dom(t_i^\perp) = \bigcup_i dom(T_i)$. Furthermore, for all $u \in dom(\|\mathcal{G}\|)$, $\|\mathcal{G}\| = a \in \Sigma$. Therefore there exists i such that for all $j \geq i$, $u \in dom(t_j^\perp)$ and $t_j^\perp = a$, therefore $T_j(u) = R_j(u) = R(u)$. Therefore R is the limit of the sequence $T_0 \sqsubseteq T_1 \sqsubseteq \cdots$.

Since for all i, $S^{q_0} \rightarrow^* t'_i$, $T_i \sqsubseteq \|\mathcal{G}'\|$. Therefore, $R \sqsubseteq \|\mathcal{G}\|$. Sicne R does not contain any occurrence of $\perp$, $R = \|\mathcal{G}\|$. □

Recall that for all node u, if $R(u) = \big(\|\mathcal{G}\|(u)\big)^{q \rightarrow (q_1,\ldots,q_k)}$ then $q \in F$ if and only if $Path(u) \in L_{\mathcal{B}}$. Let $\mathcal{G}''$ be the scheme on the ranked alphabet $\Sigma \times \{0,1\}$ obtained from $\mathcal{G}'$ by replacing any terminal symbol $a^{q \rightarrow (q_1,\ldots,q_k)}$ by a^1 if $q \in F$ and by a^0 if $q \notin F$. Then $\|\mathcal{G}''\|$ is the L-reflection of $\|\mathcal{G}\|$.

We summarise below the algorithm to solve the RPL reflection problem.

Input: A scheme $\mathcal{G}$, a finite word automaton $\mathcal{B}$ recognising the RPL L.

1. Transforms the finite word automaton $\mathcal{B}$ into an infinite tree automaton $\mathcal{A}$ that simulates $\mathcal{B}$ on every branch.
2. Create a scheme $\mathcal{G}'$ such that $\|\mathcal{G}'\|$ is the run of $\mathcal{A}$ over $\|\mathcal{G}\|$ from q_0.
3. Transforms $\mathcal{G}'$ into $\mathcal{G}''$ by changing any non terminal symbol a^θ by a^0 or a^1.

Output: The scheme $\mathcal{G}''$ whose value tree is the L-reflection of $\|\mathcal{G}\|$.

Once we have an algorithm for the automata reflection and for the RPL reflection, we can use Theorem **4.3.3** to construct an algorithm for the MSO reflection problem, described below.

Input: A scheme $\mathcal{G}$, an MSO formula $\varphi[x]$.
1. Apply the algorithm of Theorem **4.3.3** on $\varphi[x]$ to obtain the tree automation $\mathcal{A}$ and the word automaton $\mathcal{B}$.
2. Apply the automata reflection algorithm on $\mathcal{G}$ and $\mathcal{A}$ to obtain the scheme $\mathcal{G}'$.
3. Apply the RPL reflection algorithm on $\mathcal{G}'$ and $\mathcal{B}$ to obtain the scheme $\mathcal{G}''$.

Output: The scheme $\mathcal{G}''$ whose value tree is the $\varphi[x]$-reflection of $\|\mathcal{G}\|$.

4.4. Kobayashi-Ong Result

Recall Problem 8:
Input: A scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$, an automaton $\mathcal{A} = \langle \Sigma, M, Q, \delta \rangle$.
Output: A set $\delta_{\mathcal{N}} \subseteq \mathcal{N}_\Theta$ such that:

for all term $t : o$, there exists a q-run r of $\mathcal{A}$ over t that conforms to $\delta_{\mathcal{N}}$ on $\mathcal{N}$ if and only if $\mathcal{A}$ accepts $\|t\|_{\mathcal{G}}$ from state q.

In this section, we present Kobayashi and Ong algorithm to solve the automata model checking, and then we show how, from that, one can solve Probem 8. The result consists in reducing the model checking problem into deciding who wins in a two player parity game.

First we formalise the notion of occurrence in a term that we use in this section and in the next one. Then we give the definition of a parity game. Finally given an automaton and a scheme, we define the *Kobayashi-Ong game* associated with the model checking problem, and we show that finding the winning region of this game solves Problem 8.

4.4.1. Occurrences

In the following we put some formalism on the notion of occurrence of some symbols in terms. We define a conservation relation and a creation relation that matches the ideas that after a rewriting $t_1 \to t_2$ such occurrence in t_2 is the same as such other occurrence in t_1, or that such occurrence in t_2 has been created during the rewriting $t_1 \to t_2$.

For example let us look at a term $t_1 = \mathtt{a}\ (F\ H)$ with the rewrite rule $F\ x \to J\ x\ x$. The term t_1 can be rewritten in $t_2 = \mathtt{a}\ (J\ H\ H)$. We want to formalise the fact that both occurrences of H in t_2 comes from the one in t_1 or the fact that the head $\mathtt{a}$ of t_2 is the same as the one in t_1. We also want to express the fact that the occurrence of J in t_2 has been created by the occurrence of F in t_1.

An occurrence of a symbol in a term t is a symbol along with a context such that the term t is equal to the context with the given symbol in place of $\bullet$.

Definition 4.4.1 (Occurrences)**.**

Given a set of typed symbols Γ and a term $t : \tau \in \mathcal{T}(\Gamma)$, an ***occurrence*** in t is a couple $(C[\bullet], \alpha)$ with $C[\bullet : \tau'] : \tau$ a context and $\alpha : \tau'$ a symbol in Γ such that $t = C[\bullet]$.

Notation. Given an occurrence occ we write $C_{occ}[\bullet]$ the associated context and α_{occ} its associated symbol. We write Occ_t the set of occurrences in the term t, and we may say that an occurrence is in t to denote the fact that it is in Occ_t.

The following proposition formalises the idea that if we take an occurrence in a term of the form $e_{[\forall i\ x_i \mapsto t_i]}$, then either (1) this occurrence is in one of the t_i, or (2) it is in e. This is depicted in Figure 4.1, e is represented in grey, the white triangles are the terms t_i, occ_1 is in one of the t_i (1), while occ_2 is in e (2).

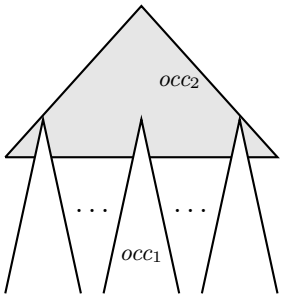


Figure 4.1.: Positions of occurrences

Proposition 4.4.1.

Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be a scheme, and $x_1, \ldots, x_k \in \mathcal{V}$. Let $e \in \mathcal{T}(\Sigma \uplus \mathcal{N} \uplus \{x_1, \ldots, x_k\})$ and $t = e_{[\forall i\ x_i \mapsto t_i]}$ for some $t_1, \ldots, t_k \in \mathcal{T}(\Sigma \uplus \mathcal{N})$. Finally let $(C[\bullet], \alpha)$ be an occurrence in t, with $\alpha \in \Sigma \uplus \mathcal{N}$. Then exactly one of the following holds.

(1) There exists $i \leq k$, and two contexts $C_e[\bullet]$ and $C_i[\bullet]$ such that:

$$e = C_e[x_i], \qquad t_i = C_i[\alpha], \qquad C[\bullet] = \Big(C_e\big[C_i[\bullet]\big]\Big)_{[\forall i\ x_i \mapsto t_i]}.$$

(2) There exists a context $C_e[\bullet]$ such that:

$$e = C_e[\alpha], \qquad C[\bullet] = C_e[\bullet]_{[\forall i\ x_i \mapsto t_i]}.$$

Proof. We prove this result by induction on the structure of e.

- If $e = \beta \in \Sigma \uplus \mathcal{N}$, then $t = \beta$, $C[\bullet] = \bullet$ and $\beta = \alpha$. Therefore (2) holds with $C_e[\bullet] = \bullet$, and (1) does not hold since e does not contains any occurrence of x_i for any i.
- If $e = x_i$ for some i, then $t = t_i$. Therefore (1) holds with $C_e[\bullet] = \bullet$ and $C_i[\bullet] = C[\bullet]$, and (2) does not hold since e does not contain any occurrence of α.
- If $e = e_1\ e_2$ then $t = e_{1[\forall i\ x_i \mapsto t_i]}\ e_{2[\forall i\ x_i \mapsto t_i]}$ and there exists $C'[\bullet]$ such that either $C[\bullet] = e_{1[\forall i\ x_i \mapsto t_i]}\ C'[\bullet]_{[\forall i\ x_i \mapsto t_i]}$ or $C[\bullet] = C'[\bullet]_{[\forall i\ x_i \mapsto t_i]}\ e_{2[\forall i\ x_i \mapsto t_i]}$. Both case are treated the same, so we just consider the case $C[\bullet] = e_{1[\forall i\ x_i \mapsto t_i]}\ C'[\bullet]_{[\forall i\ x_i \mapsto t_i]}$. By induction hypothesis, exactly one of the following holds.
  (1) There exists $i \leq k$, and two contexts $C'_{e_2}[\bullet]$ and $C'_i[\bullet]$ such that:

$$e_2 = C'_{e_2}[x_i], \qquad t_i = C'_i[\alpha], \qquad C'[\bullet] = \Big(C'_{e_2}\big[C'_i[\bullet]\big]\Big)_{[\forall i\ x_i \mapsto t_i]}.$$

  In which case (1) holds with $C_e[\bullet] = e_1\ C' e_2[\bullet]$ and $C_i[\bullet] = C'_i[\bullet]$. Furthermore (2) does not hold, indeed if it were the case, (2) would hold for e_2.

  (2) There exists a context $C_e[\bullet]$ such that:

$$e_2 = C'_{e_2}[\alpha], \qquad C[\bullet] = C'_{e_2}[\bullet]_{[\forall i\ x_i \mapsto t_i]}.$$

  In which case (2) holds with $C_e[\bullet] = e_1\ C'_{e_2}[\bullet]$. Furthermore (1) does not hold, indeed if it were the case, (1) would hold for e_2.

□

For the sake of simplicity we only define the relations on occurrences for the OI parallel rewriting, but it can be defined similarly for any rewriting. The OI parallel rewriting in a term rewriting system is the relation between two terms t and t' such that t' has been obtained by rewriting all OI redexes in t.

Definition 4.4.2 (OI Parallel rewriting)**.**

Let $\mathcal{S} = \langle \Gamma, \mathcal{V}, \mathcal{R} \rangle$ be a term rewriting system. The IO parallel relation $\Rightarrow_{\text{OI}}$ is inductively defined as follows. Given two terms $t, t' \in \mathit{termes}(\Gamma)$,

- If $t = \alpha\ t_1 \dots t_k$ such that $k < \mathit{arity}(\alpha)$ or there does not exists a rule associated with α in $\mathcal{R}$, then $t \Rightarrow_{\text{OI}} a\ t'_1 \dots t'_k$ such that for all i, $t_i \Rightarrow_{\text{OI}} t'_i$.
- If $t = \alpha\ t_1 \dots t_k$ with $\mathit{arity}(\alpha) = k$ then for all $\alpha\ x_1 \dots x_k \to e \in \mathcal{R}$, then $t \Rightarrow_{\text{OI}} e_{[\forall i\ x_i \mapsto t_i]}$.

As previously, one can show that if $t \Rightarrow_{\mathrm{OI}} t'$ then $t \rightarrow^*_{\mathrm{OI}} t'$ and that if the TRS is the one of a scheme, then for all t, there exists a unique t' such that $t \Rightarrow_{\mathrm{IO}} t'$. In that case given $t_0 : o$, if $t_0 \Rightarrow_{\mathrm{OI}} t_1 \Rightarrow_{\mathrm{OI}} t_2 \Rightarrow_{\mathrm{OI}} \cdots$, then the limit tree of $t_0^\perp \sqsubseteq t_1^\perp \sqsubseteq t_2^\perp \sqsubseteq \cdots$, is equal to the value tree of t_0, $\|t_0\|$.

Given a OI parallel derivation $t_0 \Rightarrow_{\mathrm{OI}} t_1 \Rightarrow_{\mathrm{OI}} t_2 \Rightarrow_{\mathrm{OI}} \cdots$, we say that the derivation is ***finite*** if there exists t_i such that t_i does not contain any redex. In particular for all $j \geq i$, $t_j = t_i$.

For terms of type o in a scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$, $\Rightarrow_{\mathrm{OI}}$ satisfies the following.

- If $t = a\ t_1 \dots t_k$ with $a \in \Sigma$, then $t \Rightarrow_{\mathrm{OI}} a\ t'_1 \dots t'_k$ such that for all i, $t_i \Rightarrow_{\mathrm{OI}} t'_i$.
- If $t = F\ t_1 \dots t_k$ with $F \in \mathcal{N}$ and $F\ x_1 \dots x_k \rightarrow e \in \mathcal{R}$, then $t \Rightarrow_{\mathrm{OI}} e_{[\forall i\ x_i \mapsto t_i]}$.

Given two terms t and t' such that $t \Rightarrow_{\mathrm{OI}} t'$ and an occurrence occ' in t', we want to describe the fact that either this occurrence has been *created* by the rewriting of a non terminal in t, or it were already present in t in which case we say that it as been *given* by an occurrence of the same symbol of t. We will write $occ \succ occ'$ if occ has been created by occ and $occ \gg occ'$ if occ' has been given by occ. These two relations are depicted in Figure 4.2 (creation relation is depicted by simple grey arrows, and conservation relation is depicted by double grey arrows).

Definition 4.4.3 (Conservation relation)**.**

Let $\mathcal{S} = \langle \Gamma, \mathcal{V}, \mathcal{R} \rangle$ be a term rewriting system and let t and t' be two terms such that $t \Rightarrow_{\mathrm{OI}} t'$. The ***conservation relation*** $\gg$ between the occurrences of t and of t' is inductively defined as follows.

- If $t = \alpha\ t_1 \dots t_k$ and $t' = e_{[\forall i\ x_i \mapsto t_i]}$ with $\alpha\ x_1 \dots x_k \rightarrow e \in \mathcal{R}$.

  Let $(C[\bullet], \beta)$ be an occurrence of t' such that (1) holds (*i.e.* the occurrence is in t_i for some i), and let $C_i[\bullet]$ be the associated context defined as in Proposition **4.4.1**. Then

  $$(\alpha\ t_1 \dots C_i[\bullet] \dots t_k\ ,\ \beta) \gg (C[\bullet], \beta).$$

- If $t = \gamma\ t_1 \dots t_k$ such that $k < \mathit{arity}(\gamma)$ or there does not exists a rule associated with γ in $\mathcal{R}$, and $t' = \gamma\ t'_1 \dots t'_k$ such that for all i, $t_i \Rightarrow_{\mathrm{OI}} t'_i$. Then $(\bullet, \gamma) \gg (\bullet, \gamma)$, and for all i and for all occurrences $occ_i = (C_i[\bullet], \beta)$ in t_i and $occ'_i = (C'_i[\bullet], \beta)$ in t'_i such that $occ_i \gg occ'_i$,

  $$(\alpha\ t_1 \dots C_i[\bullet] \dots t_k\ ,\ \beta) \gg (\alpha\ t'_1 \dots C'_i[\bullet] \dots t'_k\ ,\ \beta).$$

Definition 4.4.4 (Creation relation)**.**

Let $\mathcal{S} = \langle \Gamma, \mathcal{V}, \mathcal{R} \rangle$ be a term rewriting system and let t and t' be two terms such that $t \Rightarrow_{\mathrm{OI}} t'$. The ***creation relation*** $\succ$ between the occurrences of t and of t' is inductively defined as follows.

- If $t = \alpha\ t_1 \dots t_k$ and $t' = e_{[\forall i\ x_i \mapsto t_i]}$ with $\alpha\ x_1 \dots x_k \to e \in \mathcal{R}$.

  Let $(C[\bullet], \eta)$ be an occurrence of t' such that (2) holds (*i.e.* the occurrence appears in e), then

  $$(\bullet\ t_1 \dots t_k, \alpha) \succ (C[\bullet], \eta).$$

- If $t = \gamma\ t_1 \dots t_k$ such that $k < \mathit{arity}(\gamma)$ or there does not exists a rule associated with γ in $\mathcal{R}$, and $t' = \gamma\ t'_1 \dots t'_k$ such that for all i, $t_i \Rightarrow_{\mathrm{OI}} t'_i$. Then for all i and for all occurrences $\mathit{occ}_i = (C_i[\bullet], \alpha)$ in t_i and $\mathit{occ}'_i = (C'_i[\bullet], \eta)$ in t'_i such that $\mathit{occ}_i \succ \mathit{occ}'_i$,

  $$(\alpha\ t_1 \dots C_i[\bullet] \dots t_k\ ,\ \alpha) \succ (\alpha\ t'_1 \dots C'_i[\bullet] \dots t'_k\ ,\ \eta).$$

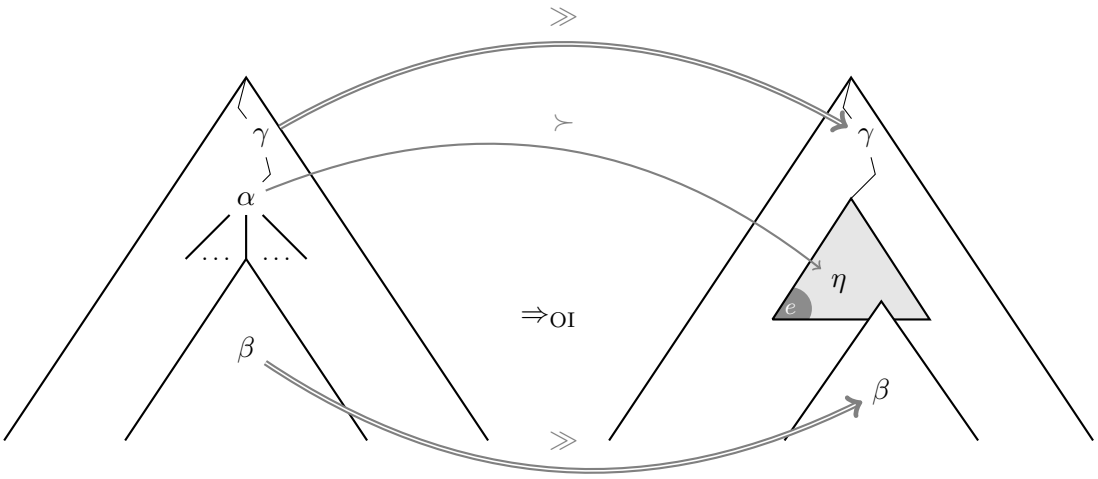


Figure 4.2.: Relations between occurrences

Now we want to look at occurrences of terms inside a derivation. To specify the term from which comes an occurrence we say that an occurrence in a derivation is an occurrence of a term inside the derivation along with the index of the term. This allows to avoid ambiguity if two terms in the derivation are the same.

In the following we let $d = t_0 \Rightarrow_{\mathrm{OI}} t_1 \Rightarrow_{\mathrm{OI}} t_2 \Rightarrow_{\mathrm{OI}} \cdots$ be an OI parallel derivation.

Definition 4.4.5 (Occurrences in derivation)**.**

We define the set Occ_d of ***occurrences*** in d as:

$$Occ_d = \{(i, occ_i) \mid occ_i \text{ is an occurrence in } t_i\}.$$

The conservation relation in the occurrences derivation is defined as the reflexive and transitive closure of the conservation relation in a parallel rewrite.

Definition 4.4.6 (Conservation relation in Occ_d)**.**

We define the relation $\gg_d$ (or simply $\gg$ when the derivation is clear from the context) on $Occ_d \times Occ_d$ as the smallest reflexive and transitive relation satisfying that if occ_i and occ_{i+1} are occurrences of respectively t_i and t_{i+1} such that $occ_i \gg_{t_i \to t_{i+1}} occ_{i+1}$ then $(i, occ_i) \gg_d (i+1, occ_{i+1})$.

The creation relation is defined from the conservation relation as if occ creates occ' in a parallel rewrite, and then $occ' \gg_d occ''$, then occ also creates occ''.

