Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Unificação, Confluência e Tipos com Interseção para Sistemas de Reescrita Nominal

Ana Cristina Rocha Oliveira Valverde

Tese apresentada como requisito parcial para conclusão do Doutorado em Informática

Orientador
Prof. Dr. Maurício Ayala Rincón

Coorientadora
Prof. Dr. Maribel Fernández

Brasília
2016

Ficha Catalográfica de Teses e Dissertações

Está página existe apenas para indicar onde a ficha catalográfica gerada para dissertações de mestrado e teses de doutorado defendidas na UnB. A Biblioteca Central é responsável pela ficha, mais informações nos sítios:

http://www.bce.unb.br
http://www.bce.unb.br/elaboracao-de-fichas-catalograficas-de-teses-e-dissertacoes

Esta página não deve ser inclusa na versão final do texto.

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Unificação, Confluência e Tipos com Interseção para Sistemas de Reescrita Nominal

Ana Cristina Rocha Oliveira Valverde

Tese apresentada como requisito parcial para conclusão do Doutorado em Informática

Prof. Dr. Maurício Ayala Rincón (Orientador)
CIC/UnB

Prof. Dr. Maribel Fernández (Coorientadora)
Informatics/KCL

Prof. Dr. Cesar Augusto Muñoz
NASA Langley

Prof. Dr. Christian Urban
Informatics/KCL

Prof. Dr. Daniel Lima Ventura
CIC/UFG

Prof. Dr. Marcelo Finger
DCC/USP

Prof. Dr. Flávio Leonardo Cavalcanti de Moura (Membro suplente)
CIC/UnB

Prof. Dr. Célia Ghedini Ralha
Coordenadora do Programa de Pós-graduação em Informática

Brasília, 15 de setembro de 2016

Dedicatória

Ao meu filho querido, que nem sequer tem nome ainda, mas já é muito amado.

Agradecimentos

Primeiramente a Deus, que me deu forças e capacidade para concluir este trabalho.

Aos meus pais Assis e Tiana e ao meu irmão Bruno, pela alegria que me vem facilmente na companhia de vocês.

Ao meu esposo e grande amor Ricardo, pela vida e encantos inesperados que encontrei ao seu lado.

Aos meus companheiros de laboratório que se revelaram verdadeiros amigos: Ariane, Thiago, José Luís, Lucas, Daniel e Washington.

Aos colegas que conheci em Londres e me fizeram sentir parte do grupo numa cidade desconhecida. Particularmente, agradeço ao Elliot, que não somente foi um ótimo anfitrião, como também teve parte importante nesta tese através da disponibilização solícita do seu próprio trabalho.

Aos meus orientadores Mauricio e Maribel, que sempre tiveram confiança em mim e me guiaram com toda paciência.

Aos membros da banca examinadora, pela atenção dedicada a esse trabalho e pelas recomendações feitas. Em especial, agradeço ao Daniel Ventura, que foi um verdadeiro colaborador deste trabalho, dispensando horas de discussão e muitas viagens.

Ao Jamie Gabbay, pela oportunidade de trabalharmos juntos num trabalho que também faz parte desta tese.

À CAPES, pelo apoio financeiro que nunca faltou durante este curso.

Acknowledgements

Firstly to God, who has given me strength and capacity to conclude this work.

To my parents Assis and Tiana and to my brother Bruno, for the joy that I easily feel in your company.

To my husband and love Ricardo, for the unexpected life and delights that I have found by your side.

To my lab peers, who have shown to be true friends: Ariane, Thiago, José Luís, Lucas, Daniel and Washington.

To my colleagues who I have met in London and who made feel in a group, even in an unfamiliar city. Particularly, I thank to Elliot, who was such an excellent host and whose work was solicitously available, helping in an important part of this thesis.

To my advisors Mauricio and Maribel, who have always trusted me and have guided me so patiently.

To the members of the jury, for your devoted attention to this work and for your suggestions. Specially, I owe thanks to Daniel Ventura, who has largely contributed to this work, spending several hours in discussion and a lot of travels.

To Jamie Gabbay, for the opportunity of working together in a paper which is part of this thesis.

To CAPES, for the uninterrupted funding during this research.

Resumo

Sistemas nominais são uma abordagem alternativa para o tratamento de variáveis em sistemas computacionais, onde a reescrita de primeira ordem é generalizada através do suporte para especificação de ligação de variáveis e de α-equivalência, fazendo uso do conceito de *freshness* e da troca bijetiva de nomes (*swapping*). Teoremas bem conhecidos em reescrita de primeira ordem podem ser adaptados a fim de serem adicionalmente válidos em reescrita nominal. Nesta tese, nós analisamos sob que condições a confluência de sistemas de reescrita e o Critério dos Pares Críticos são válidos para a reescrita nominal, assim como para a reescrita nominal fechada (que é uma noção de reescrita mais eficiente nesse contexto). As condições definidas aqui são de fácil checagem através de um algoritmo de unificação nominal. A unificação nominal foi inicialmente estudada por Urban, Pitts e Gabbay e formalizada por Urban no assistente de prova Isabelle/HOL. Neste trabalho, são também apresentadas uma especificação nova de unificação nominal na linguagem do PVS e uma formalização da sua terminação, correção e completude. Em nossa especificação, ao invés de aplicar regras de simplificação a condições restritivas de unificação e *freshness*, soluções para um problema de unificação são construídas recursivamente através de uma especificação funcional direta, obtendo uma formalização que é mais próxima a implementações algorítmicas. Esta formalização é o passo inicial com vistas a ter mais resultados formalizados sobre reescrita nominal em PVS, onde um forte arcabouço de sistemas de reescrita de termos já é disponibilizado. Ademais, um sistema de tipos com interseção para termos nominais é apresentado. O sistema de tipos presente possui a importante propriedade de redução do sujeito para uma noção especializada de reescrita nominal tipada, o que significa que a tipabilidade no sistema é coerente com execuções computacionais.