Definition 4.4.7 (Creation relation in Occ_d)**.**

We define the relation $\succ_d$ (or simply $\succ$ when the derivation is clear from the context) on $Occ_d \times Occ_d$ as $(i, occ_i) \succ_d (j, occ_j)$ if occ_i (*resp.* occ_j) is an occurrence of t_i (*resp.* t_j), and if there exists an occurrence occ_{i+1} in t_{i+1} such that $occ_i \succ_{t_i \to t_{i+1}} occ_{i+1}$ and $(i+1, occ_{i+1}) \gg_d (j, occ_j)$.

4.4.2. Parity games

A parity game is a two players turn-based game played on a labelled graph, whose winning condition is described by a parity constraint defined as in parity automata.

Definition 4.4.8 (Parity games)**.**

A ***parity game*** is a tuple $\langle V = V_E \uplus V_A, E, \Omega \rangle$ such that:

- V_E is a finite set of ***vertices of Eve***,
- V_A is a finite set of ***vertices of Adam***,
- $E \subseteq V \times V$, is the set of ***actions***,
- $\Omega : V \to \mathcal{N}$ is the ***colouring mapping***.

A play in a parity game is a sequence of vertex that respects the set of actions. The winning condition in such a play is similar to the one of parity automata.

Definition 4.4.9 (Play)**.**

Given a game $\langle V = V_E \uplus V_A, E, \Omega \rangle$ and a vertex v, a ***play*** from v is a finite or countable sequence $v_0, v_1, v_2, \cdots$ such that $v_0 = v$ and for all i, $(v_i, v_{i+1}) \in E$. It is ***maximal*** if either it is infinite or it is finite and there are no possible action from the last vertex v_k, *i.e.* $\{v' \mid (v_k, v') \in E\} = \emptyset$.
A maximal play is ***winning*** (for Eve) if:

- either it is finite and the last vertex v is in V_A,
- or the maximum colour appearing infinitely often in $\Omega(v_0), \Omega(v_1), \ldots$ is even.

A strategy is a mapping that decides which action to play according to the past. It is positional if the decision only takes into account the current state, and not the whole history.

Definition 4.4.10 (Strategy)**.**

A ***strategy*** for Eve is a partial mapping $\pi : V^* \cdot V_E \to V$ such that for any finite play $p = v_1 \cdots v_k$ with $v_k \in V_E$, if $\pi(p)$ is defined then: $(v_k, \pi(p)) \in E$.
A play $p = v_1 v_2 \cdots$ ***agrees with the strategy*** π if for all strict sub-play $p' = v_1 \cdots v_k$ of p with $v_k \in V_E$, $\pi(p')$ is defined and $v_{k+1} = \pi(p')$.
A strategy π is ***winning*** from vertex v if for all maximal play p from v that agrees with π, p is winning.
A strategy π is ***positional*** if there exists a partial mapping $f : V_E \to V$ such that for all play $p = v_1 \cdots v_k$ with $v_k \in V_E$, then $\pi(p)$ is defined if and only if $f(v_k)$ is defined, and $\pi(p) = f(v_k)$. In general we refer to f as the strategy π.

A vertex is winning if Eve has strategy that allows her to win any play starting from this vertex.

Definition 4.4.11 (Winning vertex)**.**

Given a game $\langle V = V_E \uplus V_A, E, \Omega \rangle$, a vertex v is ***winning*** if there exists a strategy π that is winning from v.

For every game, there exists a positional strategy that allows to win from any winning vertex.

Proposition 4.4.2 ([EJ91])**.**

Given a game $\langle V = V_E \uplus V_A, E, \Omega \rangle$, one can compute a positional strategy f such that f is winning from all winning vertex. This problem is in $\mathbf{NP} \cap \mathbf{coNP}$.

4.4.3. Adding colours to annotations

Vertices in the Kobayashi-Ong game will be either annotated symbols or annotated terms. However, for technical reasons the annotation needs to be enriched the following way.

We add to Γ_Θ annotations of colours, without changing the types:

$$\Gamma_{\Theta+} = \{\alpha^{\theta,m} : \tau \mid \alpha^\theta : \tau \in \Gamma_\Theta, m \in M\}.$$

Given a term t in $\mathcal{T}(\Gamma_{\Theta+} \uplus \mathit{Boxes})$ we write $\mathit{rem}(t)$ the term in $\mathcal{T}(\Gamma_{\Theta} \uplus \mathit{Boxes})$ obtained by removing these colour annotations (*i.e.* $\mathit{rem}(\alpha^{\theta,m}) = \alpha^{\theta}$, $\mathit{rem}(\Box^{\tau}) = \Box^{\tau}$ and $\mathit{rem}(t_1\ t_2) = \mathit{rem}(t_1)\ \mathit{rem}(t_2)$).

Given a term $t \in \mathcal{T}(\Gamma)$, a term $t' \in \mathcal{T}(\Gamma_{\Theta+})$ is a ***θ-annotation*** of t if $\mathit{rem}(t')$ is a θ-annotation of t (no condition is required on the colour annotations). We write $CA^{+}(\Gamma) \subseteq \mathcal{T}(\Gamma_{\Theta+} \uplus \mathit{Boxes})$ the set of correct annotations of some terms t in $\mathcal{T}(\Gamma)$. We write $CC^{+}(\Gamma)$ the set of correct contexts of $\Gamma_{\Theta+}$.

Given a correct context $C[\bullet] \in CC^{+}(\Gamma)$ the colour of $C[\bullet]$ is the greatest colour seen along the path from the head of $C[\bullet]$ to the occurrence of $\bullet$. This notion does not depends on the added colours, but only to the colour in the transitions. It is illustrated in Figure 4.3, in this representation, in a term $\alpha^{\theta,m}\ \vec{t_1} \dots \vec{t_k}$ and an argument $t_i^{\theta',m'}$ the colour labelling the edge between $\alpha^{\theta,m}$ and $t_i^{\theta',m'}$ is m' (and not m).

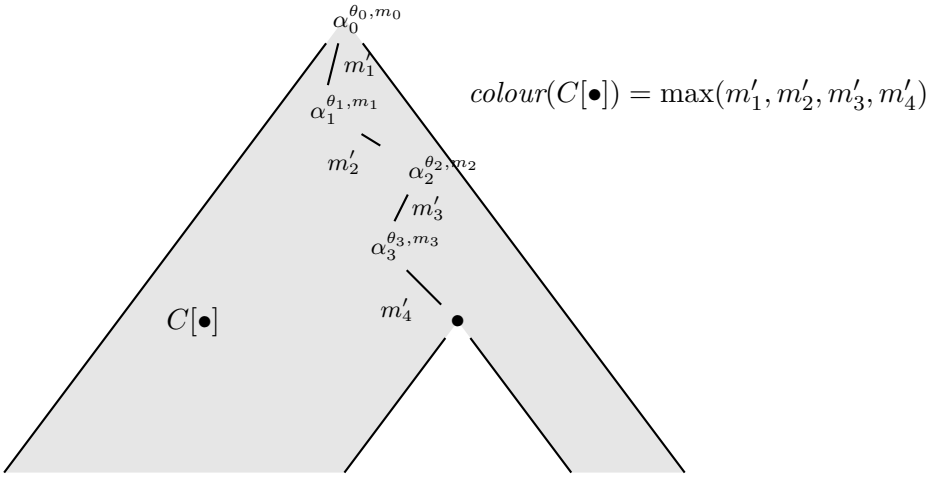


Figure 4.3.: The colour of a context

Definition 4.4.12 (Colour of a context)**.**

Let $C[\bullet]$ be a correct context. The ***colour*** of $C[\bullet]$ written $\mathit{colour}(C[\bullet])$ is inductively defined by:

- If $C[\bullet] = \bullet\ \vec{t_1} \cdots \vec{t_k}$, $\mathit{colour}(C[\bullet]) = 0$,
- If $C[\bullet] = \alpha^{q \to (\sigma_1,\dots,\sigma_k),m}\ \vec{t_1} \dots t_{j \leq k}$ with $\alpha : \tau_1 \to \cdots \to \tau_k \to o$ such that there exists i and $(\theta, m') \in \sigma_i$ satisfying $t_i^{(\theta,m')} = C'[\bullet]$ for some context $C'[\bullet]$. Then

$$\mathit{colour}(C[\bullet]) = \max\big(m', \mathit{colour}(C'[\bullet])\big).$$

Observe that given two correct contexts $C_1[\bullet]$ and $C_2[\bullet]$,

$$\mathit{colour}\left(C_1\left[C_2[\bullet]\right]\right) = \max\left(\mathit{colour}(C_1[\bullet]), \mathit{colour}(C_2[\bullet])\right).$$

Definition 4.4.13 (Well-coloured occurrence, term)**.**

Given a term $t \in \mathcal{T}(\Gamma)$, $t' \in CA^+(\Gamma)$ a correct annotation of t', and an occurrence $occ = (C[\bullet], \alpha^{\theta,m})$ in t'. Then occ is ***well-coloured*** if $m = \mathit{colour}(C[\bullet])$.
Given a term $t \in \mathcal{T}(\Gamma)$ and $t' \in CA^+(\Gamma)$ a correct annotation of t. The term t' is ***well-coloured*** if all occurrences of t' are well coloured.

Remark that for all correct annotation $t' \in CA(\Gamma)$ of a term t there exists a unique well-coloured annotation $r \in CA^+(\Gamma)$ of t such that $\mathit{rem}(t) = t'$. We illustrate a well-coloured annotation in Figure 4.4, the colour annotating occurrence must correspond to the maximum colour seen along the path from the head of the term.

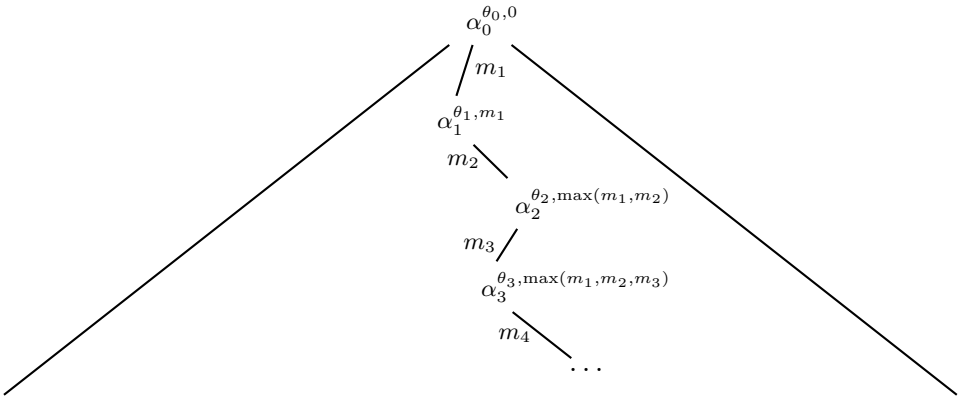


Figure 4.4.: A well-coloured annotation

In the following we fix a scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ and an automaton $\mathcal{A} = \langle \Sigma, M, Q, \delta \rangle$.

Let $\delta_\Sigma = \{a^{q \to (\{q_1,m_1\},\ldots,\{q_k,m_k\}),m} \mid a^{q \to ((q_1,m_1),\ldots,(q_k,m_k))} \in \delta, m \in M\}$, in particular $\mathit{rem}(\delta_\Sigma) = \tilde{\delta}$. A ***coloured run*** of $\mathcal{A}$ is a correct annotation $t \in CA^+(\Sigma \uplus \mathcal{N})$ such that $\mathit{rem}(t)$ is a run of $\mathcal{A}$. We write $\mathcal{A}^+(\Sigma \uplus \mathcal{N})$ the set of coloured runs.

Given a rewrite rule $F\ x_1 \ldots x_k \to e \in \mathcal{R}$ and an annotated version $F^{\theta,m}$ of F. An annotation e' of e is compatible with $F^{\theta,m}$ if it is well coloured and all variables x_i in e' are annotated with transitions specified in θ, in the sense of the following definition.

Definition 4.4.14 (Annotated rewrite rule)**.**

Let $F\ x_1 \dots x_k \to e \in \mathcal{R}$ and $F^{\theta,m} \in \mathcal{N}_{\Theta+}$ with $\theta = q \to (\sigma_1, \dots, \sigma_k)$. The set $\delta_{F^{\theta,m}} \subseteq \mathcal{V}_{\Theta+}$ of annotated variables is defined as

$$\delta_{F^{\theta,m}} = \bigcup_{i \leq k} \{x_i^{\theta',m'} \mid (\theta', m') \in \sigma_i\}.$$

Let e' be a q-run on e. It is ***compatible with*** $F^{\theta,m}$ if

- e' is well-coloured,
- e' conforms to $\delta_{F^{\theta,m}}$ on $\mathcal{V}$.

We denote $Comp(F^{\theta,m})$ the set of all q-runs on e compatible with $F^{\theta,m}$.

4.4.4. Kobayashi-Ong game

We are now in position to give the definition of the Kobayashi-Ong game associated with the scheme and the automaton. In this game, Eve's vertices will be annotated non-terminal symbols, *i.e.* symbols of $\mathcal{N}_{\Theta+}$, Adam vertices are the runs compatible with the non-terminal symbols.

On $F^{\theta,m}$, Eve must chose a term e in $Comp(F^{\theta,m})$, and on e, Adam chose a symbol $H^{\theta',m'} \in \mathcal{N}_{\Theta+}$ occurring in e. The colours of Eve's vertices is the colours annotating the symbols, *e.g.* the colour of the vertex $F^{\theta,m}$ is m, and the colour of Adam's vertices is 0 for all vertices.

Definition 4.4.15 (The Kobayashi-Ong game)**.**

The ***Kobayashi-Ong game*** $\langle V = V_E \uplus V_A, E, \Omega \rangle$ associated with $\mathcal{G}$ and $\mathcal{A}$ is defined as follows.

- $V_E = \mathcal{N}_{\Theta+}$,
- $V_A = \bigcup_{F^{\theta,m} \in \mathcal{N}_{\Theta+}} Comp(F^{\theta,m})$,
- $E = \{(F^{\theta,m}, e') \mid e' \in Comp(F^{\theta,m})\} \uplus \{(e', H^{\theta',m'}) \mid H^{\theta',m'} \in \delta_{e'}\}$,
- For all $F^{\theta,m} \in V_E$, $\Omega(F^{\theta,m}) = m$, for all $e' \in V_A$, $\Omega(e') = 0$.

Kobayashi and Ong have shown in [KO09] that if Eve wins in the game from vertex $S^{q,0}$ then the value tree of the scheme is accepted from state q.

Theorem 4.4.3 ([KO09])**.**

For all $q \in Q$, $S^{q,0}$ is winning if and only if $\mathcal{A}$ accepts $\|\mathcal{G}\|$ from state q.

This theorem gives an algorithm to solve the automata model-checking problem (equivalently the MSO model-checking problem): first compute the game, then solve the game. This algorithm is in n-EXPTIME, n being the order of the scheme, and Ong have shown that the model-checking problem is n-EXPTIME complete [Ong06].

Intuition. We informally describe how, if Eve has a winning strategy, one can construct an accepting run on the value tree of the scheme.

First we remove the colours and give an informal *process* that constructs a run of the automaton from a winning strategy of Eve. Then we give an intuition about why, if Eve follows a winning strategy, the maximum colour appearing infinitely often along each branche of this run is even.

We first study the case where the set of colours is equal to $\{0\}$, in that case eve wins a play if she can always play when it is her turn. We describe the intuition on an order-1 scheme, which allows us to use the graphical representations introduced in the beginning of this chapter.

Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ with:

- $\mathcal{V} = \{x : o\}$,
- $\Sigma = \{\mathtt{a}, \mathtt{d} : o^2 \to o, \mathtt{b}, \mathtt{c} : o \to o\}$,
- $\mathcal{N} = \{S : o, H : o, F : o \to o\}$,
- $\mathcal{R}$ contains the following rewrite rules.

$$\begin{array}{lcl} S & \to & \mathtt{a}\ (F\ H)\ (\mathtt{b}\ H) \\ H & \to & \mathtt{c}\ H \\ F\ x & \to & \mathtt{d}\ x\ (\mathtt{b}\ x) \end{array}$$

Let $\mathcal{A} = \langle \Sigma, Q = \{q, q_1, q_2, q_3\}, \delta, M = \{0\} \rangle$ be the automaton such that δ contains the following annotated symbols (we omit the colours since they are always equal to 0):

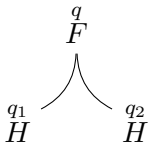
$$\mathtt{a}^{q \to (q,q)}, \quad \mathtt{b}^{q \to q_2}, \quad \mathtt{b}^{q \to q_3}, \quad \mathtt{c}^{q_1 \to q_1}, \quad \mathtt{c}^{q_2 \to q_2}, \quad \mathtt{c}^{q_3 \to q_3}, \quad \mathtt{d}^{q \to (q_1,q)}.$$

In this example (and in this example only), we consider equal the symbols of δ and $\tilde{\delta}$, *i.e.* we do not differentiate terminal symbols annotated with simple transitions, *e.g.* $\mathtt{a}^{q \to (q,q)}$ and those annotated with higher-order transitions, *e.g.* $\mathtt{a}^{q \to (\{q\},\{q\})}$. In accordance with this equality, we omit the occurrences of $\Box^o$ in the graphical representation of terms, for example the terms $\mathtt{b}^{q \to \{q_3\}}\ \Box^o\ \Box^o\ \Box^o\ H^{q_3}$ is simply represented:

This applies also to terms whose head is a non-terminal symbol, for example the term:

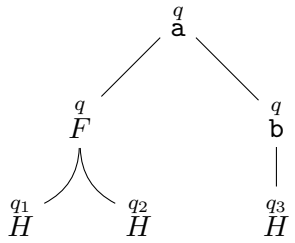
$$F^{q \to \{q_1, q_2\}} \quad \square^o \; H^{q_1} H^{q_2} \; \square^o$$

Is graphically represented by:



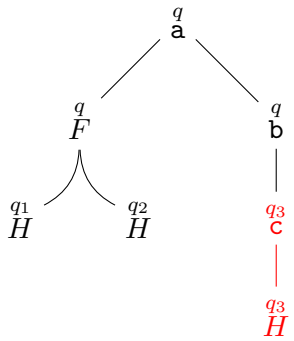
Assume now that Eve follows a winning strategy, *i.e.* each time Eve has to play on a non-terminal symbol α^θ, $Comp(\alpha^\theta)$ is non-empty. We show that by looking at the winning strategy of Eve, one can construct a run of the automaton over $\|\mathcal{G}\|$ from state q.

First we look at her strategy from vertex S^q, *i.e.* we asks her to give a q-run on the term $\mathtt{a}\ (F\ H)\ (\mathtt{b}\ H)$:

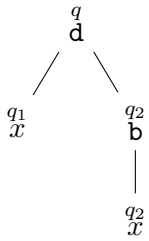


Now we pick one non-terminal symbol in this term, for example H^{q_3} and asks Eve to play on that symbol, *i.e.* to give a q_3-run on $\mathtt{c}\ H$.

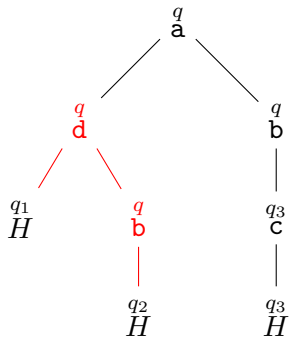
Then we place this run inside the previously given run, replacing the occurrence of H^{q_3} by the new term.



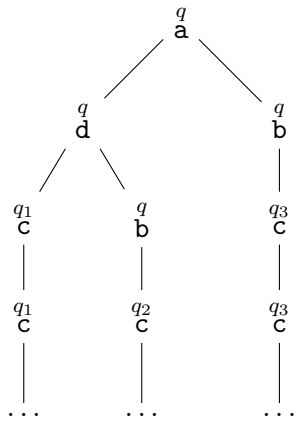
We pick another non-terminal symbol in this term, for example $F^{q \to \{q_1, q_2\}}$ and again, we ask Eve for an annotation of d x (c x) compatible with $F^{q \to \{q_1, q_2\}}$, *i.e.* a q-run such that the annotated versions of x are either x^{q_1} or x^{q_2}. We know that there must exists such a run, otherwise there would exists a play where she follows her winning strategy but loses the play.



Again we replace $F^{q \to \{q_1, q_2\}}$ by this run in the term:



Since Eve follows her winning strategy, she can always play for all non-terminal symbols appearing in that term. Therefore, by pursuing this procedure, we construct a run of $\mathcal{A}$ over $\|\mathcal{G}\|$.



We now add colours to the intuition, and show that if Eve follows her winning strategy the colour of every branch (*i.e.* the maximum colour appearing infinitely often) is even. To simplify the idea, we look at a scheme whose value tree contains a single, infinite, branch.

Let $\mathcal{G}' = \langle \mathcal{V}', \Sigma', \mathcal{N}', I, \mathcal{R}' \rangle$ be the order-2 scheme with:

- $\Sigma' = \{\mathtt{e}, \mathtt{f}, \mathtt{g} : o \to o\}$,
- $\mathcal{V}' = \{x : o, \varphi : o \to o\}$,
- $\mathcal{N}' = \{S' : o, I : o, J, K : o \to o, L : (o \to o) \to o\}$,
- $\mathcal{R}'$ contains the following rewrite rules.

$$\begin{array}{lcl} I & \to & L\ K \\ J\ x & \to & \mathtt{e}\ (\mathtt{f}\ x) \\ K\ x & \to & \mathtt{e}\ (\mathtt{g}\ x) \\ L\ \varphi & \to & \varphi\ (J\ I) \end{array}$$

Let $\mathcal{A}'$ be the automaton $\langle \Sigma', Q', M', \delta' \rangle$ with:

- $Q' = \{q_1, q_2\}$,
- $M' = \{0, 1, 2, 3, 4\}$,
- δ' contains the following annotated symbols:

$$\mathtt{e}^{q_1 \to (q_2, 2)}, \quad \mathtt{e}^{q_2 \to (q_1, 2)}, \quad \mathtt{f}^{q_1 \to (q_1, 3)}, \quad \mathtt{g}^{q_2 \to (q_2, 4)}.$$

We assume that Eve has a winning strategy (that we will describe step by step), to prove that the value tree of $\mathcal{G}'$ is accepted from state q_1. The process starts with the symbol $I^{q_1, 0}$. Eve strategy pick the following q_1-run on the term $L\ K$:

$$L^{q_1 \to \{(q_1 \to (\{q_2\}, 4), 0)\}, 0}\ \square^{o \to o}\ \ldots\ K^{q_1 \to (\{q_2\}, 4), 0}\ \ldots\ \square^{o \to o}.$$

We use the same graphical representation as previously, but we add colours annotation to the symbols, *e.g.* this term is represented as follows.

$$\begin{array}{c} q_1 \\ L^0 \\ \mid 0 \\ q_1 \to (\{q_2\},3) \\ K^0 \end{array}$$

Now observe that during the process, given a *redex*, when we replace the head of the redex by the compatible term chosen by Eve according to her winning strategy, then every symbols that have been added are *well-coloured*. Let us replace $L^{q_1 \to \{(q_1 \to (\{q_2\},4),0)\},0}$ by the following choice of Eve.

$$\begin{array}{c} q_1 \\ \varphi^0 \\ \mid 4 \\ q_2 \\ J^4 \\ \mid 3 \\ q_1 \\ I^4 \end{array}$$

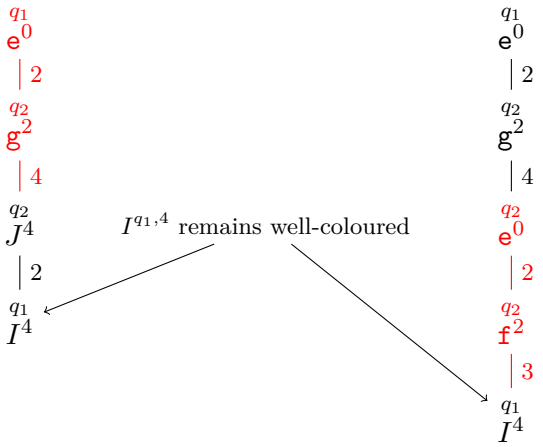
Then every added symbol is well coloured as depicted below. This is a simple consequence of the fact that the annotations chosen by Eve must be well-coloured.

$$\begin{array}{c} q_1 \\ K^0 \\ \mid 4 \\ q_2 \\ J^4 \\ \mid 2 \\ q_1 \\ I^4 \end{array}$$

Furthermore when we keep on replacing some symbols by the choices of Eve, the *unchanged symbols* remain well-coloured. We show below that when we transform $K^{q_1 \to (\{q_2\},4),0}$ and then $J^{q_2 \to (\{q_1\},2),4}$, the occurrence of $I^{q_1,4}$ remain well coloured *i.e.* the maximum colour appearing from the head of the term to this occurrence is 4.

Replace $K^{q_1 \to (\{q_2\},4),0}$ by:

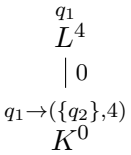
$$\mathsf{e}^{q_1 \to (\{q_2\},2),0}\ \Box^o(\mathsf{g}^{q_2 \to (\{q_2\},4),2}\ \Box^o\ x^{q_2,4})$$



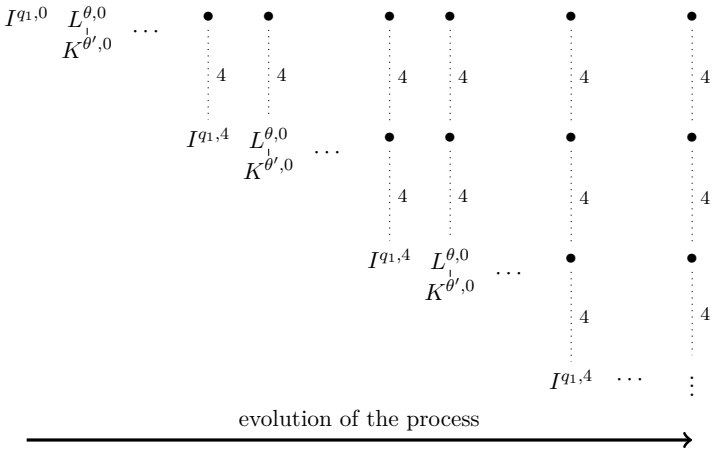
Replace $J^{q_2 \to (\{q_1\},2),3}$ by:

$$\mathsf{e}^{q_2 \to (\{q_1\},2),0}(\mathsf{f}^{q_1 \to (\{q_1\},3),2}\ x^{q_1,4}\ \Box^o)\ \Box^o$$

Finally, assume that Eve's strategy on $I^{q_1,4}$ is to play the annotation of $L\ K$ already introduced:



It means that in this example the procedure for creating the run loop and asks Eve to rewrite always the same symbols. In particular, we rewrite $L^{\theta,0}$ (with $\theta = q_1 \to \{(q_1 \to (\{q_2\}, 4), 0)\}$), then we create a little piece of run, then we rewrite $I^{q_1,4}$, then we rewrite $L^{\theta,0}$ again, and so on. As we have seen, the little pieces of run we construct have the same maximum colour as the one annotating $I^{q_1,4}$, *i.e.* 4. We illustrate this evolution below, the dashed line represent the little pieces of run, and we skip the steps of rewriting $L^{\theta,0}$, $K^{q_1 \to (\{q_2\},4),0}$ and $J^{q_2 \to (\{q_1\},2),4}$ (represented by $\cdots$).



This induces a run on the value tree of $\mathcal{G}'$ such that in each little piece of run, the maximum colour is 4, therefore the maximum colour appearing infinitely often in that run is 4.

In the general case, there is not necessarily a loop, but for every infinite branch we can find a sequence of symbols that appears along the branch in the process described above:

$$\alpha_1^{\theta_1,m_1}, \alpha_2^{\theta_2,m_2}, \alpha_3^{\theta_3,m_3}, \dots$$

and such that for all i, $\alpha_{i+1}^{\theta_{i+1},m_{i+1}}$ appears in the compatible annotation chosen by Eve on $\alpha_i^{\theta_i,m_i}$ (the existence if such a sequence is a non-trivial result stated in [KO09]). For the same reason as in the example, in that process the maximum colour that appear in the little pieces of run between these symbols, are the m_i.

In our example the sequence is:

$$I^{q_1,0}, L^{\theta,0}, I^{q_1,4}, L^{\theta,0}, I^{q_1,4}, L^{\theta,0}, I^{q_1,4}, \dots$$

Note that this sequence corresponds to a possible play in the game where Eve follows her winning strategy: Eve apply her strategy on $\alpha_1^{\theta_1,m_1}$ choosing the compatible annotation e_1; then Adam pick an occurrence of $\alpha_2^{\theta_2,m_2}$ in e_1, then Eve choses e_2, and so on.

Since Eve wins that play, the maximum colour appearing in the sequence $m_1, m_2, m_3, \dots$ is even. The maximum colour appearing infinitely often in this sequence is also the maximum colour appearing infinitely often along the branch, therefore, as we can find such a sequence of symbols for every infinite branch, the run produced by the process is accepting.

Remark. This intuition is formalised in [KO09], they construct the process described above, and prove that it generates an accepting run. The proof of the selection, as we will see, consists in showing that one can formalise this process as a HORS.

4.4.5. Solving problem 8

Now we prove that once we have computed the set of winning vertices of the game, one can solve Problem 8. In the following, let $Win \subseteq V_E$ be the set of winning vertices of Eve, and let f be a positional winning strategy for Eve. First we show that the colour annotation has no impact on whether a vertex is winning or not.

Lemma 4.4.4.

Let $F : \tau \in \mathcal{N}$, $\theta : \tau \in \Theta$, and $m_0 \in M$. If $F^{\theta,m_0} \in Win$ then for all $m \in M$, $F^{\theta,m} \in M$.

Proof. Indeed, if we define f' the strategy such that $dom(f') = dom(f) \cup \{F^{\theta,m}\}$ for all $H^{\theta',m'} \neq F^{\theta,m}$, $f'(H^{\theta',m'}) = f(H^{\theta',m'})$ and $f'(F^{\theta,m}) = f(F^{\theta,m_0})$ (this is possible since $Comp(F^{\theta,m}) = Comp(F^{\theta,m_0})$, then if there exists a play $F^{\theta,m_0} \cdot v_1 \cdot v_2 \cdots$ that agrees with f, then the play $F^{\theta,m} \cdot v_1 \cdot v_2 \cdots$ agrees with f'. In particular since there exists a winning play that agrees with f from F^{θ,m_0}, there exists a winning play that agrees with f' from $F^{\theta,m}$. □

We now define $\delta_{\mathcal{N}} = \{F^{\theta} \mid \exists m\ F^{\theta,m} \in Win\}$. Thanks to Lemma **4.4.4**, we know that $Win = \{F^{\theta,m} \mid F^{\theta} \in \delta_{\mathcal{N}}, m \in M\}$. Let $t : o$ be a term of $\mathcal{T}(\Sigma \uplus \mathcal{N})$. Recall that the scheme $\mathcal{G}_t = \langle \mathcal{V}, \Sigma, \mathcal{N}', \mathcal{R}', I \rangle$ is defined by:

- $\mathcal{N}' = \mathcal{N} \uplus \{I : o\}$,
- $\mathcal{R}' = \mathcal{R} \uplus \{I \to t\}$.

We will compare the game associated with $\mathcal{G}$ to the one associated with $\mathcal{G}_t$, therefore let $\mathbb{G} = \langle V = V_E \uplus V_A, E, \Omega \rangle$ be the game associated with $\mathcal{G}$ and $\mathbb{G}_t = \langle V' = V'_E \uplus V'_A, E', \Omega' \rangle$ be the one associated with $\mathcal{G}_t$.

We have:

- $V'_E = V_E \uplus \{I^{q,m} \mid q \in Q, m \in M\}$,
- $V'_A = V_A \cup \bigcup_{q,m} Comp(I^{q,m})$,
- $E' = E \cup \{(I^{q,m}, s) \mid q \in Q, m \in M, s \in Comp(I^{q,m})\}$.

Furthermore, Let $Win' \subseteq V'_E$ be the winning vertices of Eve in $\mathbb{G}_t$. Observe that since for all $F\ x_1 \dots x_k \to e \in \mathcal{R}$ there are no occurrence of I in e, the set of plays $v_1 \cdot v_2 \cdot \dots$ in $\mathbb{G}_t$ such that $v_1 \neq I^{q,m}$ for any q and m, is equal to the set of plays in $\mathbb{G}$.

We want to show that $\delta_{\mathcal{N}}$ satisfies the requirement of Problem 8, *i.e.* there exists a run $t' \in \mathcal{T}(\Sigma_\Theta \uplus \mathcal{N}_\Theta)$ of $\mathcal{A}$ on t, that conforms to $\delta_{\mathcal{N}}$ on $\mathcal{N}$ if and only if $\|t\|_{\mathcal{G}}$ is accepted by $\mathcal{A}$ from state q. The main ingredient is the following proposition.

Proposition 4.4.5.

Let $q \in Q$. There exists a run $t' \in \mathcal{A}(\Sigma \uplus \mathcal{N})$ on t that conforms to $\delta_{\mathcal{N}}$ on $\mathcal{N}$ if and only if for all m, $I^{q,m} \in \mathit{Win}'$.

Proof. Let s be the well coloured annotation of t such that $\mathit{rem}(s) = t'$. Then $s \in \mathit{Comp}(()I^{q,m})$. Let f' be the positional strategy for Eve such that $\mathit{dom}(f') = \mathit{dom}(f) \cup \{I^{q,m}\}$; for all $H^{\theta',m'} \neq I^{q,m}$, $f'(H^{\theta',m'}) = f(H^{\theta',m'})$ and $f'(I^{q,m}) = s$.

Let $p = I^{q,m} \cdot v_1 \cdot v_2 \cdot v_3 \cdots$ be a play in $\mathbb{G}_t$ agreeing with f'. Then $v_2 \in \mathit{Win}$ and the play $v_2 \cdot v_3 \cdots$ is a play in $\mathbb{G}$ that agrees with f. Therefore it is winning for eve, therefore p is winning for Eve in $\mathbb{G}_t$.

On the other hand suppose that there exists no run $t' \in \mathcal{T}(\Sigma_\Theta \uplus \mathcal{N}_\Theta)$ of $\mathcal{A}$ on t that conforms to $\delta_{\mathcal{N}}$ on $\mathcal{N}$. Let $m \in M$. Let f' be a strategy for Eve. Then there exists an occurrence of an annotated nonterminal $H^{\theta',m'}$ in e such that $H^{\theta'} \notin \delta_{\mathcal{N}}$, *i.e.* $H^{\theta',m'} \notin \mathit{Win}$. Therefore any play $I^{q,m} \cdot f(I^{q,m}) \cdot H^{\theta',m'} \cdot v_4 \cdots$ is losing for Eve. In particular f' is not a winning strategy. Hence $I^{q,m}$ is not a winning vertex in $\mathbb{G}_t$. □

Now we can state the following corollary.

Corollary 4.4.6.

For all term $t : o$, there exists a run $t' \in \mathcal{T}(\Sigma_\Theta \uplus \mathcal{N}_\Theta)$ of $\mathcal{A}$ on t, that conforms to $\delta_{\mathcal{N}}$ on $\mathcal{N}$ if and only if $\|t\|_{\mathcal{G}}$ is accepted by $\mathcal{A}$ from state q.

Proof. From the previous proposition, there exists a run $t' \in \mathcal{T}(\Sigma_\Theta \uplus \mathcal{N}_\Theta)$ of $\mathcal{A}$ on t, that conforms to $\delta_{\mathcal{N}}$ on $\mathcal{N}$ if and only if for all m, $I^{q,m} \in \mathit{Win}'$, if and only if $\|\mathcal{G}_t\|$ is accepted by $\mathcal{A}$ from state q, if and only if $\|t\|_{\mathcal{G}}$ is accepted by $\mathcal{A}$ from state q. □

We summarise below the algorithm that solves Problem 8.

Input: A scheme $\mathcal{G}$, an automaton $\mathcal{A}$.

1. Compute the Kobayashi Ong game associated with $\mathcal{G}$ and $\mathcal{A}$.
2. Compute the set Win of winning vertices of the game.

Output: $\delta_{\mathcal{N}} = \{F^\theta \mid \exists m\ F^{\theta,m} \in \mathit{Win}\}$ such that for all term $t : o$, there exists a run from $\mathcal{A}$ on t from state q that conforms to $\delta_{\mathcal{N}}$ on $\mathcal{N}$ if and only if $\mathcal{A}$ accepts $\|t\|$ from state q.

4.5. Selection

We show how to solve the selection problem. More precisely we show that if we compute a positional winning strategy for the Kobayashi-Ong game, one can construct a scheme $\mathcal{G}'$ whose value tree is an accepting run of $\mathcal{A}$ on $\|\mathcal{G}\|$.

The proof proceeds in three steps:

1. We introduce a TRS on annotated symbols, extract from it a decorated ARS and give a condition on derivation in this ARS such that if it is satisfy by a derivation d, the limit tree of d is a run of the automaton.

2. We define a condition on such derivations, such that if it is satisfied, the associated run is accepting.

3. We show that if Eve win in the Kobayashi-Ong game, from $S^{q,0}$, then one can construct a scheme whose OI-parallel derivation from $S^{q,0}$ is a derivation of the decorated ARS that satisfy the conditions. Therefore the value tree of the scheme is an accepting run of $\mathcal{A}$ from state q.

Remark. As we said, the proof is based on the soundness proof of Kobayashi-Ong theorem [KO09].They introduced a "process" that construct an accepting run from a positional winning strategy of Eve (this process is informally described in section 4.4.4. Here we show that this process can be formalised as a scheme, *i.e.* from a positional winning strategy of Eve, one can construct a scheme whose value tree is an accepting run of the automaton.

4.5.1. Constructing the decorated ARS

In this section, we construct a term rewriting system from $\mathcal{G}$, whose symbols are in $\Sigma_{\Theta+} \uplus \mathcal{N}_{\Theta+} \uplus \mathit{Boxes}$. As in schemes, there are no rewrite rule associated with the terminal symbols, but there are several possible rewrite rules (or none) associated with the non-terminal symbols. The TRS is not confluent, but it will be a decorated ARS in the sense that with any terms of type o, one associates a tree, and when one rewrites a term, the tree obtained extends the previous one.

Furthermore, we see that given a term $t : o$ of the initial scheme, $s : o$ a correct annotation of t, and a OI-parallel derivation from s in the TRS, the limit tree of this derivation is a run of the automaton on the value tree of t.

We start by defining the TRS $\mathcal{S}$. For all $x : \tau \in \mathcal{V}$, let $\vec{x}$ be the tuple of $\mathcal{V}_{\Theta+}^{|\tau|}$ defined by $\vec{x} = \left(x^{p_1^\tau}, \ldots, x^{p_{|\tau|}^\tau}\right)$. Recall that $p_1^\tau, \cdots, p_{|\tau|}^\tau$ is a complete ordering of $\Theta^\tau \times M$.

Definition 4.5.1 (The annotated term rewriting system $\mathcal{S}$)**.**

Let $\mathcal{S} = \langle \Sigma_{\Theta+} \uplus \mathcal{N}_{\Theta+} \uplus \mathit{Boxes}, \mathcal{V}_{\Theta+}, \mathcal{R}' \rangle$ be the term rewriting system such that:

$$\mathcal{R}' = \left\{ F^{\theta,m}\ \vec{x_1} \dots \vec{x_k} \to e' \mid F^{\theta,m} \in \mathcal{N}_{\Theta+}, e' \in \mathit{Comp}(F^{\theta,m}) \right\}.$$

In the following we are only interested in terms of $\Sigma_{\Theta+} \uplus \mathcal{N}_{\Theta+} \uplus \mathit{Boxes}$ that are coloured runs of some terms in $\mathcal{T}(\Sigma \uplus \mathcal{N})$. Therefore the following proposition shows that when applying a rewrite rule on a coloured run of a redex t, what we obtain is still a coloured run, of another term t'. Furthermore t' can be obtained from t by applying the rewrite rule of the redex.