Palavras-chave: sintaxe nominal, reescrita, confluência, ligação de variáveis, formalização, unificação, tipos com interseção

Abstract

Nominal systems are an alternative approach for the treatment of variables in computational systems where first-order rewriting is generalised by providing support for the specification of binding operators and α-equivalence using the notions of freshness and name swapping. Famous theorems in the context of first-order rewriting can be adapted for nominal rewriting as well. In this thesis, we analyse the conditions under which confluence of orthogonal rewrite systems and the Critical Pair Criterion hold for nominal rewriting as well as for closed nominal rewriting (an efficient notion of rewriting in this context). The conditions we define are easy to check using a nominal unification algorithm. Nominal unification was initially studied by Urban, Pitts and Gabbay and then first formalised by Urban in the proof assistant Isabelle/HOL. In this work, we also present a new specification of nominal unification in the language of PVS and a formalisation of its termination, soundness and completeness. In our specification, instead of applying simplification rules to unification and freshness constraints, we recursively build solutions for the original problem through a straightforward functional specification, obtaining a formalisation that is closer to algorithmic implementations. This formalisation is a first step in order to have more formalised results about nominal rewriting in PVS, where a huge background for term rewriting system is already available. Additionally, an intersection type system is presented for nominal terms. The present type system possesses the important property of subject reduction for a specialised notion of typed nominal rewriting, that means the soundness of typability under computational execution.

Keywords: nominal syntax, rewriting, confluence, binding, formalisation, unification, intersection types

Contents

1 Introduction **1**
1.1 Related work . . . 7
1.2 Contributions . . . 10

2 Preliminaries **12**
2.1 Nominal Syntax . . . 12
2.2 PVS . . . 16

3 Nominal Unification in PVS **23**
3.1 Specification . . . 24
3.1.1 Freshness and α-equivalence . . . 26
3.2 A Direct Formalisation of Transitivity of α-equivalence . . . 27
3.3 Minimal Freshness Contexts . . . 31
3.4 Nominal unification algorithm . . . 33

4 Ambiguity of Nominal Rules **40**
4.1 Nominal Rewriting . . . 40
4.2 Confluence of Nominal Rewriting . . . 45
4.2.1 Critical Pair Criterion and Orthogonality . . . 45
4.2.2 Criterion for α-stability . . . 48
4.3 Better Criteria for Confluence of Closed Rewriting . . . 49

5 Nominal Essential Intersection Types **56**
5.1 Types, ordering and operations . . . 56
5.2 Type Inference System and Basic Properties . . . 63
5.3 Typed Matching and Typed Rewrite Relation . . . 69
5.3.1 Subject Reduction . . . 76

6 Conclusions and Future Work **78**

References **84**

Acronyms

ASP Abstract Skeleton Preserving.

BCD Barendregt−Coppo−Dezani-Ciancaglini Intersection Type.

CP critical pair.

NRS Nominal Rewriting System.

PVS Prototype Verification System.

TCC Type Check Condition.

TRS Term Rewriting System.

Chapter 1

Introduction

Introducing variable binders in a language that works with names requires some mechanism to deal with α-equivalence, that is the invariance of objects modulo the renaming of bound variables. From logic, the existential and universal quantifiers are examples of constructors that need the binding engine to work. For instance, it must be possible to derive the equivalence between the formulas $\exists x : x > 1$ and $\exists y : y > 1$, despite the syntactical differences. Nominal theories treat binders in a way that is closer to informal practice, using variable names (or atoms), atom-permutations to proceed with renamings and freshness (constraints). This approach was presented in [Gabbay and Pitts, 1999], where the Fraenkel-Mostowski permutation model of set theory with atoms (FM-sets) is indicated as "the semantic basis of meta-logics for specifying and reasoning about formal systems involving name binding, α-conversion," etc.

First-order rewriting systems and the λ-calculus provide two useful notions of terms and reduction. However, both have limitations, which motivated extensions such as higher-order rewriting systems (see [Klop et al., 1993, Mayr and Nipkow, 1998]). Explicit substitution calculi are associated with higher-order rewriting systems, where substitutions are manipulated explicitly, and some of them use the de Bruijn indices to implement the substitution operation together with α-conversion in a first-order setting (see [Stehr, 2000]). In Nominal Rewriting Systems (NRS), we can specify capture-avoiding substitutions based on first-order techniques without the need to manage indices, since names and α-equivalence are primitive notions [Fernández and Gabbay, 2007].

Nominal rewriting generalises first-order rewriting by providing support for the specification of languages with binding operators. In nominal syntax, there are two kinds of variables: *atoms*, which are used to represent object-level variables and can be abstracted but not be substituted, and meta-variables, called simply *variables* or *unknowns*, which can be substituted but cannot be abstracted. Substitution of a variable by a term is closer to first-order substitution, where variables act as holes that

are supposed to be filled without worrying to adapt any part of the term, what can enable capture of atoms (unlike higher-order theories, where substitution is non-capturing). Another feature is that β-reduction is not a primitive notion in nominal rewriting, in contrast to the higher-order and explicit substitutions approaches (cf. [Huet, 1975, Dowek et al., 2000, Ayala-Rincón and Kamareddine, 2001]).

In the programming paradigm of rewriting, confluence and termination are computational properties that have a special role. Termination has to do with an algorithmic behavior of a program, i.e., the guarantee of a final answer for any input. Confluence refers to determinism of programs in the sense that, given an input, it does not matter the paths the program takes, it is always possible to have a common computation, even when the program does not terminate. Together, they are required properties of rewriting systems to decide equality modulo a related equational theory. This theme is well investigated in first-order systems [Baader and Nipkow, 1998] and the same result is achieved for closed systems in the context of nominal setting [Fernández and Gabbay, 2010]. Both properties are not always required in every rewriting system but, in general, one wants to be able to decide if a system is confluent/terminating or not. Unfortunately, such properties are undecidable in general; however, there are some criteria to guarantee them.