Proposition 4.5.1.

Let $x_1 : \tau_1, \dots, x_k : \tau_k \in \mathcal{V}$, let $e : \tau \in \mathcal{T}(\Sigma \uplus \mathcal{N} \uplus \{x_1, \dots, x_k\})$, and let $t_1 : \tau_1, \dots, t_k : \tau_k$ be some terms of $\mathcal{T}(\Sigma \uplus \mathcal{N})$.
Let e' be a coloured θ-run of e and for all $x_i^{\theta',m'} : \tau_i^+$ appearing in e, let $t_i^{\theta',m'}$ be a coloured θ'-run on t_i. Then

$$e'_{\left[\forall i,\theta',m'\ x_i^{\theta',m'} \mapsto t_i^{\theta',m'}\right]}$$

is a coloured θ-run on $e_{[\forall i\ x_i \mapsto t_i]}$.

Proof. We prove this result by induction on e. Assume that $\theta = q_1 \to (\sigma_1, \dots, \sigma_k)$.

- If $e = \alpha \in \Sigma \uplus \mathcal{N}$ then $e' = \alpha^{\theta,m}$ for some m, then

  $$e'_{\left[\forall i,\theta',m'\ x_i^{\theta',m'} \mapsto t_i^{\theta',m'}\right]} = \alpha^{\theta,m}$$

  is a coloured θ-run of $e_{[\forall i\ x_i \mapsto t_i]} = \alpha$.

- if $e = x_i$ for some i then $e' = x_i^{\theta,m}$ for some m, then

  $$e'_{\left[\forall i,\theta',m'\ x_i^{\theta',m'} \mapsto t_i^{\theta',m'}\right]} = t_i^{\theta,m}$$

  is a coloured θ-annotation of $e_{[\forall i\ x_i \mapsto t_i]} = t_i$.

- if $e = e_1\ e_2$ with $e_1 : \tau' \to \tau$ and $e_2 : \tau'$, then there exists σ, e'_1, and $\vec{e'_2}$ such that:
  - $e' = e'_1 \vec{e'_2}$,
  - e'_1 is a coloured $q \to (\sigma, \sigma_1, \dots, \sigma_k)$-run annotation on e_1,

– for all $(\theta', m') \in \sigma$, ${e'}_2^{\theta',m'}$ is a coloured θ'-run on e_2,

– for all $(\theta', m') \notin \sigma$, ${e'}_2^{\theta',m'} = \Box^{\tau'^+}$.

Then by induction hypothesis, the following holds.

– The term:

$$e'_1\left[\forall i,\theta',m'\ x_i^{\theta',m'} \mapsto t_i^{\theta',m'}\right]$$

is a coloured $q \to (\sigma, \sigma_1, \ldots, \sigma_k)$-run on $e_{[\forall i\ x_i \mapsto t_i]}$.

– For all $(\theta', m') \in \sigma$, the term

$${e'}_2^{\theta',m'}\left[\forall i,\theta',m'\ x_i^{\theta',m'} \mapsto t_i^{\theta',m'}\right]$$

is a coloured θ'-run of $e_{2[\forall i\ x_i \mapsto t_i]}$.

Therefore

$$e'\left[\forall i,\theta',m'\ x_i^{\theta',m'} \mapsto t_i^{\theta',m'}\right]$$

is a coloured θ-run on $e_{[\forall i\ x_i \mapsto t_i]}$.

□

The next proposition extends the previous one, by showing that when applying an OI-parallel rewrite to a coloured run s of a term t, we obtain the term s' which is a coloured run of the term t' obtained by applying an OI-parallel rewrite to t. This, however is not true for any coloured run, but it is true when s is *non-blocking* as defined below. Basically a term is non-blocking if all annotated non-terminals that may be rewritten during an OI parallel rewrite, have a compatible term, *i.e.* are actually rewritten if one rewrites the term.

Definition 4.5.2 (Non-blocking term)**.**

The set of ***non-blocking*** coloured runs of type o is inductively defined as follows.

- If $s = F^{\theta,m}\ \vec{t_1} \ldots \vec{t_k}$ with $F \in \mathcal{N}$, s is non-blocking if $Comp(F^{\theta,m}) \neq \emptyset$.
- If $s = a^{\theta,m}\ \vec{t_1} \ldots \vec{t_k}$ with $a \in \Sigma$, s is non-blocking if for all i, θ', m' either $t_i^{\theta',m'} = \Box^o$, or $t_i^{\theta',m'}$ is non blocking.

One can state the proposition establishing a correspondence between the OI parallel rewrite of $\mathbb{S}$ and $\mathcal{G}$.

Proposition 4.5.2.

Let $t : o \in \mathcal{T}(\Sigma \uplus \mathcal{N})$ and t' the term defined by $t \Rightarrow_{\mathrm{OI},\mathcal{G}} t'$. Let s be a non-blocking coloured q-run on t for some state q. Then for all s' such that $s \Rightarrow_{\mathrm{OI},\mathcal{S}} s'$, s' is a coloured q-run on t'.

Proof. Straightforward induction on the structure of t. □

Remark that the coloured q-run s' is not necessary non-blocking, even though s is.

Now we extract a decorated ARS from $\mathcal{S}$. The objects are the coloured runs of terms in $\mathcal{T}(\Sigma \uplus \mathcal{N})$, the reduction relation is the OI-parallel rewrite, and the partial mapping is the ▲-transformation on ground typed terms, as defined below.

Let $t : o \in \mathcal{T}(\Sigma \uplus \mathcal{N})$ and s a correct annotation of t. We define the ▲-transformation of s as a special kind of $\bot$-transformation: it removes the non-terminal symbols of s and it turns what remains into a finite tree over the ranked alphabet Σ_{Θ_s}.

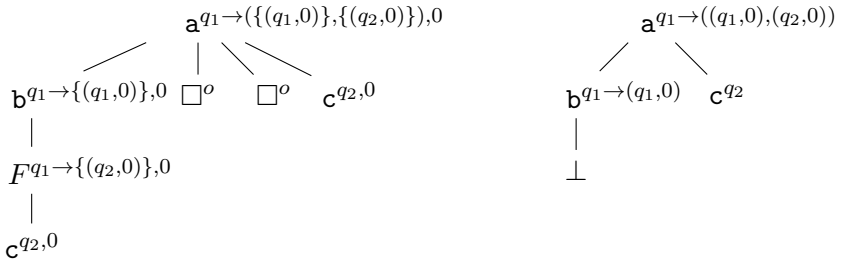
Definition 4.5.3 (The ▲-transformation)**.**

Let $t : o \in \mathcal{T}(\Sigma \uplus \mathcal{N})$ and s a coloured run on t. The tree $s^{\blacktriangle}$ is inductively defined as follows.

- If $s = F^{\theta,m}\ \vec{t_1} \dots \vec{t_k}$ with $F^{(\theta,m)} \in \mathcal{N}_{\Theta+}$, then $s^{\blacktriangle} = \bot$.
- If $s = a^{q \to (\{(q_1,m_1)\},\dots,\{(q_k,m_k)\}),m}\ \vec{t_1} \dots \vec{t_k}$, then

$$s^{\blacktriangle} = a^{q \to ((q_1,m_1),\dots,(q_k,m_k))}\ t_1^{(q_1,m_1)^{\blacktriangle}} \dots t_k^{(q_k,m_k)^{\blacktriangle}}.$$

Example 4.5.1. Let $M = \{0\}$, $Q = \{q_1, q_2\}$, $\Sigma = \{\mathtt{a} : o^2 \to o, \mathtt{b} : o \to o, \mathtt{c} : o\}$ and $\mathcal{N} = \{F : o \to o\}$. Let t be the term $t = \mathtt{a}\ (\mathtt{b}\ (F\ \mathtt{c}))\ \mathtt{c}$. We depict below a q_1-run s of t along with the tree $s^{\blacktriangle}$.



The q_1-run s The tree $s^{\blacktriangle}$

In order to show that it is a decorated ARS, we need to have that when applying the reduction relation, the associated trees are extended.

Proposition 4.5.3.

Let $t : o \in \mathcal{T}(\Sigma \uplus \mathcal{N})$ and s be a coloured q-run on t. Let s' be such that $s \Rightarrow_{\mathrm{OI},\mathcal{S}} s'$. Then $s^{\blacktriangle} \sqsubseteq s'^{\blacktriangle}$.

Proof. Straightforward induction on the structure of t. □

Therefore we obtain a decorated ARS $\langle \mathcal{A}^{+}(\Sigma \uplus \mathcal{N}), \Rightarrow_{\mathrm{OI}}, f_{\blacktriangle} \rangle$ where $f_{\blacktriangle}$ is the mapping defined on terms of type o by $f_{\blacktriangle}(s) = s^{\blacktriangle}$.

Let $s_0 \Rightarrow_{\mathrm{OI}} s_1 \Rightarrow_{\mathrm{OI}} s_2 \Rightarrow_{\mathrm{OI}} \cdots$ be a parallel derivation in $\mathcal{S}$. We say that this derivation is ***non-blocking*** if for all i, s_i is non-blocking.

Proposition 4.5.4.

Let $t : o \in \mathcal{T}(\Sigma \uplus \mathcal{N})$ and s a correct annotation of t from state q. Let $s \Rightarrow_{\mathrm{OI}} s_1 \Rightarrow_{\mathrm{OI}} s_2 \Rightarrow_{\mathrm{OI}} \cdots$ be a non-blocking parallel derivation from s. Then the limit R of the sequence $s^{\blacktriangle} \sqsubseteq s_1^{\blacktriangle} \sqsubseteq \cdots$, is a run of $\mathcal{A}$ on $\|t\|_{\mathcal{G}}$ from state q.

Proof. Let $t \Rightarrow_{\mathrm{OI}} t_1 \Rightarrow_{\mathrm{OI}} \cdots$ be the parallel derivation from t in $\mathcal{G}$. Recall that we assume that $\|t\|_{\mathcal{G}}$ do not contains $\bot$. We write $T = \|t\|_{\mathcal{G}}$.

First remark that $dom(s_i^{\blacktriangle}) = dom(t_i^{\bot})$. Then for all $u \in dom(t_i^{\bot})$, if $t_i^{\bot} = a \in \mathcal{N}$, $s_i{}^{\blacktriangle} = a^{\theta}$ for some $\theta \in \Theta_s$. Therefore $dom(R) = dom(T)$ and for all $u \in dom(T)$, let $a : o^k \to o \in \Sigma$ such that $T(u) = a$. Then $R(u) = a^{\theta}$ for some $a \in \Theta_s$.

Furthermore there exists i such that for all $j \leq k$, $u \cdot j \in dom(t_i^{\bot})$ and $t_i^{\bot}(u \cdot j) = T(u \cdot j) = b_j^{\theta_j}$ for some $b_j \in \Sigma$ and $\theta_j \in \Theta_s$. Therefore $source(\theta_j) = dst_j(\theta)$.

Then R is a run of $\mathcal{A}$ over T from state q. □

4.5.2. A winning condition for derivations

This section constitutes the technical core of the proof of the selection. It uses some steps of Kobayashi and Ong proof for model checking, but not in the same formalism. Given an OI parallel derivation d in the TRS that produces a run of the scheme, we give a sufficient condition on d for the run to be accepting. More precisely this condition is a requirement on some infinite sequences of occurrences in the derivation.

Therefore, we define the set of sequences of occurrences that we are interested in. They are the infinite sequences linked with the creation relation, *i.e.* sequences of the form $occ_0 \succ occ_1 \succ occ_2 \succ \cdots$.

Definition 4.5.4 (Creation sequences)**.**

Let t be the term $t = S$, and s_0 a correct annotation of t, *i.e.* $s_0 = S^{q,m}$ for some $q \in Q$ and $m \in M$. Let d be an OI-parallel derivation from s_0: $d = s_0 \Rightarrow_{\mathrm{OI}} s_1 \Rightarrow_{\mathrm{OI}} s_2 \Rightarrow_{\mathrm{OI}} \cdots$.
A ***creation sequence*** in d, is an infinite sequence of occurrences $occ_0, occ_1, occ_2, \ldots$ such that:

- occ_0 is the only occurrence of s_0, *i.e.* $occ_0 = (0, (\bullet, S^{q,m}))$,
- for all i, $occ_i \succ occ_{i+1}$.

Observe that the symbol of each occurrence is an annotated non terminal $F^{\theta,m}$. Therefore we define the colour of a creation sequence as the maximum colour annotating infinitely often the occurrences of the sequence.

Definition 4.5.5 (Colour of a creation sequence)**.**

Let t be the term $t = S$, and s_0 a correct annotation of t, *i.e.* $s_0 = S^{q,m}$ for some $q \in Q$ and $m \in M$. Let d be an OI-parallel derivation from s_0 and let $occ_0, occ_1, occ_2, \ldots$ be a creation sequence in d.
For all i, let $F_i^{\theta_i,m_i}$ be the symbol associated with occ_i. The ***colour*** of the sequence is the greatest colour appearing infinitely often in $m_0, m_1, m_2, \ldots$.

We are now in position to state the sufficient condition on a derivation in order to produce an accepting run. The rest of this section consists in a proof of this theorem.

Theorem 4.5.5.

Let t be the term $t = S$, and s_0 a correct annotation of t, *i.e.* $s_0 = S^{q,m}$ for some $q \in Q$ and $m \in M$. Let d be a non-blocking OI-parallel derivation from s_0.
If for every creation sequence *seq* in d, *colour*(*seq*) is even, then the limit tree of d is an accepting run of $\mathcal{A}$ on $\|\mathcal{G}\|$ from state q.

The proof proceeds in two steps. First we associate with every infinite branch b of the run produced by the derivation, a creation sequence *seq*. Then we show that the maximum colour appearing infinitely often along b, is equal to the colour of *seq*.

Remark that in order for any occurrence of a creation sequence to create other occurrences, it must be in a position in the term such that it may be rewritten in an OI-rewrite. We give a characterisation of such occurrence. Given a correct annotation $s : o$ of some terms t, an OI-*occurrence* of s is an occurrence of a symbol in $\Sigma_{\Theta+} \cup \mathcal{N}_{\Theta+}$ such that all symbols above it are in $\Sigma_{\Theta+}$.

Definition 4.5.6 (OI-occurrence)**.**

Let $s : o$ be a correct annotation of some term $t : o$. The set of OI-occurrences of s is inductively defined as follows.

- If $s = \alpha^{\theta,m}\ \vec{t_1} \dots \vec{t_k} : o$ with $\alpha^{\theta,m} \in \Sigma_{\Theta+} \cup \mathcal{N}_{\Theta+}$ then

$$\left(\bullet\ \vec{t_1} \dots \vec{t_k}, \alpha^{\theta,m}\right)$$

  is an OI-occurrence of s.

- If $s = a^{q \to (\{(q_1,m_1)\},\dots,\{(q_k,m_k)\}),m_0}\ \vec{t_1} \dots \vec{t_k}$ with $a^{q \to (\{(q_1,m_1)\},\dots,\{(q_k,m_k)\}),m} \in \Sigma_{\Theta+}$. for all $i \leq k$, and for all OI-occurrence $(C'[\bullet], \alpha^{\theta,m})$ of $t_i^{(q_i,m_i)}$, let

$$C[\bullet] = a^{q_0 \to (\{(q_1,m_1)\},\dots,\{(q_k,m_k)\}),m_0}\ \vec{t'_1} \dots \vec{t'_k}$$

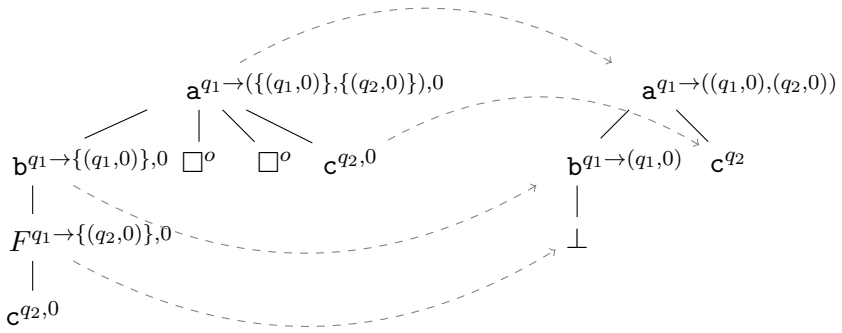
  Such that $t'^{(q_i,m_i)}_i = C'[\bullet]$, and $t'^{(q',m')}_j = t_j^{(q',m')}$ for $j \neq i$ or $(q',m') \neq (q_i,m_i)$. Then$(C[\bullet], \alpha^{\theta,m})$ is an OI-occurrence of s.

Given a derivation $d = s_0 \Rightarrow_{\text{OI}} s_1 \Rightarrow_{\text{OI}} \cdots$ and an occurrence (i, occ) of d, then (i, occ) is an OI-***occurrence of*** d, if occ is an OI-occurrence of s_i.

As we stated before, observe that during an OI parallel derivation from a ground type term, if an occurrence occ creates another one, then it is an OI-occurrence.

To any OI-occurrence in a correct annotation $s : o$ of some terms, we associate a node in $s^{\blacktriangle}$, that corresponds to the location of the occurrence in the tree. This association matches the intuitive correspondence between occurrences in a term and node in the tree obtained by $\blacktriangle$-transformation, as depicted in the example below.

Example 4.5.1 continued. We show the correspondence between OI-occurrences of s and nodes of $s^{\blacktriangle}$, depicted in grey dashed arrows.



The q_1-annotation s The tree $s^{\blacktriangle}$

Definition 4.5.7 (Node of an OI-occurrence)**.**

Let $s = \beta^{\theta_0,m_0}\ \vec{t_1} \dots \vec{t_\ell} : o$ be a correct annotation, and $occ = (\alpha^{\theta,m}, C[\bullet])$ be an OI-occurrence in s. The ***node of*** $C[\bullet]$, $node(C[\bullet])$ is inductively defined as follows.

- If $C[\bullet] = \bullet\ \vec{t_1} \dots \vec{t_\ell}$, $node(occ) = \varepsilon$.
- If there exists an occurrence $occ' = (\alpha^{\theta,m}, C'[\bullet])$ of $t_i^{(q_i,m_i)}$ such that $C[\bullet] = \beta^{\theta_0,m_0}\ \vec{t'_1} \dots \vec{t'_\ell}$ with $t'^{(q_1,m_i)}_i = C'[\bullet]$ then $node(occ) = i \cdot node(occ')$.

Remark that the mapping *node* is a bijection between the set of occurrences of s and the set of nodes in $s^{\blacktriangle}$, and for all OI-occurrence $occ = (\alpha^{\theta,m}, C[\bullet])$ in a correct annotation $s : o$ of some term t, if $u = node(occ)$ then:

- if $\alpha^{\theta,m} = a^{q\to(\{(q_1,m_1)\},\dots\{(q_k,m_k)\}),m} \in \Sigma_{\Theta+}$ then $s^{\blacktriangle}(u) = a^{q\to((q_1,m_1),\dots(q_k,m_k))}$,
- if $\alpha^{\theta,m} \in \mathcal{N}_{\Theta+}$, then $s^{\blacktriangle}(u) = \bot$.

We are now in position to associate with every branch of the tree produced by a derivation a creation sequence. The following result is proven in [KO12].

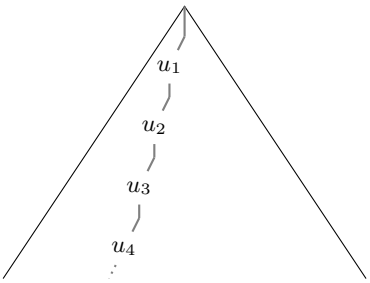
Theorem 4.5.6.

Let $S^{q,0} \Rightarrow_{\mathrm{OI}} s_1 \Rightarrow_{\mathrm{OI}} s_2 \Rightarrow_{\mathrm{OI}} \cdots$ be a parallel derivation. Let b be a branch of the tree associated with this derivation. Then there exists a creation sequence seq_b,

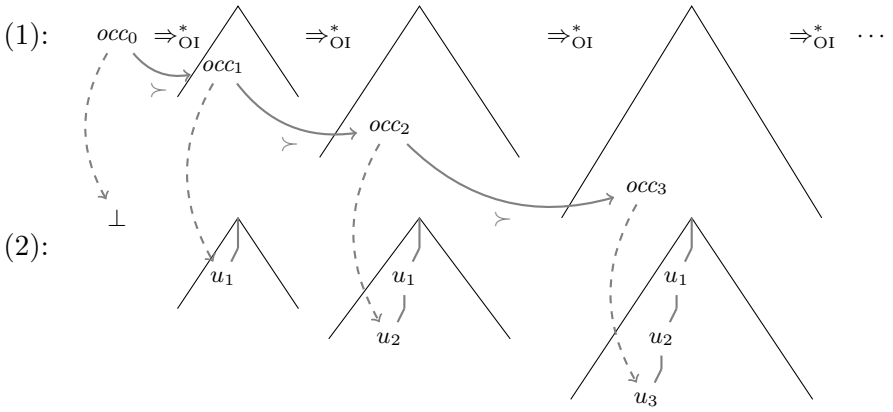
$$seq_b \quad = \quad occ_0 \succ occ_1 \succ occ_2 \succ \cdots,$$

Such that $\lim_i node(occ_i) = b$.

Let us illustrate this theorem. Let d be the an OI-parallel derivation $d = S^{q,0} \Rightarrow_{\mathrm{OI}} s_1 \Rightarrow_{\mathrm{OI}} s_2 \Rightarrow_{\mathrm{OI}} \cdots$. Let R be the associated tree of d and b an infinite branch of R. Then there exists a sequence of nodes $u_1, u_2, \ldots$ along this branch (in grey in the illustration):



and a creation sequence $occ_0 \succ occ_1 \succ \cdots$ in the derivation, such that for all $i > 0$, $node(occ_i) = u_i$. We illustrate this below, terms of the derivation are represented (1) along with their ▲-transformations (2). The function *node* is represented by grey dashed arrows between terms and their associated trees.



The next part of the section consists in studying how colours behave between occurrences, and between nodes of these occurrences. We first prove a lemma that states that substituting a variable with a term in a context does not affect the colour of the context.

Lemma 4.5.7.

Let $C[\bullet]$ be a correct context in $CC^+(\Sigma \uplus \mathcal{V}_{\Theta+})$. Let $x^{\theta,m} : \tau$ in $\mathcal{V}_{\Theta+}$, and $s : \tau$ be a θ-run of some term.
Then $colour(C[\bullet]) = colour(C[\bullet]_{[x^{\theta,m} \mapsto s]})$.

Proof. We prove this result by induction on $C[\bullet]$. Assume that $\theta = q \to (\sigma_1, \ldots, \sigma_k)$ with $k \geq j$. The only interesting case is when

$$C[\bullet] = x^{\theta,m}\ \vec{t_1} \ldots \vec{t_j},$$

such that there exists i and $(\theta, m') \in \sigma_i$ satisfying $t_i^{(\theta,m')} = C'[\bullet]$ for some context $C'[\bullet]$. We have $\mathit{colour}(C[\bullet]) = \max\big(m', \mathit{colour}(C'[\bullet])\big)$.

Since s is a θ-annotation of some term, there exists $\sigma'_1, \ldots, \sigma'_\ell$, $\alpha \in \Sigma \uplus \mathcal{N}$, $m'' \in M$ and $\vec{s_1}, \ldots, \vec{s_\ell}$ such that:

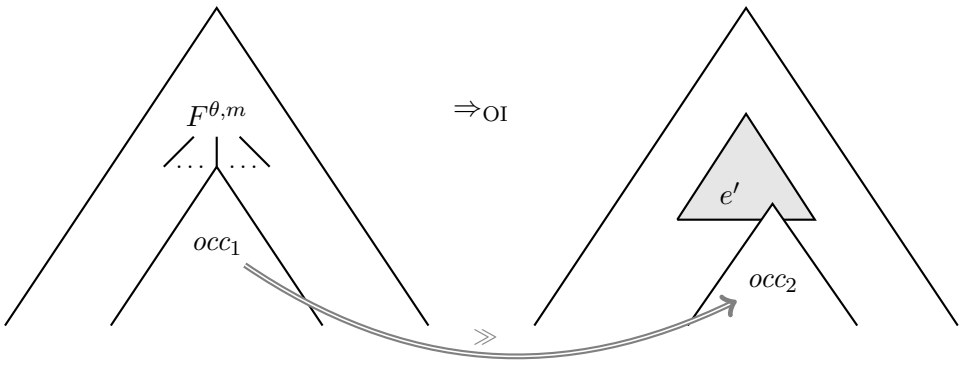
$$s = \alpha^{q \to (\sigma'_1, \ldots, \sigma'_\ell, \sigma_1, \ldots, \sigma_k), m''}\ \vec{s_1} \ldots \vec{s_\ell}.$$

We have $C[\bullet]_{[x^{\theta,m} \mapsto s]} = s\ \vec{t_1}_{[x^{\theta,m} \mapsto s]} \ldots \vec{t_k}_{[x^{\theta,m} \mapsto s]}$. Therefore

$$\mathit{colour}(C[\bullet]_{[x^{\theta,m} \mapsto s]}) = \max\big(m', \mathit{colour}(C'[\bullet]_{[x^{\theta,m} \mapsto s]})\big).$$

By induction hypothesis, $\mathit{colour}(C'[\bullet]_{[x^{\theta,m} \mapsto s]}) = \mathit{colour}(C'[\bullet])$, therefore $\mathit{colour}(C[\bullet]_{[x^{\theta,m} \mapsto s]}) = \mathit{colour}(C[\bullet])$. □

The next proposition shows that the well-colouredness of an occurrence is preserved by the conservation relation. More precisely, given two θ-runs s and s' such that $s \Rightarrow_{\mathrm{OI}} s'$, and given two occurrences occ_1 in s and occ_2 in s', such that $\mathit{occ}_1 \gg \mathit{occ}_2$. For instance assume that occ_1 is in a subterm of a redex as depicted below (here we assume that $e' \in \mathit{Comp}(F^{\theta,m})$).



Then $\mathit{colour}(\mathit{occ}_1) = \mathit{colour}(\mathit{occ}_2)$, *i.e.* if occ_1 is well-coloured then occ_2 is well-coloured too.

Proposition 4.5.8 (Colours through the conservation relation)**.**

Let $t : o \in \mathcal{T}(\Sigma \uplus \mathcal{N})$, s be a q-annotation of t for some state q and s' be such that $s \Rightarrow_{\mathrm{OI}} s'$. Let occ_1 be a well-coloured occurrence in s and occ_2 be an occurrence in s' such that $\mathit{occ}_1 \gg \mathit{occ}_2$. Then occ_2 is well-coloured.

Proof.

- If $s = F^{\theta,m}\ \vec{t_1} \dots \vec{t_k}$ with $F \in \mathcal{N}$ and $s' = e'_{[\forall i\ x_i \mapsto t_i]}$ with $e' \in \mathit{Comp}(F^{\theta,m})$. Then there exists $i \leq k$, (θ_i, m_i), and two contexts $C_i[\bullet]$ and $C_e[\bullet]$ such that, if we let $\vec{t'_i}[\bullet]$ be equal to $\vec{t_i}$ except for $t'^{\theta_i,m_i}_i[\bullet] = C_i[\bullet]$,

$$\mathit{occ}_1 = \left(F^{\theta,m}\ \vec{t_1} \dots \vec{t'_i}[\bullet] \dots \vec{t_k}\ ,\ \beta\right)$$

$$\mathit{occ}_2 = \left(C_e\left[C_i[\bullet]\right]_{\left[\forall i,\theta',m'\ x_i^{\theta',m'} \mapsto t_i^{\theta',m'}\right]}, \beta\right),$$

$$e' = C_e[x_i^{\theta_i,m_i}].$$

  We have $\mathit{colour}(\mathit{occ}_1) = \max(m_i, \mathit{colour}(C_i[\bullet]))$ and $\mathit{colour}(\mathit{occ}_2) = \max(\mathit{colour}(C_e[\bullet]_{\left[\forall i,\theta',m'\ x_i^{\theta',m'} \mapsto t_i^{\theta',m'}\right]}), \mathit{colour}(C_i[\bullet]))$. Due to the previous lemma, $\mathit{colour}(C_e[\bullet]_{\left[\forall i,\theta',m'\ x_i^{\theta',m'} \mapsto t_i^{\theta',m'}\right]}) = \mathit{colour}(C_e[\bullet])$. Furthermore, since e' is well-coloured, and $e' = C_e[x_i^{\theta_i,m_i}]$, $\mathit{colour}(C_e[\bullet]) = m_i$. Therefore $\mathit{colour}(\mathit{occ}_1) = \mathit{colour}(\mathit{occ}_2)$. In particular, if occ_1 is well-coloured, occ_2 is well-coloured too.

- If $s = a^{\theta,m}\ \vec{t_1} \dots \vec{t_k}$ with $a \in \Sigma$. If $\mathit{occ}_1 = (a^{\theta,m}, \bullet\ t_1 \dots t_k)$, $\mathit{colour}(\mathit{occ}_1) = 0$. Furthermore $s' = a^{\theta,m}\ \vec{t'_1} \dots \vec{t'_k}$, such that for all i, θ', m', $t_i^{\theta',m'} \Rightarrow_{\mathrm{OI}} t'^{\theta',m'}_i$. Therefore $\mathit{occ}_2 = (a^{\theta,m}, \bullet\ t_1 \dots t_k)$ and $\mathit{colour}(\mathit{occ}_2) = 0$. If occ_1 is well-coloured, then $m = 0$ and occ_2 is well-coloured too. For any other occurrence of s, the induction hypothesis concludes.

□

Now we see that if we rewrite a redex, all occurrences created during this rewriting are well coloured. This is a direct consequence of the requirement that the production of the rewrite rules in the TRS are well coloured.

Proposition 4.5.9 (Colours through the creation relation)**.**

Let $s = F^{\theta,m}\ \vec{t_1} \dots \vec{t_k}$ be a redex, $e' \in \mathit{Comp}(F^{\theta,m})$ and $s' = e'_{\left[\forall i,\theta',m'\ x_i^{\theta',m'} \mapsto t_i^{\theta',m'}\right]}$. Then any occurrence occ' in s' that has been created by $\mathit{occ} = (F^{\theta,m}, \bullet\ \vec{t_1} \dots \vec{t_k})$ in s is well-coloured.

Proof. By definition of the creation relation, there exists a context $C_{e'}[\bullet]$ such that if $\mathit{occ}' = (\alpha^{\theta_0,m_0}, C[\bullet])$ then $C[\bullet] = C_{e'}[\bullet]_{\left[\forall i,\theta',m'\ x_i^{\theta',m'} \mapsto t_i^{\theta',m'}\right]}$, and such that $e' = C_{e'}[\alpha^{\theta_0,m_0}]$. Since e' is well-coloured, $\mathit{colour}(C_{e'}[\bullet]) = m_0$. Furthermore, thanks to Lemma **4.5.7**, $\mathit{colour}(C[\bullet]) = \mathit{colour}(C_{e'})$. Therefore occ' is well coloured. □

The following is a direct corollary of the two previous results.

Corollary 4.5.10.

Let $s = F^{\theta,m}\ \vec{t_1} \dots \vec{t_k}$ be a redex, d an IO parallel derivation from s, and $occ = (0, (F^{\theta,m}, \bullet\ \vec{t_1} \dots \vec{t_k}))$.
Then any occurrence created by occ is well-coloured.

The next proposition state that if an occurrence creates another one in an OI parallel derivation, then the node of the second occurrence is below the node of the first one. It is an easy result since when an occurrence creates another, it is the head of a rewritten redex and then all symbols above it are terminals symbols.

Proposition 4.5.11.

Let $t : o \in \mathcal{T}(\Sigma \uplus \mathcal{N})$ and s a correct annotation of t from state q. Let $s \Rightarrow_{\text{OI}} s_1 \Rightarrow_{\text{OI}} s_2 \Rightarrow_{\text{OI}} \cdots$ be a parallel derivation from s. And let $occ_i \succ occ_j$ be two OI-occurrences of the derivation, then $node(occ_i) \leq node(occ_j)$.

Proof. Straightforward induction on the context of occ_i. □

Given a tree T on Σ_{Θ_s}, and two nodes $u \leq v$ in $dom(T)$. The ***colour between*** u ***and*** v ***in*** T is inductively defined as follows.

- If $u = v$ then $colour(T, u, v) = 0$.
- If $u = i \cdot u'$ with $u' \leq v$, let $a^{q \to ((q_1,m_1),\dots(q_k,m_k))} = s^{\blacktriangle}(u)$. Then $colour(T, u, v) = \max(m_i, colour(T, u', v))$.

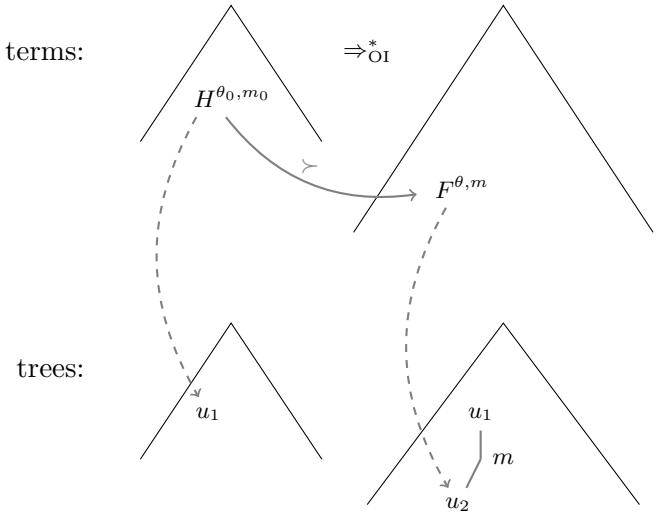
Remark that if $T \sqsubseteq T'$ then for all $u \leq v$ in $dom(T)$, $colour(T, u, v) = colour(T', u, v)$.
Now we establish the equality between the colour of an OI-occurrence in a term s and the colour between the root and its node in the tree $s^{\blacktriangle}$.

Proposition 4.5.12.

Let $s : o$ be a correct annotation and occ be an OI-occurrence in s. Then $colour(occ) = colour(s^{\blacktriangle}, \varepsilon, node(occ))$.

Proof. Straightforward induction on the context of occ. □

The next proposition establishes the link between colour annotations on the symbols during a derivation, and actual colours between nodes in the tree produced during a derivation. It states that given two OI-occurrences occ_1, occ_2 in a derivation such that occ_1 creates occ_2, if the symbol of occ_2 is annotated with the colour m, then, the colour between the nodes of the two occurrence is equal to m, as depicted below.



Proposition 4.5.13.

Let $t : o \in \mathcal{T}(\Sigma \uplus \mathcal{N})$ and s a correct annotation of t from state q. Let $s \Rightarrow_{\mathrm{OI}} s_1 \Rightarrow_{\mathrm{OI}} s_2 \Rightarrow_{\mathrm{OI}} \cdots$ be a parallel derivation from s. And let $occ_i \succ occ_j$ be two occurrences of the derivation such that $occ_j = (j, (C[\bullet], F^{\theta,m}))$ is an OI-occurrence of s_j. Then $\mathit{colour}(s_j{}^{\blacktriangle}, \mathit{node}(occ_i), \mathit{node}(occ_j)) = m$.

Proof. The proof proceeds by induction of $\mathit{node}(occ_i)$. If $\mathit{node}(occ_i) = \varepsilon$ then $s_i = F^{\theta,m}\ \vec{t_1} \dots \vec{t_k}$ is a redex and $occ_i = (i, (F^{\theta,m}, \bullet\ \vec{t_1} \dots \vec{t_k}))$. Therefore Corollary **4.5.10** states that occ_j is well-coloured, and Proposition **4.5.12** states that $\mathit{colour}(occ_j) = \mathit{colour}(s_j{}^{\blacktriangle}, \varepsilon, \mathit{node}(occ_j))$, therefore $\mathit{colour}(s_j{}^{\blacktriangle}, \mathit{node}(occ_i), \mathit{node}(occ_j)) = m$.

If $\mathit{node}(occ_i) = \ell \cdot u'$. Then for all $n \geq i$, $s_n = a^{\theta,m}\ \vec{s_{n,1}} \dots \vec{s_{n,k}}$ for some $a \in \Sigma$. Let $q \to (\{(q_1, m_1)\}, \dots, \{(q_k, m_k)\}) = \theta$. Let d' be the derivation:

$$s_{i,\ell}^{q_\ell,m_\ell} \Rightarrow_{\mathrm{OI}} s_{i+1,\ell}^{q_\ell,m_\ell} \Rightarrow_{\mathrm{OI}} \cdots .$$

Let $C_i[\bullet]$ be the context associated with occ_i, *i.e.* $occ_i = (H^{\theta_0,m_0}, C_i[\bullet])$. Since $\mathit{node}(occ_i) = \ell \cdot u'$,

$$C_i[\bullet] = a^{\theta,m}\ \vec{s'_{i,1}} \dots \vec{s'_{i,k}},$$

such that $s'^{q_\ell,m_\ell}_{i,\ell} = C'[\bullet]$ for some a context $C'[\bullet]$ and if $(\ell', q', m') \neq (\ell, q_\ell, m_\ell)$, then $s'^{q',m'}_{i,\ell'} = s^{q',m'}_{i,\ell'}$.

Let occ'_i be the occurrence of d' defined by $occ'_i = (0, (H^{\theta_0,m_0}, C'[\bullet]))$. We know that $node(occ_i) \leq node(occ_j)$ therefore there exists v' be such that $node(occ_i) = \ell \cdot v'$. As for occ'_i, we define the occurrence occ'_j in d' appearing in $s_{j,\ell}^{q_\ell,m_\ell}$ that corresponds to occ_j. A simple induction on $j - i$ shows that $occ'_i \succ occ'_j$. Furthermore, $node(occ'_i) = u'$ and $node(occ'_i) = v'$.

By induction hypothesis, $colour((s_j, \ell^{q_\ell,m_\ell})^\blacktriangle, node(occ'_i), node(occ'_j)) = m$. Therefore, since $(s_j, \ell^{q_\ell,m_\ell})^\blacktriangle$ is the ℓ-th son of node ε in $s_j{}^\blacktriangle$, $colour(s_j{}^\blacktriangle, node(occ_i), node(occ_j)) = m$. □

The next proposition concludes the proof of Theorem **4.5.5**. Indeed it states that the colour of a creation sequence associated with a branch is equal to the colour of the branch. Therefore if the colour of every creation sequence is even, then the colour of every infinite branch is be even.

Proposition 4.5.14.

Let $S^{q,0} \Rightarrow_{\mathrm{OI}} s_1 \Rightarrow_{\mathrm{OI}} s_2 \Rightarrow_{\mathrm{OI}} \cdots$ be a parallel derivation. Let b be a branch of the tree associated with this derivation and seq_b, be the creation sequence associated with b.
Then $colour(b) = colour(seq_b)$.

Proof. Direct application of Proposition **4.5.13**. □

4.5.3. Solving the selection problem

We introduce *run-constructing schemes*, as schemes whose symbols are annotated symbols of $\mathcal{G}$, and whose value tree, as we see in the next proposition, is a run of $\mathcal{A}$ on the value tree of $\mathcal{G}$.