Such criteria for confluence of rewriting theories were first investigated in the context of the λ-calculus and abstract rewriting theories in works such as [Newman, 1942], in which the famous Newman's Lemma was stated: confluence and local confluence coincide for terminating rewriting theories. Nowadays this is seen as a combinatorial property of abstract rewriting theories that strictly depends on noetherianity, that is, well-foundedness of the rewriting relation [Huet, 1980].

In the context of Term Rewriting Systems (TRS), the Critical Pair Lemma, which is the kernel of the famous Knuth-Bendix completion procedure, guarantees local confluence of term rewriting theories [Knuth and Bendix, 1970], but to get confluence, in general, one needs to prove termination to the system. The most famous sufficient condition for confluence without termination, giving also rise to a programming discipline, is *orthogonality*. Essentially, orthogonality avoids indeterminism through two easily verifiable syntactic constraints on the rewrite rules: left-linearity, that constrains each variable occurring in the left-hand side of each rule to appear only once, and non-ambiguity, that constrains left-hand sides of rules to have no overlaps (except for trivial ones, at variable positions or between a rule and its copy at the root position). With these syntactic restrictions confluence of orthogonal rewriting theories is guaranteed [Rosen, 1973]. This result has been formalised for TRSs in a couple of proof assistants [Rocha-Oliveira et al., 2016, Thiemann, 2013].

For nominal rewriting theories, the Critical Pair Lemma and confluence of orthogonal theories were first investigated in [Fernández and Gabbay, 2007], where it was shown that the above-mentioned results extend to the class of uniform nominal rewriting theories, that is, theories where rules do not generate new atoms. More precisely, in [Fernández and Gabbay, 2007] it is shown that for the class of uniform theories, if all non-trivial critical pairs are joinable, then the theory is locally confluent, and therefore confluent if it is also terminating (by Newman's Lemma). Another sufficient condition for confluence of uniform theories is orthogonality: if the rules are left-linear and have no non-trivial critical pairs then the theory is confluent [Fernández and Gabbay, 2007]. As for first-order rewrite theories, trivial critical pairs are defined by overlaps at variable positions, or overlaps at the root between a rule and its copy. This way permuted variants of the same rule are not allowed to overlap in an orthogonal system. Both of these criteria rely on checking all non-trivial critical pairs. It is important to check also the overlaps at the root between a rule and its permuted variants, because if we miss those overlaps the theory might not be confluent (as it will be shown in Chapter 4).

In [Suzuki et al., 2015], the authors relax the orthogonality condition given in [Fernández and Gabbay, 2007] to permit overlaps at the root between a rule and its permuted copies, but only for uniform rules that satisfy an additional condition, called α-stability. This flexibility made the theorem of confluence for orthogonal systems more expressive, since it is possible to have orthogonal rules with bindings.

In this thesis, we present new criteria for (local) confluence of NRSs. Firstly, we show that also the conditions in the Critical Pair Lemma can be weakened if rules are uniform and α-stable: if all the non-trivial critical pairs are joinable, except possibly those caused by overlaps at the root between a rule and its permuted variants, then the theory is locally confluent. This condition was first explored in a previous paper developed during this research in [Ayala-Rincón et al., 2016a], but it was concomitantly studied in [Suzuki et al., 2016]. We develop it further by giving a new sufficient condition for α-stability, which is easy to check as it relies simply on nominal matching.

In addition, we give new stronger criteria for closed nominal rewriting: it is sufficient to check the generated overlaps using just one variant of each rule. The main advantage of working with closed rewriting is that it does not require the equivariant matching [Aoto and Kikuchi, 2016], as the standard nominal rewriting, which is exponential in time over the number of atoms occurring in rules. Instead, it uses nominal matching without equivariance, which can be solved in linear time on the size of the matching problem [Calvès and Fernández, 2010], making the rewrite step much more efficient.

Matching is actually a particular case of the equality problem of unification. Nominal unification is a problem that has been investigated since [Urban et al., 2004], where

a solution, whenever existent, is a pair formed by a set of freshness constraints with a first-order substitution and the unification algorithm provided is supposed to solve equality problems modulo α-equivalence. This first algorithm is exponential in time on the size of the problem , i.e., size of terms and freshness constraints, but in [Calvès, 2010, Levy and Villaret, 2010], this problem has been proved to be at most quadratic. Translations between nominal unification problems and higher-order pattern unification problems are given in [Cheney, 2005, Levy and Villaret, 2012].

There are also variations of the nominal syntax of expressions, as explored in [Schmidt-Schauss et al., 2016], what implies into a more complex analysis of the nominal unification problem; in that paper, it is proved to be NP-complete for a nominal grammar enhanced with recursive let.

In this thesis, we also present a specification of a nominal unification algorithm in the proof assistant Prototype Verification System (PVS) [Ayala-Rincón et al., 2016b] and the formalisation of its soundness and completeness. The decision to do it was motivated by the establishment of a nominal theory in PVS, which has a large library on TRSs, including the formalisation of the correctness of the Robinson's first-order unification algorithm [Avelar et al., 2014]. In rewriting, syntactic unification is a resource used, for instance, to identify overlaps between rewrite rules while matching, a restriction of unification, is used to perform rewrite steps, as mentioned before. That is why unification was the starting point to develop a nominal theory in PVS.

The following expressions contain X, Y as variables and i, k as atoms and can illustrate the way we are supposed to treat a nominal unification problem. The name k is bound in the first expression and i is bound in the second one by the sum operator:

$$\sum_{k=0}^{7}\sum_{i=0}^{5}(i - X)^i \text{ and } \sum_{i=0}^{7}\sum_{k=0}^{5}(X - Y)^k.$$

They admit a most general unifier with solution $[X \mapsto k][Y \mapsto i]$ according to the algorithm in [Urban et al., 2004], that inspired our specification. Note that i and k are captured. In a higher-order unification approach, this solution would not be accepted because bound variable capture is forbidden.

On the other hand, the unification problem with the expressions

$$\sum_{i=0}^{5}(i - X)^i \text{ and } \sum_{k=0}^{5}(X - Y)^k$$

has no solution in the nominal setting. One could argue that a solution could be obtained instantiating the terms with the substitution $[X \mapsto i][Y \mapsto i]$ and renaming k as i. But this is not possible since i should be a "fresh" name in the scope of the second sum in order

to proceed with this renaming, and the chosen substitution contradicts this condition. In other words, the meta-variable X should be instantiated uniformly throughout the problem.

The style of our specification is close to the functional presentations of Robinson's first-order unification algorithm, and the formalisation avoids the use of intermediate equivalence relations, obtaining in a straightforward manner transitivity and symmetry of the nominal α-equivalence relation. Indeed, in [Urban, 2010], a "weak equivalence" is used in order to simplify the proof of transitivity for the standard nominal α-equivalence. However, in this thesis, we present an even simpler proof, avoiding formalisations of properties of this weak intermediate relation. This is obtained following the analytic scheme of proof shown in [Fernández and Gabbay, 2007].

The nominal unification algorithm given in Isabelle/HOL in [Urban, 2004] is essentially specified as the transformation rule system presented in [Urban et al., 2004]. These rules transform unification problems with their associated freshness contexts into simpler ones. This approach is very elegant and allows a higher level of abstraction that simplifies the analysis of computational properties such as termination and uniqueness of solutions, but it is not so useful in implementations due to its inherent non-determinism (regarding the application of the transformation rules).

Here we present a new nominal unification algorithm that has only two nominal terms (but no freshness context) as inputs, as in [Calvès and Fernández, 2010, Levy and Villaret, 2010]. However, the algorithms presented in [Calvès and Fernández, 2010, Levy and Villaret, 2010] focus on efficiency, whereas our goal is to formalise the proof of correctness by specifying the algorithm in PVS as a recursive function "*unify*" that works directly on terms and deals with freshness contexts separately. Although the function "*unify*" does not carry freshness contexts, it builds them at the end of the execution together with the substitution solution. The freshness problems arisen during the recursive computation are solved separately due to the independence of solutions for freshness and without generating extra fresh atoms. This differs from the treatment given in [Levy and Villaret, 2010] where freshness constraints, as well as suspensions, are encoded as equations, that was proved equivalent to the treatment in [Calvès and Fernández, 2010] in [Calvès, 2013]. The main results regarding the nominal unification in PVS are also available in [Ayala-Rincón et al., 2016b].

Additionally, an important computational feature that has been investigated in nominal theories are type systems. At first, type theory was the answer to the problematic cases of paradoxes in set theory. In [Russell, 1908], it is identified the problem of self-reference in all known paradoxes and explicitly introduced the type theory in order to

eliminate them. In Computer Science, type systems are used to detect which programs are safe to be executed.

Types can be given in two styles: *à la* Church, where terms are explicitly decorated with types and only well-typed terms are allowed; and *à la* Curry, where terms from the type free syntax need a type inference to be classified as typable.In this thesis, we have chosen to present a type system in the second style due to its higher level of expressivity with respect to the first one. Indeed, in a type system *à la* Curry, we can consider terms that are not typable, but are valid terms in the grammar.

The Simple Type System for the λ-calculus is well investigated [Hindley, 2002], but despite its powerfulness and convenience, it has some drawbacks. It is known that typable terms in this system are strongly normalisable for $\beta\eta$-reduction. However, the opposite is not true: there are terms that are strongly normalisable but not typable in the system. For example, the term $\lambda x.x\,x$ that represents the self application function is in β-normal form and it has no type in the type system of [Hindley, 2002].

An intersection constructor was introduced to extend the Simple Type System in [Coppo and Dezani-Ciancaglini, 1978] and further improved in [Coppo et al., 1981, Barendregt et al., 1983], the famous Barendregt−Coppo−Dezani-Ciancaglini Intersection Type System (BCD). Intersection types provide a finitary polymorphism to terms and give a characterisation of terms that cannot originate infinite paths of reduction, i.e., $\beta\eta$-reduction is terminating in the class of terms typable with ω-free typings (ω is universal type) and, conversely, strongly normalisable terms are typable with ω-free typings. Moreover, there are interesting non-strongly normalisable terms that are typable with a type not equivalent to ω: the solvable terms, which are head-normalisable, such as the fixed point combinator λ-term $\lambda f.(\lambda x.f\ (x\ x))\ (\lambda x.f\ (x\ x))$.

In [van Bakel, 1995], a restriction of the BCD Type System [Barendregt et al., 1983] called Essential Intersection Type System was presented. It preserves the main properties of the BCD system, such as subject reduction for η-reduction, subject conversion of β-reduction, characterisation of head/strongly normalisable terms and the principal pair property. The Essential system is also an extension of the previous Strict Type System of [van Bakel, 1992] because typings are not closed under η-reduction in the latter. The main advantage of the Essential system regarding the BCD one is the reduced number of equivalent types to each term.

Some differences may be noticed in the nominal framework because substitutions are first-order, what means they allow capture of names. In the typed λ-calculus *à la* Curry, the well-known substitution lemma ensures that $\Gamma \vdash t[x \mapsto s] : \sigma$ whenever $\Gamma, x : \gamma \vdash t : \sigma$ and $\Gamma \vdash s : \gamma$ hold. This can be proved in this context because no free variable, atom in a nominal syntax, of s should be captured in this substitution action. In a nominal

system, one must be very carefully looking not only at this outermost environment Γ, but also at the type annotations in the leaves of the corresponding type derivation where the nominal variables are typed, because the free atoms that occur in the instance of such variables are possibly captured.