$\mathcal{G}' = \langle \Sigma', \mathcal{N}', \mathcal{V}', S', \mathcal{R}' \rangle$ is a ***run-constructing scheme from state*** q for $\mathcal{G}$, if:

- $\Sigma' = \Sigma_{\Theta_s}$,
- $\mathcal{N}' \subseteq \Sigma_{\Theta+} \cup \mathcal{N}_{\Theta+} \cup Boxes$,
- $\mathcal{V}'$ contains $\mathcal{V}_{\Theta+}$ (for rules on $\mathcal{N}_{\Theta+}$), $\{\eta_i^{\theta,m} : o \mid i \leq arity_{\max}, \theta \in \Theta+, m \in M\}$ (for rules on $\Sigma_{\Theta+}$), and $\{y_i^\tau : \tau_i \mid \Box^\tau \in \mathcal{N}', \tau = \tau_1 \to \cdots \tau_k \to o, i \leq k\}$ (for rules on $Boxes$).
- $S' = S^{q,0}$,
- For all $a^{q \to (\{(q_1,m_1)\},\ldots,\{(q_k,m_k)\}),m} \in \Sigma_{\Theta+} \cap \mathcal{N}'$

$$a^{q \to (\{(q_1,m_1)\},\ldots,\{(q_k,m_k)\}),m} \vec{\eta_1} \ldots \vec{\eta_k} \quad \to \quad a^{q \to ((q_1,m_1),\ldots,(q_k,m_k))}\ \eta_1^{(q_1,m_1)} \ldots \eta_k^{(q_k,m_k)} \in \mathcal{R}'.$$

For all $F^{\theta,m} \in \mathcal{N}_{\Theta+} \cap \mathcal{N}'$ there exists $e' Comp(F^{\theta,m})$ such that, e' conforms to $\mathcal{N}' \cap \mathcal{N}_{\Theta+}$ on $\mathcal{N}$, and

$$F^{\theta,m}\ \vec{x_1} \dots \vec{x_k} \to e'\ \in \mathcal{R}'.$$

For all $\square^\tau \in \mathcal{N}'$,

$$\square^\tau\ y_1^\tau \dots y_k^\tau \to \square^\tau\ y_1^\tau \dots y_k^\tau.$$

Note that there does not necessarily exists a run-constructing scheme. Indeed one may not be able to find a subset of $\mathcal{N}_{\Theta+}$ such that for all $F^{\theta,m}$ in that subset $Comp(F^{\theta,m})$ is non-empty and furthermore there exists a term in $Comp(F^{\theta,m})$ such that all annotated non-terminal symbols occurring in this term also are in the subset.

The next proposition states that, unsurprisingly, a run-constructing scheme constructs a run of the automaton.

Proposition 4.5.15.

Given $\mathcal{G}'$ a run-constructing scheme from state q for $\mathcal{G}$. Then $\|\mathcal{G}'\|$ is a run of $\mathcal{A}$ over $\|\mathcal{G}\|$ from state q.

Proof. In order to prove this proposition, we associate with $\mathcal{G}$ a non-blocking OI-parallel derivation d of the TRS $\mathcal{S}$, and we show that the value tree of $\mathcal{G}$ is equal to the run generated by d.

First, for all term $t : o \in \mathcal{T}(\mathcal{N}')$, the term $f(t)$ is defined by induction on the structure of t as follows.

- If $t = a^{\theta,m}\ \vec{t_1} \dots \vec{t_k}$ with $a^{\theta,m} \in \Sigma_{\Theta+}$, then $f(t) = a^{\theta,m}\ \vec{t'_1} \dots \vec{t'_k}$ where for all i, θ', m', $t'^{\theta',m'}_i = f(t_i^{\theta',m'})$.
- If $t = F^{\theta,m}\ \vec{t_1} \dots \vec{t_k}$ with $a^{\theta,m} \in \mathcal{N}_{\Theta+}$ then, if $F^{\theta,m}\ \vec{x_1} \dots \vec{x_k} \to e \in \mathcal{R}'$, then

$$f(t) = e_{\left[\forall i,\theta',m'\ x_i^{\theta',m'} \mapsto t_i^{\theta',m'}\right]}.$$

- If $t = \square^o$, $f(t) = \square^o$.

Notice that $t \to^*_{\mathcal{G}'} f(t)$ and furthermore, $t \Rightarrow_{\mathrm{OI}} f(t)$ in the term rewriting system $\mathbb{S}$.

We define a sequence of ground typed terms $s_0, s_1, s_2, \dots$ in $\mathcal{T}(\mathcal{N}')$ such that for all i, $s_i \to^*_{\mathcal{G}'} s_{i+1}$ and $s_i \Rightarrow_{\mathrm{OI}} s_{i+1}$ in $\mathbb{S}$. The term s_0 is defined by $s_0 = S^{q,0}$ and s_{i+1} is define as $s_{i+1} = f(s_i)$.

For all i, we have $(s_i)^\perp = \perp$. Therefore we will show that there exists s'_i such that $s_i \to^*_{\mathcal{G}'} s'_i$, and $(s'_i)^\perp = (s_i)^\blacktriangle$. We define s'_i as $s'_i = g(s_i)$ where g is defined as follows.

for all term $t : o \in \mathcal{T}(\mathcal{N}')$, such that $t \neq \square^o$, the term $g(t)$ is defined by induction on the structure of t as follows.

- If
$$t = a^{q\to(\{(q_1,m_1)\},\ldots,\{(q_k,m_k)\}),m}\ \vec{t_1}\ldots\vec{t_k}$$
with $a^{q\to(\{(q_1,m_1)\},\ldots,\{(q_k,m_k)\}),m} \in \Sigma_{\Theta+}$, then
$$g(t) = a^{q\to((q_1,m_1),\ldots,(q_k,m_k))}\ 1^{q_1,m_1}\ldots k^{q_k,m_k}.$$
- If $t = F^{\theta,m}\ \vec{t_1}\ldots\vec{t_k}$ then $g(t) = t$.

The term $g(t)$ satisfies that $(g(t))^{\perp} = t^{\blacktriangle}$. Furthermore since we assumed that for all $F^{\theta,m} \in \mathcal{N}' \cap \mathcal{N}_{\Theta+}$, $Comp(F^{\theta,m}) \neq \emptyset$, all coloured run $t \in \mathcal{T}(\mathcal{N}')$ is non-blocking. Therefore the OI-parallel derivation $s_0 \Rightarrow_{\mathrm{OI}} s_1 \Rightarrow_{\mathrm{OI}} s_2 \Rightarrow_{\mathrm{OI}} \cdots$ in $\mathbb{S}$ is non-blocking, thus its associated tree R, is a run on $\|\mathcal{G}\|$ from state q.

We know that R is the supremum of $\{s_i{}^{\blacktriangle} \mid i \geq 0\}$ and since for all i, there exists $S^{q,0} \to^*_{\mathcal{G}'} s'_i$ such that $(s'_i)^{\perp} = s_i{}^{\blacktriangle}$, we have that $\|\mathcal{G}\|$ is an upper-bound of $\{s_i{}^{\blacktriangle} \mid i \geq 0\}$. Therefore $R \sqsubseteq \|\mathcal{G}\|$. Since R is a run on a tree that does not contain $\perp$, R do not contain $\perp$, therefore $R = \|\mathcal{G}\|$. □

We add a condition on these schemes in order for them to produce accepting runs.

Definition 4.5.8.

Given $\mathcal{G}'$ a run-constructing scheme from state q for $\mathcal{G}$. The scheme $\mathcal{G}'$ is ***winning*** if for all infinite path on elements of $\mathcal{N}_{\Theta+}$ in the recursion graph of $\mathcal{G}$:
$$\alpha_1^{\theta_1,m_1}, \alpha_2^{\theta_2,m_2}, \ldots$$
the greatest colour seen infinitely often in the sequence $m_1, m_2, \ldots$ is even.

The next proposition shows that this condition is sufficient for a scheme to produce an accepting run.

Proposition 4.5.16.

Given $\mathcal{G}'$ a winning run-constructing scheme from state q. Then $\|\mathcal{G}'\|$ is an accepting run of $\mathcal{A}$ over $\|\mathcal{G}\|$ from state q.

Proof. Take the derivation d associated with $\mathcal{G}'$, and take a creation sequence *seq* in that derivation. Note that if an occurrence $(F^{\theta,m}, C[\bullet])$ creates another $(H^{\theta',m'}, C'[\bullet])$, then there is an edge $(F^{\theta,m}, H^{\theta',m'})$ in the recursion graph of $\mathcal{G}'$. Therefore, due to the winning constraint on $\mathcal{G}'$, one can map *seq* to an infinite path in the recursion graph of $\mathcal{G}'$, thus the colour of *seq* is even. □

This last proposition shows that one can construct a winning run-constructing scheme from a winning strategy in the game.

Proposition 4.5.17.

If Eve wins in Kobayashi-Ong game from vertex $S^{q,0}$, then there exists a winning run-constructing scheme from state q.

Proof. Let $\mathcal{N}'' \subseteq \mathcal{N}_{\Theta+}$ be the set of winning vertices of Eve, and let f be a positional winning strategy. Then for all $F^{\theta,m} \in \mathcal{N}''$, $f(F^{\theta,m}) \in \mathit{Comp}(F^{\theta,m})$ and furthermore $f(F^{\theta,m})$ conforms to $\mathcal{N}''$ on $\mathcal{N}$. Therefore one can construct a run-constructing scheme $\mathcal{G}' = \langle \Sigma', \mathcal{N}', \mathcal{V}', S', \mathcal{R}' \rangle$ as follows.

- $\Sigma' = \Sigma_{\Theta_s}$,
- $\mathcal{N}' = \Sigma_{\Theta+} \cup \mathcal{N}'' \cup \mathit{Boxes}'$, where Boxes' is the finite set of elements $\bot^\tau$ where τ is a subtype of types appearing in $\Sigma_{\Theta+} \cup \mathcal{N}''$.
- $\mathcal{V}'$ is equal to the union of $\mathcal{V}_{\Theta+}$, $\{\eta_i^{\theta,m} : o \mid i \leq \mathit{arity}_{\max}, \theta \in \Theta+, m \in M\}$, and $\{y_i^\tau : \tau_i \mid \square^\tau \in \mathit{Boxes}', \tau = \tau_1 \to \cdots \tau_k \to o, i \leq k\}$.
- $S' = S^{q,0}$,
- For all $F^{\theta,m} \in \mathcal{N}_{\Theta+} \cap \mathcal{N}'$,

$$F^{\theta,m}\ \vec{x_1} \dots \vec{x_k} \to f(F^{\theta,m})\ \in \mathcal{R}'.$$

Take an infinite path on elements of $\mathcal{N}_{\Theta+}$ in the recursion graph of $\mathcal{G}'$:

$$\alpha_1^{\theta_1,m_1}, \alpha_2^{\theta_2,m_2}, \dots$$

Since for all i, there is an occurrence of $\alpha_{i+1}^{\theta_{i+1},m_{i+1}}$ in $f(\alpha_i^{\theta_i,m_i})$, the following sequence is a play in the game where Eve respects here winning strategy f:

$$\alpha_1^{\theta_1,m_1}, f(\alpha_1^{\theta_1,m_1}), \alpha_2^{\theta_2,m_2}, f(\alpha_2^{\theta_2,m_2}), \dots$$

The sequence of colour along this play is $m_1, 0, m_2, 0, \cdots$, and since Eve follows her winning strategy, the maximum colour appearing infinitely often in this sequence is even. Therefore $\mathcal{G}'$ is winning. □

Remark that we did not use Kobayashi-Ong Theorem to prove this result. Therefore it constitutes a proof of the soundess of this theorem, *i.e.* it shows that if $S^{q,0}$ is winning in the game, there exist an accepting run of the automaton over $\|\mathcal{G}\|$. We do not give here a proof of the completeness of this theorem, it can be found in [KO09], and a more complete proof can be found in [KO12].

Finally, Taking into accout Kobayashi-Ong Theorem, one can give an algorithm that solves the automata selection problem, depicted below.

Input: A scheme $\mathcal{G}$, an automaton $\mathcal{A}$, a state q such that $\mathcal{A}$ accepts $\mathcal{G}$ from q.

1. Compute the Kobayashi-Ong game associated with $\mathcal{G}$ and $\mathcal{A}$.
2. According to their theorem, $S^{q,0}$ is winning in that game, therefore one can compute a winning run-constructing scheme $\mathcal{G}'$ from q.

Output: $\mathcal{G}'$ such that that $\|\mathcal{G}'\|$ is an accepting run of $\mathcal{A}$ on $\|\mathcal{G}\|$ from state q.

5. Conclusion

We conclude here the work presented in this thesis. We first recall the main contributions presented in the previous chapters, and then we discuss three possible developments of these results and of the domain of higher-order recursion scheme verification.

5.1. Contributions

In this thesis we have investigated three different problems. All three of them are expressed as scheme transformation problems, *i.e.* one of their inputs is a scheme, and the output must be another scheme related to the initial one.

Simulation of evaluation policies. Given a scheme $\mathcal{G}$ and two evaluation policies $\pi \neq \pi' \in \{\to_{\mathrm{IO}}, \to_{\mathrm{OI}}\}$, can we construct a scheme $\mathcal{G}'$ such that $\|\mathcal{G}\|_\pi = \|\mathcal{G}'\|_{\pi'}$?

Reflection. Let $\mathcal{G}$ be a scheme, $\varphi[x]$ be an MSO formula and let S be the set of nodes u in $\|\mathcal{G}\|$ recognised by $\varphi[x]$, *i.e.* such that $\|\mathcal{G}\| \models_{\{x \mapsto u\}} \varphi[x]$. Can we construct a scheme $\mathcal{G}'$ such that $\|\mathcal{G}'\|$ is the S-marking of $\|\mathcal{G}\|$?

Selection. Given a scheme $\mathcal{G}$ and an MSO formula φ of the form $\varphi = \exists X \psi[X]$, such that $\|\mathcal{G}\| \models \varphi$. Can we construct a scheme $\mathcal{G}'$ such that there exists a set of nodes S witnessing φ, *i.e.* $\|\mathcal{G}\| \models_{\{X \mapsto S\}} \psi[x]$ and such that $\|\mathcal{G}'\|$ is the S-marking of $\|\mathcal{G}\|$?

Although the last two problems were already solved in [BCOS10, CS12], we sought some *shape-preserving* scheme transformations, in particular we tried, and succeeded, to find constructions that do not involve the use of CPDA.

In Chapter 3, we introduced the notion of *morphism*, in order to define semantics for HORS. A morphism maps the terms of a scheme to the elements of a typed applicative system (TAS), *i.e.* a typed set with a partial binary operation guarded by the types of the elements. A morphism is *weakly stable by rewrite* if the value of a term remains unchanged when rewriting it. Then, we constructed a scheme transformation that takes as input a scheme $\mathcal{G}$ and a morphism φ that is weakly stable by rewrite, and outputs a scheme $\mathcal{G}_\varphi$ that consists of an embedding of φ in $\mathcal{G}$. During a derivation, $\mathcal{G}_\varphi$ annotates terms with their values according to the morphism.

Then, we investigated the *semantics problem*, that we can state as follows. Given a type-preserving mapping ζ from the terminal symbols of a scheme to a TAS, can we define/construct a morphism φ such that on all terminal symbol, φ acts as ζ, and that is stable by rewrite? When we see a scheme as a program whose built-in functions are the terminal symbols of the scheme, this problem is expressed as follows. Can we give a semantics of the program such that the interpretation of the built-in function corresponds to their meaning, and such that the interpretations of the user-defined functions are equal to the interpretation of their definitions?

Note that this problem is different from the one of giving a semantics to the whole programming language. In our case, we fix a program, and then define a semantics accordingly. As a comparison, this later problem corresponds to the problem of giving a semantics to the λY-calculus.

We presented a fixpoint procedure, which is a standard semantics construction, and a reformulation of [Kob09], that allows to solve the semantics problem in some cases (when the domain of ζ satisfies some usual properties, and ζ is continuous). When the domain of ζ is finite, this procedure can effectively be constructed. This allows to construct morphisms that recognise some useful properties such as deciding whether a term is unproductive or not. We gave another example: given a TAC automaton, one can encode the transitions of the automaton into a mapping ζ, and therefore obtain a morphism φ such that for all term $t : o$, $\varphi(t)$ is equal to the set of states from which the automaton accepts the value tree of t. When we input these morphisms, along with the scheme, to the embedding algorithm presented earlier, we are able to solve some problems such as the $\bot$-elimination problem, or the TAC-automata reflection problem.

The main contributions of this chapter are the simulation algorithms that allows to simulate with an evaluation policy the behaviour of a scheme according to another evaluation policy. More precisely we solved the two following problems:

(1) Given a scheme $\mathcal{G}$, can we construct a scheme $\mathcal{G}'$ such that $\|\mathcal{G}'\| = \|\mathcal{G}\|_{\mathrm{IO}}$?

(2) Given a scheme $\mathcal{G}$, can we construct a scheme $\mathcal{G}'$ such that $\|\mathcal{G}'\|_{\mathrm{IO}} = \|\mathcal{G}\|$?

The second problem is solved by a simple CPS-like transformation. The first problem is more difficult, and we used semantics techniques to solve it. We designed a fixpoint procedure, as the one above, to compute a morphism that recognises whether the IO-value tree of a term is equal to $\bot$. Then we presented an algorithm that solves (2), working as follows. Given a scheme $\mathcal{G}$, the algorithm first computes the morphism described above. Then it computes an embedding $\mathcal{G}'$ of this morphism into $\mathcal{G}$. Finally, it constructs $\mathcal{G}''$, obtained from $\mathcal{G}'$ by forcing, even during an OI derivation, to produce $\bot$ when it detects that the IO-value tree would do so.

In Chapter 4, we gave two scheme-transformations to solve MSO-reflection and MSO-selection problems. Due to the equi-expressivity of the MSO logic and the parity automata, we focus on the automata version of these problems[1]. First, we extended the notions of transition of run of an automaton, in order to be able to associate transitions with higher-order non-terminal symbols, and to define runs of an automaton over a term. Then we stated a correspondence problem as follows. Given a scheme and an automaton, can we define a set δ of transitions associated with the non-terminal symbols, such that for all term $t : o$, the value tree of t is accepted by the automaton if and only if there exists a run of the automaton over t using the transitions in δ? We showed later that this problem is solved by computing the winning region of the parity game introduced in [KO09]. Finally we explained how, once we have solved this problem, one can compute a morphism that describes the behaviour of an automaton over terms of a

[1]Nonetheless, the reduction from automata reflection to MSO reflection is not trivial, as seen in Chapter 4.

scheme. Embedding this morphism into the scheme allows then to solve the reflection problem.

The scheme-transformation for the selection uses the winning strategy of the game of [KO09]. Recall that the automata-selection requests, given a scheme and an automaton, to produce another scheme whose value tree is an accepting run of the automaton over the value tree of the initial scheme. To solve this problem, we embed a positional winning strategy of the game in the scheme annotated with transitions of the automaton, such that with every infinite branch in the scheme is associated a play in the game. Furthermore we have shown that the greatest colour appearing infinitely often in the play is equal to the greatest colour appearing infinitely often along the branch. Since the strategy is winning, this colour is even, thus the value tree of the new scheme is an accepting run of the automaton.

Although any instance of the MSO-reflection problem can be linearly encoded into an instance of the MSO-selection problem, *i.e.* solving the MSO-selection solves the MSO-reflection, we chose to present an independent solution for the reflection. This choice is motivated by the fact that on the other hand, the only known reduction from automata reflection to automata selection requires to complement the given automaton, which deeply increases the size of it.

5.2. Perspectives

We present three further investigations related to this thesis. These ideas are not direct extensions of our results. Rather, they describe three directions for the domain that may greatly enrich it.

Applications. Kobayashi developed a (TAC-automata) model-checker tool for higher-order recursion scheme, T-RecS [Kob11]. Neatherway, Ong and Ramsay conceived another one based on games semantics [NOR12]. Broadbent, Carayol, Hague and Serre designed another (TAC-automata) model-checker, C-SHORe, based on a saturation algorithm for CPDA [BCHS13]. These three programs achieved some impressive benchmarks regarding the high theoretical complexity of the problem. T-RecS has been extended with prototype verification tools, MoCHi and EHMTT verifier, and it takes parts in a great effort to develop a model-checker for higher-order programs.

Therefore, it would be interesting to make these tools feature global model-checking/reflection and witness generation/selection.

Semantics and λY-calculus. Some of the problems we introduced here are simple in the domain of HORS, but their difficulty raises when transposed to λY-calculus. The question of morphism/semantics, in particular for the MSO logic, becomes particularly interesting and involved when we seek a semantics for the λY-calculus that describes the behaviour of an automaton, as we did for HORS. In my opinion, although studying schemes and scheme transformations is strongly justified by the need to stay as close as possible to the programs we describe, the λY-calculus is a more interesting model regarding semantics problem. Salvati and Walukiewicz have found some nice results on that topic in the past few years [SW13b, SW13a].

Precisely investigate where lies the difficulty in the λY-problems compared to the HORS ones may allow to spot what needs to be overcome. In particular, if we restrict the structure of λ-terms in order for them to be closer to HORS, would we be then able to solve such problems?

Model-Checking. All the model-checking algorithms for HORS work the same way. They reduce the problem of the acceptance of the value tree by the automaton, to the problem of solving a game. If the automaton has a safety winning condition, then the game is a safety game, if the automaton has a parity winning condition, so does the game. Other kinds of automata and corresponding games are interesting and have been recently studied: for instance automata and games with boundedness conditions (*e.g.* in [CF13]), or automata and games with counters (*e.g.* in [BKL12]). One may wonder whether the acceptance of the value tree of a scheme by one of these kinds of automata can be reduced to deciding who wins in a game of the same kind. This leads to a wide range of problems about HORS model-checking.

Bibliography

[ACD90] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking for real-time systems. In *Proc. 14th Annual IEEE Symp. on Logic in Computer Science (LICS'90)*. IEEE Computer Society, 1990.

[AdMO05] K. Aehlig, J. G. de Miranda, and C. H. L. Ong. Safety is not a restriction at level 2 for string languages. In *Proc. 8th Int'l Conf. on Foundations of software science and computational structures (FOSSACS'05)*. Springer-Verlag, 2005.

[Aeh06] Klaus Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. In *Proc. 20th Annual Conf. of the EACSL (CSL'06)*. Springer-Verlag, 2006.

[Aeh07] Klaus Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *Logical Methods in Computer Science (LMCS)*, 3, 2007.

[Bar85] H.P. Barendregt. *The Lambda Calculus, Its Syntax and Semantics*. Elsevier, 1985.

[BCHS13] Christopher Broadbent, Arnaud Carayol, Matthew Hague, and Olivier Serre. C-shore: A collapsible approach to verifying higher-order programs. Private communication, 2013.

[BCM+92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98, 1992.

[BCOS10] Christopher H. Broadbent, Arnaud Carayol, C.-H. Luke Ong, and Olivier Serre. Recursion schemes and logical reflection. In *Proc. 25th Annual IEEE Symp. on Logic in Computer Science (LICS'10)*. IEEE Computer Society, 2010.

[BEM97] Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proc. 8th Int'l Conf. on Concurrency Theory (CONCUR'97)*. Springer-Verlag, 1997.

[BKL12] Dietmar Berwanger, Łukasz Kaiser, and Simon Leßenich. Solving counter parity games. In *Proc. 37th Int'l Symp. on Mathematical Foundations of Computer Science (MFCS'12)*. Springer-Verlag, 2012.

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[BO09] Christopher Broadbent and Luke Ong. On global model checking trees generated by higher-order recursion schemes. In *Proc.12th Int'l Conf. on Foundations of software science and computational structures (FOSSACS'09)*. Springer-Verlag, 2009.

[Car06] Arnaud Carayol. *Automates infinis, logiques et languages*. PhD thesis, Université de Rennes 1, 2006.

[Cau96] Didier Caucal. On infinite transition graphs having a decidable monadic theory. In *Proc. 23rd Int'l Conf. on Automata, Languages, and Programming (ICALP'96)*. Springer-Verlag, 1996.

[Cau02] Didier Caucal. On infinite terms having a decidable monadic theory. In *Proc. 27th Symposium on Mathematical Foundations of Computer Science (MFCS'02)*. Springer-Verlag, 2002.

[CDG+07] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007.

[CE81] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*. Springer-Verlag, 1981.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. on Programming Languages and Systems (TOPLAS)*, 8, 1986.

[CF13] Krishnendu Chatterjee and Nathanaël Fijalkow. Infinite-state games with finitary conditions. In *Proc. 22th EACL Anual Conf. on Computer Science Logic (CSL'13)*. to be published, 2013.

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 1999.

[CHM+08] Arnaud Carayol, Matthew Hague, Antoine Meyer, C.-H. Luke Ong, and Olivier Serre. Winning regions of higher-order pushdown games. In *Proc. 23th Annual IEEE Symp. on Logic in Computer Science (LICS'08)*. IEEE Computer Society, 2008.

[CKV06] Hana Chockler, Orna Kupferman, and Moshe Y. Vardi. Coverage metrics for temporal logic model checking. *Formal Methods in System Design (FMSD)*, 28, 2006.

[CN78] Bruno Courcelle and Maurice Nivat. The algebraic semantics of recursive program schemes. In *Proc. 7th Symposium on Mathematical Foundations of Computer Science (MFCS'78)*. Springer-Verlag, 1978.

[Cou78a] Bruno Courcelle. A representation of trees by languages I. *Theoretical Computer Science (TCS)*, 6, 1978.

[Cou78b] Bruno Courcelle. A representation of trees by languages II. *Theoretical Computer Science (TCS)*, 7, 1978.

[Cou95] Bruno Courcelle. The monadic second-order logic of graphs ix: Machines and their behaviours. *Theoretical Computer Science (TCS)*, 151, 1995.

[CS12] Arnaud Carayol and Olivier Serre. Collapsible pushdown automata and labeled recursion schemes: Equivalence, safety and effective selection. In *Proc. 27th Annual IEEE Symp. on Logic in Computer Science (LICS'12)*. IEEE Computer Society, 2012.

[CW98] Bruno Courcelle and Igor Walukiewicz. Monadic second-order logic, graph coverings and unfoldings of transition systems. *Annals of Pure and Applied Logic*, 92, 1998.

[CW07] Thierry Cachat and Igor Walukiewicz. The complexity of games on higher order pushdown automata. *CoRR*, 2007.

[Dam77a] Werner Damm. Higher type program schemes and their tree languages. In *Proc. 3rd GI-Conference on Theoretical Computer Science*. Springer-Verlag, 1977.

[Dam77b] Werner Damm. Languages defined by higher type program schemes. In *Proc. 4th Colloq. on Automata, Languages, and Programming (ICALP'77)*, Lecture Notes in Comput. Sci. Springer-Verlag, 1977.

[Dam82] Werner Damm. The IO- and OI-hierarchies. *Theoretical Computer Science (TCS)*, 20, 1982.

[dG01] Philippe de Groote. Towards abstract categorial grammars. In *Proc. 39th Annual Meeting on Association for Computational Linguistics (ACL'01)*. Association for Computational Linguistics, 2001.

[dM06] Jolie G. de Miranda. *Structures Generated by Higher-Order Grammars and the Safety Constraint*. PhD thesis, Oxford University, 2006.

[EJ91] E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy. In *Proc. 32nd Ann. Symp. Found. Comput. Sci. (FOCS'91)*. IEEE Computer Society, 1991.

[Eng91] Joost Engelfriet. Iterated stack automata and complexity classes. *Information and Computation*, 95(1), 1991.

[Esp94] javier Esparza. On the decidability of model checking for several μ-calculi and petri nets. In *Proc. 19th Colloquium on Trees in Algebra and Programming (CAAP'94)*. Springer-Verlag, 1994.

[Had12] Axel Haddad. Io vs oi in higher-order recursion schemes. In *Proc. 8th Workshop on Fixed Points in Computer Science (FICS'12)*. Electronic Proceedings in Theoretical Computer Science, 2012.

[HMOS08] Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. In *Proc. 23th Annual IEEE Symp. on Logic in Computer Science (LICS'08)*. IEEE Computer Society, 2008.

[Hun99] Hardi Hungar. Model checking and higher-order recursion. In *Proc. 24th Symposium on Mathematical Foundations of Computer Science (MFCS'99)*. Springer, 1999.

[Ind76] Klaus Indermark. Schemes with recursion on higher types. In *Proc. 5th Symposium on Mathematical Foundations of Computer Science (MFCS'76)*. Springer-Verlag, 1976.

[KNU01] Teodor Knapik, Damian Niwinski, and PawełUrzyczyn. Deciding monadic theories of hyperalgebraic trees. In *Proc. 5th Int'l Conf. on Typed lambda-Calculi and Applications (TLCA'01)*. Springer-Verlag, 2001.

[KNU02] Teodor Knapik, Damian Niwiński, and Pawel Urzyczyn. Higher-order pushdown trees are easy. In *Proc. 5th Int'l Conf. on Foundations of Software Science and Computation Structures (FoSSaCS'02)*. Springer-Verlag, 2002.

[KNUW05] Teodor Knapik, Damian Niwiński, PawełUrzyczyn, and Igor Walukiewicz. Unsafe grammars and panic automata. In *Proc. 32nd Int'l Conf. on Automata, Languages, and Programming (ICALP'05)*. Springer-Verlag, 2005.

[KO09] Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proc. 24th Annual IEEE Symp. on Logic in Computer Science (LICS'09)*. IEEE Computer Society, 2009.

[KO12] Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. Private communication, 2012.

[Kob09] Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *Proc. 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'09)*. ACM, 2009.

[Kob11] Naoki Kobayashi. A practical linear time algorithm for trivial automata model checking of higher-order recursion schemes. In *Proc.14th Int'l Conf.*

on Foundations of software science and computational structures (FOSSACS'11). Springer-Verlag, 2011.

[KS13] M. Kobele, Gregory and Sylvain Salvati. The io and oi hierarchies revisited. In *Proc. 40th Int'l Conf. on Automata, Languages, and Programming (ICALP'13)*. Springer-Verlag, 2013.

[LBBO01] Yassine Lakhnech, Saddek Bensalem, Sergey Berezin, and Sam Owre. Incremental verification by abstraction. In *Proc. 7th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01)*. Springer-Verlag, 2001.

[Lin93] C. H. Lindsey. A history of ALGOL 68. In *Proc. 2nd ACM SIGPLAN Conference on History of programming languages (HOPL'93)*. ACM, 1993.

[McC62] John McCarthy. *LISP 1.5 Programmer's Manual*. The MIT Press, 1962.

[Miq] Alexandre Miquel. A combinatorial proof of strong normalisation for the simply typed -calculus. Manuscript.

[MS85] David E. Muller and Paul E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science (TCS)*, 37, 1985.

[Mus01] Reinhard Muskens. Lambda-grammars and the syntax-semantics interface. In *Proc. 13th Amsterdam Colloquium (AC'01)*. Springer-Verlag, 2001.

[Niv72] Maurice Nivat. Langages algébriques sur le magma libre et sémantique des schémas de programme. In *Automata, Languages, and Programming: Proceedings of a Symposium (ICALP'72)*. Springer-Verlag, 1972.

[Niv75] M. Nivat. On the interpretation of recursive program schemes. *Symposia Matematica*, 15, 1975.

[NOR12] Robin P. Neatherway, C.-H. Luke Ong, and Steven J. Ramsay. A traversal based algorithm for higher-order model checking. In *Proc. 17th ACM SIGPLAN International Conference on Functional Programming (ICFP '12)*. ACM New York, NY, USA, 2012.

[Ong06] C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In *Proc. 21th Annual IEEE Symp. on Logic in Computer Science (LICS'06)*. IEEE Computer Society, 2006.

[Par12] Pawel Parys. On the significance of the collapse operation. In *Proc. 27th Annual IEEE Symp. on Logic in Computer Science (LICS'12)*. IEEE Computer Society, 2012.

[Pla66] R. G. Platek. *Foundations of Recursion Theory*. PhD thesis, Standford University, 1966.

[Plo75] Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theoretical Computer Science (TCS)*, 1, 1975.

[PV04] Nir Piterman and MosheY. Vardi. Global model-checking of infinite-state systems. In *Computer Aided Verification*, volume 3114. Springer-Verlag, 2004.

[QS82] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. In *Int'l Symposium on Programming, 5th Colloquium*. Springer-Verlag, 1982.

[Rab69] M.O. Rabin. Decidability of second order theories and automata on infinite trees. *Trans. American Mathematical Society*, 141, 1969.

[Sén97] Géraud Sénizergues. The equivalence problem for deterministic pushdown automata is decidable. In *Proc. 24th Int'l Conf. on Automata, Languages, and Programming (ICALP'97)*. Springer-Verlag, 1997.

[SS75] Gerald Jay Sussman and Guy L Jr. Steele. Scheme: An interpreter for extended lambda calculus. In *MEMO 349*. MIT AI LAB, 1975.

[SS04] Tobias Schuele and Klaus Schneider. Global vs. local model checking: A comparison of verification techniques for infinite state systems. In *Proc. 2nd Int'l Conf. on Software Engineering and Formal Methods (SEFM'04)*. IEEE Computer Society, 2004.

[SUK13] Ryosuke Sato, Hiroshi Unno, and Naoki Kobayashi. Towards a scalable software model checker for higher-order programs. In *Proc. ACM SIGPLAN workshop on Partial evaluation and program manipulation (PEPM'13)*. ACM, 2013.

[SW11] Sylvain Salvati and Igor Walukiewicz. Krivine machines and higher-order schemes. In *Proc. 38th Int'l Conf. on Automata, Languages, and Programming (ICALP'11)*. Springer-Verlag, 2011.

[SW12] Sylvain Salvati and Igor Walukiewicz. Recursive schemes, krivine machines, and collapsible pushdown automata. In *Proc. 6th Int'l Conf. on Reachability Problems (RP'12)*. Springer-Verlag, 2012.

[SW13a] Sylvain Salvati and Igor Walukiewicz. Evaluation is msol compatible. Research report, INRIA Bordeaux - Sud-Ouest - LaBRI, 2013.

[SW13b] Sylvain Salvati and Igor Walukiewicz. Using models to model-check recursive schemes. In *Proc. 11th Int'l Conf. on Typed lambda-Calculi and Applications (TLCA'13)*. Springer-Verlag, 2013.

[Tho09] Wolfgang Thomas. Facets of synthesis: Revisiting church's problem. In *Proc.12th Int'l Conf. on Foundations of software science and computational structures (FOSSACS'09)*. Springer-Verlag, 2009.

[Wal96] Igor Walukiewicz. Monadic second order logic on tree-like structures. In *Proc. 13th Symp. Theoretical Aspects of Computer Science (STACS'96)*. Springer-Verlag, 1996.

[Wal01] Igor Walukiewicz. Pushdown processes: games and model-checking. *Information and Computation*, 164, 2001.

[Wij69] A. Wijngaarden. *Report on the algorithmic language ALGOL 68*. Printing by the Mathematisch Centrum, 1969.

A. Appendix

A.1. Proof of Theorem 3.4.6

Lemma 1.1.1.

For all term t, for all i, if t does not contain any redex, then $\pi(\Phi_i^{\mathcal{G}}(t)) = \texttt{fin}$.

Proof. The proof proceeds by induction on the structure of t.

- If $t = F \in \mathcal{N}$ and has not type o, or $t = a \in \Sigma$, $\pi(t) = \texttt{fin}$ by definition.
- If $t = a\, t_1 \dots t_k$ with $a \in \Sigma$, by induction hypothesis, for all $j \leq k$, $\pi(\Phi_i^{\mathcal{G}}(t_j)) = \texttt{fin}$, therefore $\pi(\Phi_i^{\mathcal{G}}(t)) = \texttt{fin}$ by definition of $\zeta(a)$.
- If $t = F\ t_1 \dots t_k$ with $F \in \mathcal{N}$ and $k \leq \mathit{arity}(F)$, by induction hypothesis, for all $j \leq k$, $\pi(\Phi_i^{\mathcal{G}}(t_j)) = \texttt{fin}$, therefore $\pi(\Phi_i^{\mathcal{G}}(t)) = \texttt{fin}$ by definition of $\Phi_0^{\mathcal{G}}$ if $i = 0$, or by definition of $\mathcal{H}^{\mathcal{G}}$ if $i > 0$.

□

We can now prove that given a scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$, $\Phi^{\mathcal{G}}(S) = \|\mathcal{G}\|_{\mathrm{IO}}^{\zeta}$. In order to do so we prove that $\|\mathcal{G}\|_{\mathrm{IO}}^{\zeta} \sqsubseteq \Phi^{\mathcal{G}}(S)$ (Proposition **1.1.2**) and that $\Phi^{\mathcal{G}}(S) \sqsubseteq \|\mathcal{G}\|_{\mathrm{IO}}^{\zeta}$ (Proposition **1.1.11**).

Proposition 1.1.2.

Given a scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$,

$$\|\mathcal{G}\|_{\mathrm{IO}}^{\zeta} \sqsubseteq \Phi^{\mathcal{G}}(S).$$

Claim 1.1.3. Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$. For all terms t, t', for all k, if $t \Rightarrow_{\mathrm{IO}} t'$, then

$$\Phi_k^{\mathcal{G}}(t') \sqsubseteq \Phi_{k+1}^{\mathcal{G}}(t).$$

⌟

Proof. We proceed by induction on t.

- If $t = a\ t_1 \dots t_k$ with $a \in \Sigma$ then $t' = a\ t'_1 \dots t'_k$ such that for all i, $t_i \Rightarrow_{\mathrm{IO}} t'_i$. Then $\Phi^{\mathcal{G}}_k(t') = \zeta(a)\ \Phi^{\mathcal{G}}_k(t'_1) \dots \Phi^{\mathcal{G}}_k(t'_k)$. By induction hypothesis, for all i, $\Phi^{\mathcal{G}}_k(t'_i) \sqsubseteq \Phi^{\mathcal{G}}_{k+1}(t_i)$, therefore
$$\Phi^{\mathcal{G}}_k(t') \sqsubseteq \zeta(a)\ \Phi^{\mathcal{G}}_{k+1}(t_1) \dots \Phi^{\mathcal{G}}_{k+1}(t_k) = \Phi^{\mathcal{G}}_{k+1}(t).$$
- If $t = F\ t_1 \dots t_k$ is not an IO redex, then $t' = F\ t'_1 \dots t'_k$ such that for all i, $t_i \Rightarrow_{\mathrm{IO}} t'_i$. Then $\Phi^{\mathcal{G}}_k(t') = \Phi^{\mathcal{G}}_k(F)\ \Phi^{\mathcal{G}}_k(t'_1) \dots \Phi^{\mathcal{G}}_k(t'_k)$. By induction hypothesis, for all i, $\Phi^{\mathcal{G}}_k(t'_i) \sqsubseteq \Phi^{\mathcal{G}}_{k+1}(t_i)$. Furthermore, by continuity of $\mathcal{H}^{\mathcal{G}}$ $\Phi^{\mathcal{G}}_k(F) \sqsubseteq \Phi^{\mathcal{G}}_{k+1}(F)$, therefore
$$\Phi^{\mathcal{G}}_k(t') \sqsubseteq \Phi^{\mathcal{G}}_{k+1}(F)\ \Phi^{\mathcal{G}}_{k+1}(t_1) \dots \Phi^{\mathcal{G}}_{k+1}(t_k) = \Phi^{\mathcal{G}}_{k+1}(t).$$
- If $t = F\ t_1 \dots t_k$ is a IO redex (*i.e.* $k = \mathit{arity}(F)$ and for all i, t_i contains no redex), then using Lemma **1.1.1** for all i, $\pi(\Phi_k(t_i)) = \pi(\Phi_{k+1}(t_i)) = \mathtt{fin}$. Let $F\ x_1 \dots x_k \to_{\mathrm{IO}} e \in \mathcal{R}$.
  - We have $t' = e_{[\forall i\ x_i \mapsto t_i]}$ then $\Phi^{\mathcal{G}}_k(t') = \Phi^{\mathcal{G}}_k(e_{[\forall i\ x_i \mapsto t_i]})$.
  - Furthermore, since $\Phi^{\mathcal{G}}_{k+1} = \mathcal{H}_{\mathcal{G}}(\Phi^{\mathcal{G}}_k)$, and for all i $\pi(\Phi_k(t_i)) = \mathtt{fin}$, we have $\Phi^{\mathcal{G}}_{k+1}(t) = \Phi^{\mathcal{G}}_{k\,\rho}(e)$ with ρ is the environment such that for all i, $\rho(x_i) = \Phi^{\mathcal{G}}_{k+1}(t_i)$.