This thesis then presents an Essential Intersection Type System for nominal terms, which overcomes the specificities of the nominal framework as described in the previous paragraph, providing results of preservation of typings for α-equivalent terms and subject reduction for a notion of typed rewriting in uniform rewrite systems. Throughout the Chapter 5, examples are given to show the necessity of the conditions added in a typed matching, in the typed rewriting and, finally, in the theorem of subject reduction. The main ideas on the restrictions over the nominal typed rewriting were based on the presentation of the polymorphic nominal type system in [Fairweather and Fernández, 2016].

Objectives: the main objective of this thesis is the investigation of equational and computational properties on the nominal setting. More specifically, we could achieve results on the formalisation of properties about nominal unification, new theoretical results on confluence of NRSs (not formalised yet) and the development of an intersection type system for nominal terms and nominal rewriting.

1.1 Related work

There are formalisations of nominal theories in other proof assistants. The most famous formalisation has been implemented in Isabelle/HOL [Urban, 2008], where α-equivalence between terms is effectively obtained by representing terms as "abstraction functions". Thus, [Urban, 2008] presents some basic conditions that are sufficient to guarantee the equivalence between two representations of terms. Then, an induction principle is presented, to obtain proofs by induction over abstracted terms in a more natural way. For instance, the well-known Substitution Lemma in the context of the λ-calculus was formalised using these techniques.

A similar work was done in Coq [Aydemir et al., 2007], but bound variables were encoded by using de Bruijn indices and the terms were defined as having the type of locally nameless terms. An induction principle was implemented in order to prove properties about well-formed terms without mentioning indices. [Copello et al., 2016] also presents an α-structural induction principle in Agda and proves the Substitution Lemma using such an inductive scheme, but it criticises the use of higher-order features in [Urban, 2008] and the indices in [Aydemir et al., 2007] to represent bindings. Instead, the authors claim to

use a method similar to the Barandregt's Variable Convention, where a bound name is suposed to be chosen different from a given list of names.

Another formalisation in Isabelle/HOL is available in [Urban, 2004], to deal with nominal unification following [Urban et al., 2004]. This formalisation is closer to ours in the sense that α-equivalence is defined under some side-conditions, namely freshness conditions. The properties formalised in this system include the fact that the specified α-equivalence is indeed an equivalence relation, termination and soundness of the unification algorithm and the characterisation of normal forms generated by the algorithm.

In [Urban, 2010], the proof of transitivity of the α-equivalence relation is compared to the ones presented in [Urban, 2004, Fernández and Gabbay, 2007] and [Kumar and Norrish, 2010]. The proof in the latter was then considered the best one because it avoids a more complex inductive scheme on the size of terms. However, it requires the implementation of a "weak-equivalence" relation as a workaround. In the formalisation presented here, we follow auxiliary lemmas developed in [Fernández and Gabbay, 2007], but with a simpler proof of transitivity by induction on the structure of terms obtaining directly the result that the specified α-equivalence relation is indeed an equivalence relation.

Since the introduction of nominal unification and other features of nominal settings, some implementations have used them to enrich pre-existing programming languages, such as the logic tool of α-Prolog [Cheney and Urban, 2004], Cαml [Pottier, 2006] and FreshML [Shinwell et al., 2003], which aim for simplifying programming with binders and for eliminating the errors that may arise when using α-convertible expressions.

Concerning orthogonality, there are two notions in previous works for nominal rewriting. In [Fernández and Gabbay, 2007], orthogonality was left-linearity plus no non-trivial critical pairs. This was proved to be a sufficient condition for confluence of uniform rewrite rules. The notion of orthogonality was relaxed in [Suzuki et al., 2015] to allow overlaps at the root between permuted variants of rules. This weaker notion does not ensure confluence of uniform rules. If we also have α-stability then confluence is guaranteed [Suzuki et al., 2015]. A version of the Critical Pair Lemma was presented in [Fernández and Gabbay, 2007] too, where joinability of the non-trivial critical pairs was sufficient to prove local confluence of uniform NRSs. More recently, [Suzuki et al., 2016] extended the condition of α-stability for the Critical Pair Lemma, in order to eliminate the necessity to verify the joinability of a critical pair generated by a rule with its permuted version. The same result was achieved by us in [Ayala-Rincón et al., 2016a].

A sufficient condition for α-stability was given in [Suzuki et al., 2015], called "abstract skeleton preserving" (ASP). This is a strong restriction: it only allows identity permutations to be suspended on variables, and it requires the use of different atoms in nested

abstractions. Here we show that closedness, which does not impose such restrictions and can be checked simply by solving a nominal matching problem, is a sufficient condition for α-stability. In addition, for closed rewriting the criteria for confluence can be simplified, by checking only overlaps of freshened rules, i.e., with newly generated names with respect to the original rules and to the terms that are rewritten. Closedness and the ASP criterion are complementary in the sense that none of them implies the other. Our work in [Ayala-Rincón et al., 2016a] differs from [Suzuki et al., 2016] by the presentation of closedness as a condition for α-stability and by the treatment of the theorem of confluence of orthogonal systems and the Critical Pair Lemma for closed rewriting, where one does not need to worry about the overlap of permuted versions of rules. Besides, α-stability and uniformity come for free with closed rewriting, so that the mentioned theorems spare such conditions as premises.

With respect to type systems, other works have developed systems outside the context of the λ-calculus. In [van Bakel and Fernández, 1997], there is an Essential Intersection Type System for Curryfied TRSs. That work was based on the system presented in [van Bakel, 1995]. With a few restrictions on the rewrite rules, the authors were able to prove subject reduction for such systems. Additionally, they provided some criteria for the rules under which typability without ω implies strong normalisation. Despite intersection types are studied since the late 70's, a challenge is presented whenever one tries to move this kind of types into another calculus. For example, [Lengrand et al., 2004] proves the characterisation of strongly normalisable terms with intersection types in a composition-free calculus of explicit substitutions. In [Ventura et al., 2015], an intersection type system is presented for a few calculi written with de Bruijn indices and it shows subject reduction for all those calculi of explicit substitution.