  Therefore, we can conclude since $\Phi^{\mathcal{G}}_k(e_{\left[\forall i\ x_i \mapsto t'_i\right]}) \sqsubseteq \Phi^{\mathcal{G}}_{k\,\rho}(e)$.

□

Claim 1.1.4. Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be a scheme. Let $S = t_0 \Rightarrow_{\mathrm{IO}} t_1 \Rightarrow_{\mathrm{IO}} \cdots$ be the parallel derivation from S. For all k, $\Phi_\zeta(t_k) \sqsubseteq \Phi^{\mathcal{G}}_k(S)$. ⌟

Proof. We prove by induction on $i \leq k$ that $\Phi^{\mathcal{G}}_{k-i}(t_i) \sqsubseteq \Phi^{\mathcal{G}}_k(S)$. When $i = 0$ the result is trivial. Assume that $\Phi^{\mathcal{G}}_{k-i}(t_i) \sqsubseteq \Phi^{\mathcal{G}}_k(S)$ then, since $t_i \Rightarrow_{\mathrm{IO}} t_i + 1$, Claim **1.1.3** states that $\Phi^{\mathcal{G}}_{k-i-1}(t_{i+1}) \sqsubseteq \Phi^{\mathcal{G}}_{k-i}(t_i)$, therefore $\Phi^{\mathcal{G}}_{k-(i+1)}(t_{i+1}) \sqsubseteq \Phi^{\mathcal{G}}_k(S)$. When $i = k$ we obtain (1). □

Proof of Proposition **1.1.2**. From Claim **1.1.4** we get that for all k, $\Phi_\zeta(t_k) \sqsubseteq \Phi^{\mathcal{G}}_k(S) \sqsubseteq \Phi^{\mathcal{G}}(S)$, therefore $\Phi^{\mathcal{G}}(S)$ is an upper bound of the sequence $\Phi_\zeta(t_0) \sqsubseteq \Phi_\zeta(t_1) \sqsubseteq \cdots$ whose limit is $\|\mathcal{G}\|^\zeta_{\mathrm{IO}}$, then $\|\mathcal{G}\|^\zeta_{\mathrm{IO}} \sqsubseteq \Phi^{\mathcal{G}}(S)$. □

Now we prove $\Phi^{\mathcal{G}}(S) \sqsubseteq \|\mathcal{G}\|^\zeta$. As previously, we use scheme $\mathcal{G}^{(m)}$ with $m \geq 0$ (see Definition **3.3.9**).

Again, we extend ζ on $\Sigma \uplus \{\mathtt{bot}\}$ as $\zeta(\mathtt{bot}) = x_\perp$. For ease of exposition, we write $\mathcal{H}^{(m)} = \mathcal{H}_{\mathcal{G}^{(m)}}$, for all i $\Phi^{(m)}_i = \Phi^{\mathcal{G}^{(m)}}_i$ and $\Phi^{(m)} = \Phi^{\mathcal{G}^{(m)}}$.

The following lemma states that (1) the device morphism of $\mathcal{G}^{(m)}$ is reached after m iteration of the mapping $\mathcal{H}^{(m)}$ and (2) that before this morphisms is reached the morphisms obtained after some iterations of $\mathcal{H}^{(m)}$ are the same as those obtained in $\mathcal{G}$ after the same amount of iterations of $\mathcal{H}_{\mathcal{G}}$, *i.e.* for all $i \leq m$ and all term t, $\Phi_i^{\mathcal{G}}(t) = \Phi_i^{(m)}(t_{lab})$, where t_{lab} is the term t where the nonterminals have been labelled by i.

Lemma 1.1.5.

Let $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ be a scheme. The limit of the least sequence of $\mathcal{G}^{(m)}$ is reached at $\Phi_m^{(m)}$ *i.e.*

$$\Phi^{(m)} = \Phi_m^{(m)}. \tag{1}$$

Furthermore given a term $t \in \mathcal{T}(\Sigma \uplus \mathcal{N})$, for all $i \leq m$ we have

$$\Phi_i^{\mathcal{G}}(t) = \Phi_i^{(m)}(t_{[\forall H \in \mathcal{N}\ H \mapsto H^i]}). \tag{2}$$

Proof. To prove (1), we prove a more general result: for all $i \leq m$, for all $F^j \in \mathcal{N}^{(m)}$ with $j \leq i$,

$$\Phi^{(m)}(F^j) = \Phi_i^{(m)}(F^j).$$

Let $F^j\ x_1 \dots x_k \to e_j$ be the rewrite rule associated with F^j.

For $j = 0$ we have for all $i > 0$, $\Phi_i^{(m)}(F^0)\ h_1 \dots h_k = \zeta(\texttt{bot}) = x_\perp$, when $i = 0$, we also have $\Phi_i^{(m)}(F^0)\ h_1 \dots h_k = x_\perp$. And since the limit of the constant sequence $x_\perp$ is $x_\perp$, we have $\Phi^{(m)}(F^0)\ h_1 \dots h_k = \zeta(\texttt{bot}) = \perp$.

For $0 < j \leq i$, first observe that for all i_1, i_2, for all j', and for all non-terminal symbol $F^{j'}$, if $\mathit{tuple}(\Phi_{i_1}^{(m)}(F^{j'})) = (\nu_0^1, \dots, \nu_k^1)$ and $\mathit{tuple}(\Phi_{i_2}^{(m)}(F^{j'})) = (\nu_0^2, \dots, \nu_k^2)$, then by definition, for all $\ell < k$ $\nu_\ell^1 = \nu_\ell^2$, and for all $h_1, \dots, h_k$, if there exists ℓ such that $\pi(h_\ell) = \texttt{inf}$ then $\nu_k^1(h_1, \dots, h_k) = \nu_k^2(h_1, \dots, h_k)$. Therefore to show that $\Phi^{(m)}(F^j) = \Phi_i^{(m)}(F^j)$ we just need to prove that $\Phi_i^{(m)}(F^j)\ h_1 \dots h_k = \Phi^{(m)}(F_j)\ h_1 \dots h_k$ for all $h_1, \dots, h_k$ such that for all ℓ, $\pi(h_\ell) = \texttt{fin}$.

Since $\Phi_i^{(m)}(F^j)\ h_1 \dots h_k = \Phi_{i-1\,\rho}^{(m)}(e_j)$ and $\Phi_\rho^{(m)}(e_j) = \Phi^{(m)}(F_j)\ h_1 \dots h_k$ we need to prove

$$\Phi_{i-1\,\rho}^{(m)}(e_j) = \Phi_\rho^{(m)}(e_j),$$

with $\rho(x_i) = h_i$ for all i. Therefore, we prove by a side induction that for any $e \in \mathcal{T}(\Sigma \uplus \{F^{j-1} \mid F \in \mathcal{N}\} \uplus \{x_1, \dots, x_k\}$,

$$\Phi_{i-1\,\rho}^{(m)}(e) = \Phi_\rho^{(m)}(e).$$

- If $e = a \in \Sigma$, then $\Phi_{i-1\,\rho}^{(m)}(a) = \zeta(a) = \Phi_\rho^{(m)}(a)$,
- if $e = x_i$ for some i, then $\Phi_{i-1\,\rho}^{(m)}(x_i) = \rho(x_i) = \Phi_\rho^{(m)}(x_i)$,

- if $e = F^{j-1} \in \mathcal{N}^{(m)}$, then $\Phi^{(m)}_{i-1_\rho}(F^{j-1}) = \Phi^{(m)}_\rho(F^{j-1})$ by the main induction hypothesis,
- if $e = e_1\ e_2$, then

$$\Phi^{(m)}_{i-1_\rho}(e) = \Phi^{(m)}_{i-1_\rho}(e_1)\ \Phi^{(m)}_{i-1_\rho}(e_2) = \Phi^{(m)}_\rho(e_1)\ \Phi^{(m)}_\rho(e_2) = \Phi^{(m)}_\rho(e)$$

  by the side induction hypothesis.

We prove a slightly more general result than (2). We prove by induction on i that for all variables $\{x_1, \dots, x_k\}$, for all environment ρ whose domains contains x_i for all i, for all $e \in \mathcal{T}(\Sigma \uplus \mathcal{N} \uplus \{x_1, \dots, x_k\})$,

$$\Phi^{\mathcal{G}}_{i\ \rho}(e) = \Phi_{i\rho}{}^{(m)}(e_{[\forall H \in \mathcal{N}\ H \mapsto H^i]}).$$

If $i = 0$ then they are both equal to $\Phi_{\zeta_\rho}(e)$. For $i > 0$, we prove the result by a side induction on e, and the only interesting case is when $e = F \in \mathcal{N}$. Let $F\ y_1 \dots y_\ell \to e_F$ be the corresponding rewrite rule. By the same argument as previously, we just need to show that for all $h_1, \dots, h_k$, such that $\pi(h_\ell) = \texttt{fin}$ for all ℓ, $\Phi^{\mathcal{G}}_i(F)\ h_1 \dots h_\ell = \Phi^{(m)}_i(F^i)\ h_1 \dots h_\ell$. We have

$$\Phi^{\mathcal{G}}_i(F)\ h_1 \dots h_\ell = \Phi^{\mathcal{G}}_{i-1_{\rho_F}}(e_F)$$

with $\rho_F(y_j) = h_j$ for all j. By main induction hypothesis,

$$\Phi^{\mathcal{G}}_{i-1_{\rho_F}}(e_F) = \Phi^{(m)}_{i-1_{\rho_F}}(e_{F[\forall H \in \mathcal{N}\ H \mapsto H^{i-1}]}),$$

and since $F^m\ y_1 \dots y_\ell \to e_{F[\forall H \in \mathcal{N}\ H \mapsto H^{i-1}]} \in \mathcal{R}^{(m)}$, we have

$$\Phi^{(m)}_{i-1_{\rho_F}}(e_{F[\forall H \in \mathcal{N}\ H \mapsto H^{i-1}]}) = \Phi^{(m)}_i(F^i)\ h_1 \dots h_\ell,$$

therefore $\Phi^{\mathcal{G}}_i(F)\ h_1 \dots h_\ell = \Phi^{(m)}_i(F^i)\ h_1 \dots h_\ell$.

□

Before continuing, we prove that $\Phi^{\mathcal{G}}$ is stable by IO-rewrite.

Proposition 1.1.6.

Given a scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$, and to terms t, t' such that $t \Rightarrow_{\text{IO}} t'$, then $\Phi^{\mathcal{G}}(t) = \Phi^{\mathcal{G}}(t')$.

Proof. First we show that for all i, $\Phi_i^{\mathcal{G}}(t) \sqsubseteq \Phi_i^{\mathcal{G}}(t')$.We prove this result by main induction on i, and side induction on the structure of t, the only relevant case is when $t = F\ t_1 \dots t_k$ is an IO redex. Let $F\ x_1 \dots x_k = e \in \mathcal{R}$. If $i = 0$, then $\Phi_0^{\mathcal{G}}(t) = x_\perp \sqsubseteq \Phi_i^{\mathcal{G}}(t')$. Otherwise, since for all i, t_i does not contain any redex, therefore $\pi(t_i) = \texttt{fin}$ (thanks to Lemma **1.1.1**), and $\Phi_i^{\mathcal{G}}(t) = \Phi^{\mathcal{G}}_{i-1\,\rho}(e)$ where $\rho(x_i)$ is defined for all i, and equal to $\Phi_i^{\mathcal{G}}(t_i)$. Furthermore $\Phi_i^{\mathcal{G}}(t') = \Phi_i^{\mathcal{G}}(e_{[\forall i\ x_i \mapsto t_i]}) = \Phi^{\mathcal{G}}_{i\,\rho}(e)$. By induction hypothesis, $\Phi_i^{\mathcal{G}}(t) \sqsubseteq \Phi_i^{\mathcal{G}}(t')$. Since, $\Phi^{\mathcal{G}}(t)$ is the limit of the sequence $\Phi_0^{\mathcal{G}}(t) \sqsubseteq \Phi_1^{\mathcal{G}}(t) \sqsubseteq \cdots$, and $\Phi^{\mathcal{G}}(t')$ is the limit of the sequence $\Phi_0^{\mathcal{G}}(t') \sqsubseteq \Phi_1^{\mathcal{G}}(t') \sqsubseteq \cdots$, we have $\Phi^{\mathcal{G}}(t) \sqsubseteq \Phi^{\mathcal{G}}(t')$.

To prove that it is equal we use Claim **1.1.3**, that states that $\Phi_i^{\mathcal{G}}(t') \sqsubseteq \Phi_{i+1}^{\mathcal{G}}(t)$ for all i. Since, $\Phi^{\mathcal{G}}(t)$ is the limit of the sequence $\Phi_{0+1}^{\mathcal{G}}(t) \sqsubseteq \Phi_{1+1}^{\mathcal{G}}(t) \sqsubseteq \cdots$, and $\Phi^{\mathcal{G}}(t')$ is the limit of the sequence $\Phi_0^{\mathcal{G}}(t') \sqsubseteq \Phi_1^{\mathcal{G}}(t') \sqsubseteq \cdots$, we have $\Phi^{\mathcal{G}}(t') \sqsubseteq \Phi^{\mathcal{G}}(t)$. □

Corollary 1.1.7.

Given a scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$, if $S^m \Rightarrow^*_{\mathcal{G}^{(m)}} t$ then

$$\Phi_m^{\mathcal{G}}(S) = \Phi^{(m)}(t).$$

Proof. If $S^m \Rightarrow^*_{\mathcal{G}^{(m)}} t$, then there exists a derivation $S^m \Rightarrow_{\text{IO}} t_1 \Rightarrow_{\text{IO}} \cdots \Rightarrow_{\text{IO}} t_k = t$, therefore using the previous lemmas,

$$\Phi_m^{\mathcal{G}}(S) = \Phi_m^{(m)}(S^m) = \Phi^{(m)}(S^m) = \Phi^{(m)}(t_1) = \cdots = \Phi^{(m)}(t_k) = \Phi^{(m)}(t).$$

□

The following claim is a direct corollary of the strong normalisation theorem.

Corollary 1.1.8.

Given a scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$ and $m \geq 0$. There exists a term $t^{(m)} \in \mathcal{T}(\Sigma \uplus \{\texttt{bot}\})$ such that $S^m \Rightarrow^*_{\mathcal{G}^{(m)}} t^{(m)}$.

Lemma 1.1.9.

Given a scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$, then

$$\Phi_m^{\mathcal{G}}(S) = \Phi_\zeta(t^{(m)}).$$

Proof. $\Phi^{\mathcal{G}}_m(S) = \Phi^{(m)}(S^m) = \Phi^{(m)}(t^{(m)}) = \Phi_\zeta(t^{(m)})$. □

Lemma 1.1.10.

If $S^m \Rightarrow^*_{\mathcal{G}^{(m)}} t$ then there exists t' such that $S \Rightarrow^*_{\mathcal{G}} t'$ and $(t^\perp)_{[\mathtt{bot}\mapsto\perp]} \sqsubseteq (t')^\perp$.

Proof. We define the relation R between $\mathcal{T}(\Sigma \uplus \{\mathtt{bot}\} \uplus \mathcal{N}^{(m)} \uplus \mathcal{V})$ and $\mathcal{T}(\Sigma \uplus \mathcal{N} \uplus \mathcal{V})$ as follows. For all $t \in \mathcal{T}(\Sigma \uplus \{\mathtt{bot}\} \uplus \mathcal{N}^{(m)} \uplus \mathcal{V})$ and $t' \in \mathcal{T}(\Sigma \uplus \mathcal{N} \uplus \mathcal{V})$, $(t, t') \in R$ if and only if:

- either $t = \mathtt{bot}$,
- or $t = \alpha\ t_1 \dots t_k$ and $t' = \alpha'\ t'_1 \dots t'_k$ such that for all i, $(t_i, t'_i) \in R$, if $\alpha \in \Sigma \uplus \mathcal{V}$ then $\alpha' = \alpha$, and if $\alpha = F^j \in \mathcal{N}^{(m)}$ then $\alpha' = F \in \mathcal{N}$.

Let $t \in \mathcal{T}(\Sigma \uplus \{\mathtt{bot}\} \uplus \mathcal{N}^{(m)})$ and $t' \in \mathcal{T}(\Sigma \uplus \mathcal{N}^{(m)})$ such that $(t, t') \in R$. Observe that by induction on the structure of t, $(t^\perp)_{[\mathtt{bot}\mapsto\perp]} \sqsubseteq (t')^\perp$. Furthermore, given $e \in \mathcal{T}(\Sigma \uplus \{\mathtt{bot}\} \uplus \mathcal{N}^{(m)} \uplus \mathcal{V})$ and $e' \in \mathcal{T}(\Sigma \uplus \mathcal{N} \uplus \mathcal{V})$ such that $(e, e') \in R$, and given $x \in \mathcal{V}$, one can prove by induction on e that $(e_{[x\mapsto t]}, {e'}_{[x\mapsto t']}) \in R$. Finally, a last induction on the structure of t' shows that if $t \Rightarrow_{\mathcal{G}} s$ and $t' \Rightarrow_{\mathcal{G}^{(m)}} s'$, then $(s, s') \in R$.

Therefore, since $(S^m, S) \in R$, if $S^m \Rightarrow^k_{\mathcal{G}^{(m)}} t$ and $S \Rightarrow^k_{\mathcal{G}} t'$ for some k, then $(t, t') \in R$ (recall that $S^m \Rightarrow^k_{\mathrm{IO}} t$ means that there exists $t_1, \dots, t_{k-1}$ such that $S^m \Rightarrow_{\mathrm{IO}} t_1 \Rightarrow_{\mathrm{IO}} \cdots \Rightarrow_{\mathrm{IO}} t_{k-1} \Rightarrow_{\mathrm{IO}} t$). In particular $(t^\perp)_{[\mathtt{bot}\mapsto\perp]} \sqsubseteq (t')^\perp$. □

Proposition 1.1.11.

Given a scheme $\mathcal{G} = \langle \mathcal{V}, \Sigma, \mathcal{N}, S, \mathcal{R} \rangle$, we have $\Phi^{\mathcal{G}}(S) \sqsubseteq \|\mathcal{G}\|^\zeta$.

Proof. Let $S \Rightarrow_{\mathrm{IO}} t_1 \Rightarrow_{\mathrm{IO}} t_2 \Rightarrow_{\mathrm{IO}} \cdots$ be the parallel derivation of $\mathcal{G}$. First we prove that For all m, there exist N such that

$$\Phi^{\mathcal{G}}_m(S) \sqsubseteq \Phi_\zeta(t_N).$$

From Claim **1.1.10**, let t' be such that $S \Rightarrow^*_{\mathcal{G}} t'$ and $(t^{(m)})^\perp{}_{[\mathtt{bot}\mapsto\perp]} \sqsubseteq (t')^\perp$. Since $\zeta(\perp) = \zeta(\mathtt{bot})$, and for all $t : o$, $\Phi_\zeta(t) = \Phi_\zeta(t^\perp)$, therefore we have $\Phi_\zeta(t^{(m)}) \sqsubseteq \Phi_\zeta(t')$. From Claim **1.1.9**, we have $\Phi^{\mathcal{G}}_m(S) \sqsubseteq \Phi_\zeta(t^{(m)}) \sqsubseteq \Phi_\zeta(t')$.

For all m, there exists N such that $\Phi^{\mathcal{G}}_m(S) \sqsubseteq \Phi_\zeta(t_N)$, therefore $\|\mathcal{G}\|_\zeta$ is an upper-bound of $\Phi^{\mathcal{G}}_0(S) \sqsubseteq \Phi^{\mathcal{G}}_1(S) \sqsubseteq \cdots$, and since $\Phi^{\mathcal{G}}(S)$ is the lowest upper-bound of this sequence, $\Phi^{\mathcal{G}}(S) \sqsubseteq \|\mathcal{G}\|_\zeta$.

□