In nominal context, [Fernández and Gabbay, 2006] defines a rank 1 polymorphic type system that explores, for the first time, the peculiarities of nominal with a syntax-directed type inference for nominal terms. A principal type function is presented that applies to a term with a type environment and a freshness context and returns the most general type for the given parameters. They were able to present a proof of subject reduction for typable rules with a specialised notion of rewrite step involving types. They already mentioned the intention of developing an intersection type system for nominal systems.

The thesis in [Fairweather, 2014] follows the presentation of [Fernández and Gabbay, 2006] and defines a simple type system *à la* Church, where α-equivalence and freshness need to be redefined to consider types, and other systems *à la* Curry, which are a simple type system, a polymorphic type system and a dependent type system. For this last system, another version of terms is given, where atom substitution is a primitive notion. The author has published a preliminar version of the polymorphic

system with Fernández and Gabbay in [Fairweather et al., 2011] and a posterior version of the dependent system with Fernández, Szasz and Tasistro in [Fairweather et al., 2015]. Typed nominal rewriting and nominal algebra, both in the Church and Curry styles, are presented in [Fairweather and Fernández, 2016], where conditions for subject reduction with dynamically and statically typed rewrite rules are given. The latest improves part of the work presented in [Fairweather, 2014].

Previous works like [Urban et al., 2004] present a *sort system* while defining nominal terms, but with the view to guarantee some level of well-formedness, instead of exploring the semantics provided by type systems. There, atoms are allowed to be typed only with atom sorts and only well sorted permutations are built (swappings must occur between atoms of the same sort). [Pitts, 2003] does a similar treatment regarding sorts, but over elements of nominal sets instead of a fixed grammar of nominal terms.

In [Cheney, 2009], a simple type system is presented for nominal abstract syntax, where the nominal semantics is added to the λ-calculus, with $\beta\eta$-reduction shown as a primitive notion. Using the same approach, [Cheney, 2012] and [Pitts et al., 2015] presented dependent type systems, where a dependent name-abstraction type constructor is used in the syntax of types.

To the best of our knowledge, our type system is the first one that works with intersections in the context of nominal terms. It is based on [van Bakel, 1995, van Bakel and Fernández, 1997] with respect to the intersection type features, and on [Fairweather, 2014, Fairweather and Fernández, 2016] regarding the nominal restrictions to obtain properties such as subject reduction.

1.2 Contributions

Summarising, the main contributions of this thesis are:

1. We specify the first basic nominal PVS *theory* where it is possible to find the notions of atoms, permutations, nominal terms, freshness, α-equivalence, substitutions and nominal unification.

2. We review and formalise the proofs of transitivity and symmetry of the α-equivalence relation, comparing to the formalisation in [Urban, 2004] and the proofs in [Urban, 2010]. See Section 3.2.

3. We show a functional presentation of the nominal unification algorithm similar to the Robinson's first-order unification algorithm based on the transformation rules in [Urban et al., 2004] and we formalise the termination, soundness and completeness of this algorithm in PVS. See Section 3.4.

4. We relax the conditions in the Critical Pair Lemma for uniform rules that are α-stable: it is not necessary to consider critical pairs generated by overlaps at the root between a rule and a permuted variant. See Subsection 4.2.1.[1]

5. We show that closedness is a sufficient condition for α-stability. Since closedness is easy to check by simply solving a nominal matching problem, we get an easy to check condition for α-stability. See Subsection 4.2.2.

6. We show that for closed rewriting, the criteria can be relaxed even more: it is sufficient to check overlaps by using one freshened version of each rule; overlaps between permuted variants of rules (at the root or otherwise) do not need to be considered at all. See Section 4.3.

7. We present an Essential Nominal Intersection Type System for nominal terms with the associated type operations such that typings are preserved for α-equivalent terms. See Section 5.2.

8. We present the notions of typed nominal matching and typed rewriting relation based on [Fairweather, 2014] with the proper adaptations to the intersection type system and the type operations introduced here. For this notion of rewriting, subject reduction holds for uniform rewrite systems. See Section 5.3.

[1]This result was independently obtained by T. Suzuki, K. Kikuchi, T. Aoto and Y. Toyama, "On confluence of nominal rewriting systems", 16th JSSST Workshop on Programming and Programming Languages, 2014, in Japanese.

Chapter 2

Preliminaries

This chapter presents in Section 2.1 the basic notions that will be used in the context of the nominal framework as well as the necessary background about PVS in order to understand the development of the *theory* of nominal unification in PVS in Section 2.2.

2.1 Nominal Syntax

We fix disjoint countably infinite collections of **atoms** (or names), **unknowns** (or variables), and **term-formers** (or function symbols). We write $\mathbb{A}$ for the set of atoms and $\mathbb{V}$ for the set of variables; $a, b, c, \ldots$ will range over distinct atoms. $X, Y, Z, \ldots$ will range over distinct unknowns. $\mathsf{f}, \mathsf{g}, \ldots$ will range over distinct term-formers. We assume that to each f is associated an **arity** $n \geq 0$. A **signature** Σ is a set of term-formers with their arities.

Definition 1 A **swapping** $(a\ b)$ is a bijection from $\mathbb{A}$ into $\mathbb{A}$ that exchanges a and b and fixes any other atom. **Permutations** are also bijections of the form $\pi : \mathbb{A} \to \mathbb{A}$, which change a finite number of atoms and that are represented as lists of swappings. The notation $\pi \circ \pi'$ is used for the **functional composition** of permutations, so $(\pi \circ \pi')(a) = \pi(\pi'(a))$. Then, the action of a permutation over atoms is recursively defined as:

$$id(c) = c \text{ , where } id \text{ is the null list;}$$

$$((a\ b) \circ \pi)(c) = \begin{cases} a, & \text{if } \pi(c) = b; \\ b, & \text{if } \pi(c) = a; \\ \pi(c), & \text{otherwise.} \end{cases}$$

The inverse of π is the reverse list of swappings and it is denoted by π^{-1}.

Definition 2 The set $\mathcal{T}(\Sigma, \mathbb{A}, \mathbb{V})$ of **(nominal) terms** is recursively defined by:

$$t \quad ::= \quad a \mid \pi \cdot X \mid [a]t \mid \mathsf{f}(t_1, \ldots, t_n) \text{ where } n \text{ is the arity of } \mathsf{f}.$$

Call $\pi \cdot X$ a **suspension** and $[a]t$ an **(atom-)abstraction**; it represents '$x.e$' or '$x.\phi$' in expressions like '$\lambda x.e$' or '$\forall x.\phi$'.

Next, it is presented a different grammar for nominal terms. It will be used in Chapter 3 for terms as specified in PVS.

Definition 3 The set $\mathcal{T}_{PVS}(\Sigma, \mathbb{A}, \mathbb{V})$ of **nominal terms** is generated by the following grammar:

$$t ::= \bar{a} \mid \pi \cdot X \mid () \mid (t_1, t_2) \mid [a]t \mid \mathsf{f}\ t\ ,$$

where $\bar{a}$ is an **atomic term**, $()$ is the **unit** or empty tuple, (t_1, t_2) is a **pair** of terms, and $\mathsf{f}\ t$ is an **application**.

This distinct presentation of terms follows the one taken in previous works such as [Fernández and Gabbay, 2007, Urban et al., 2004], for instance, and it was used to encode nominal terms while creating a PVS *theory* for nominal. Notice that, in the set $\mathcal{T}_{PVS}(\Sigma, \mathbb{A}, \mathbb{V})$, we distinguish between the atom a and the term $\bar{a}$ that consists of the atom a (compare with the constructor `at` in the code of the PVS data structure of terms presented in Chapter 3). Also, the function application works for symbols with arity one. To represent a greater arity, one can use pairs to encode tuples with any number of arguments. For instance, if the symbol f is supposed to have arity 3, then we can describe the term $\mathsf{f}\ (t_1, (t_2, t_3))$ using Definition 3. More details about the syntax of $\mathcal{T}_{PVS}(\Sigma, \mathbb{A}, \mathbb{V})$ will be explored in Chapter 3; wherever else, terms are supposed to be in the set $\mathcal{T}(\Sigma, \mathbb{A}, \mathbb{V})$.

Actions of permutations can be homomorphically extended over terms. This means that permutations only change atoms and are accumulated into suspensions. A precise definition is given below.

Definition 4 Define $\pi \bullet t$ a **permutation action** by:

$$\begin{array}{ll} \pi \bullet a = \pi(a) & \pi \bullet (\pi' \cdot X) = (\pi \circ \pi') \cdot X \\ \pi \bullet [a]t = [\pi(a)](\pi \bullet t) & \pi \bullet \mathsf{f}(t_1, \ldots, t_n) = \mathsf{f}(\pi \bullet t_1, \ldots, \pi \bullet t_n) \end{array}$$

Example 2.1. Let $\prod$ and $+$ (with infixed notation) be ternary and binary symbols of a signature Σ, respectively. Consider $\prod_{i=Y}^{Z} X$ the syntactic sugar of $\prod([i]X, Y, Z)$ and the

permutation $(m\,k) \circ (k\,n)$ with its inverse $(k\,n) \circ (m\,k)$. Next, one can observe the action of both permutations over the term $\prod_{i=m}^{k}(i + X)$:

$$(m\,k) \circ (k\,n) \bullet \prod_{k=m}^{n} (k + X) = \prod_{n=k}^{m} (n + (m\,k) \circ (k\,n) \cdot X)$$

$$(k\,n) \circ (m\,k) \bullet \prod_{k=m}^{n} (k + X) = \prod_{m=n}^{k} (m + (k\,n) \circ (m\,k) \cdot X).$$

One important observation is that the variables in suspensions work as meta-variables, where a substitution that replaces unknowns by terms is a primitive notion. With this in mind, it is reasonable that nominal variables are not 'abstractable'. The denomination 'suspension' for $\pi \cdot X$ has to do with the fact that the permutation π cannot indeed be applied to X until the instance of this variable is known; so it is suspended.

Definition 5 Define ${}^{\pi}t$ the **meta-action** of π on t by:

$${}^{\pi}a = \pi(a) \quad {}^{\pi}(\pi' \cdot X) = {}^{\pi}\pi' \cdot X \quad {}^{\pi}([a]t) = [\pi(a)]^{\pi}t \quad {}^{\pi}\mathsf{f}(t_1, \ldots, t_n) = \mathsf{f}({}^{\pi}t_1, \ldots, {}^{\pi}t_n),$$

where ${}^{\pi}id = id$ and ${}^{\pi}((a\ b) \circ \pi') = (\pi(a)\ \pi(b)) \circ {}^{\pi}\pi'$.

Extend the meta-action to contexts by ${}^{\pi}\nabla = \{\pi(a)\#X |\ a\#X \in \nabla\}$.

The meta-action of permutations affects only atoms in terms (it does not suspend on variables, in contrast to the permutation action of Definition 4).

Example 2.2. Consider the same signature of Example 2.1. Notice that the permutations do not suspend on unknowns when applying the meta-action of them.

$${}^{(m\,k)\circ(k\,n)} \prod_{k=m}^{n} (k + X) = \prod_{n=k}^{m} (n + X) \qquad {}^{(k\,n)\circ(m\,k)} \prod_{k=m}^{n} (k + X) = \prod_{m=n}^{k} (m + X).$$

Definition 6 A **substitution on unknowns**, ranged over $\theta, \vartheta, \mu, \ldots$, is a function from unknowns to terms with finite domain in the sense that they change only a finite set of unknowns. Each substitution θ is represented as a list of **nuclear substitutions**, which are pairs of the form $[X \mapsto s]$, and their **action** over terms is defined as:

$$a[X \mapsto s] = a \qquad (\pi \cdot X)[X \mapsto s] = \pi \bullet s$$
$$([a]t)[X \mapsto s] = [a](t[X \mapsto s]) \qquad (\pi \cdot Y)[X \mapsto s] = \pi \cdot Y$$
$$\mathsf{f}(t_1, \ldots, t_n)[X \mapsto s] = \mathsf{f}(t_1[X \mapsto s], \ldots, t_n[X \mapsto s])$$

We write id for the substitution when the set of changed unknowns is empty (it will always be clear whether we mean 'id the identity substitution' or 'id the identity permutation'). The juxtaposition of substitutions $\theta\theta'$ denotes the composition of the respective functions, mapping each X into $(X\theta)\theta'$. So, the action of θ over terms is defined inductively by:

$$t\,id = t \qquad\qquad t\,(\theta[X \mapsto s]) = (t\,\theta)[X \mapsto s].$$

Remark 2.3. This notion of substitution is different from the simultaneous application of nuclear substitutions; instead, we apply the nuclear substitutions one by one and allow to have non-idempotent substitutions. This approach is closer to triangular substitutions as explored in [Kumar and Norrish, 2010], with the intention to be more space efficient.

$$\frac{}{\nabla \vdash a\#b}\ (\#\mathbf{ab}) \qquad \frac{\pi^{-1}(a)\#X \in \nabla}{\nabla \vdash a\#\pi\cdot X}\ (\#\mathbf{X}) \qquad \frac{}{\nabla \vdash a\#[a]s}\ (\#[\mathbf{a}])$$

$$\frac{\nabla \vdash a\#s_1 \quad \ldots \quad \nabla \vdash a\#s_n}{\nabla \vdash a\#\mathsf{f}(s_1,\ldots,s_n)}\ (\#\mathsf{f}) \qquad \frac{\nabla \vdash a\#s}{\nabla \vdash a\#[b]s}\ (\#[\mathbf{b}])$$

$$\frac{}{\nabla \vdash a \approx_\alpha a}\ (\approx_\alpha\mathbf{a}) \qquad \frac{ds(\pi,\pi')\#X \subseteq \nabla}{\nabla \vdash \pi\cdot X \approx_\alpha \pi'\cdot X}\ (\approx_\alpha\mathbf{X})$$

$$\frac{\nabla \vdash s \approx_\alpha t}{\nabla \vdash [a]s \approx_\alpha [a]t}\ (\approx_\alpha[\mathbf{a}]) \qquad \frac{\nabla \vdash s \approx_\alpha (a\ b)\bullet t \quad \nabla \vdash a\#t}{\nabla \vdash [a]s \approx_\alpha [b]t}\ (\approx_\alpha[\mathbf{b}])$$

$$\frac{\nabla \vdash s_1 \approx_\alpha t_1 \quad \ldots \quad \nabla \vdash s_n \approx_\alpha t_n}{\nabla \vdash \mathsf{f}(s_1,\ldots,s_n) \approx_\alpha \mathsf{f}(t_1,\ldots,t_n)}\ (\approx_\alpha\mathsf{f})$$

Table 2.1: Freshness and α-equality

Definition 7 A **freshness (constraint)** is a pair $a\#t$ of an atom a and a term t. We call a freshness of the form $a\#X$ **primitive**, and a finite set of primitive freshnesses a **freshness context**. Δ and ∇ will range over freshness contexts.

We denote by $\nabla\theta$ the set $\{a\#\theta(X) |\ a\#X \in \nabla\}$ of freshness constraints.

A **freshness judgement** is a pair $\Delta \vdash a\#t$ of a freshness context and a freshness constraint. An **α-equivalence judgement** is a tuple $\Delta \vdash s \approx_\alpha t$ of a freshness context and two terms. The **derivable** freshness and α-equivalence judgements are obtained by the rules in Table 2.1, where $ds(\pi,\pi') = \{a \in \mathbb{A} \mid \pi(a) \neq \pi'(a)\}$. For A

a finite set of atoms, $A\#X$ denotes the freshness context $\{a\#X \mid a \in A\}$. We call $ds(\pi, \pi')$ the **difference set** of permutations π and π'.

Definition 8 The set $Pos(t)$ of **positions** of a term t is defined below. Note that ϵ is the only position in atoms and variables.

$$\frac{}{\epsilon \in Pos(t)}\ (\mathbf{p}_\epsilon) \qquad \frac{p \in Pos(t)}{1 \cdot p \in Pos([a]t)}\ (\mathbf{p}_{[\mathbf{a}]}) \qquad \frac{p \in Pos(t_i) \quad (1 \leq i \leq n)}{i \cdot p \in Pos(\mathsf{f}(t_1, \ldots, t_i, \ldots, t_n))}\ (\mathbf{p}_\mathsf{f})$$

The notation $t|_p$ represents the **subterm** of t **at position** p, which is defined by:

$$t|_\epsilon = t \qquad [a]t|_{1 \cdot p} = t|_p \qquad \mathsf{f}(t_1, \ldots, t_i, \ldots, t_n)|_{i \cdot p} = t_i|_p \ \ (1 \leq i \leq n)$$

If $p \in Pos(s)$, then $s[p \leftarrow t]$ denotes the replacement of $s|_p$ by t in s.

Definition 9 The function *atms* is used to compute the atoms in permutations and in terms. The set $atms(\pi)$ is defined by:

$$atms(id) \ = \ \varnothing \qquad atms((a\,b) \circ \pi) \ = \ \{a, b\} \cup atms(\pi).$$

The notations $atms(t)$ and $unkn(t)$ will be used to represent the set of atoms and unknowns in a term t, respectively. They are defined by:

$$\begin{aligned} atms(a) &= \{a\} & atms(\pi \cdot X) &= atms(\pi) \\ atms([a]t) &= atms(t) \cup \{a\} & atms(\mathsf{f}(t_1, \ldots, t_n)) &= \textstyle\bigcup_i atms(t_i) \\ unkn(a) &= \varnothing & unkn(\pi \cdot X) &= \{X\} \\ unkn([a]t) &= unkn(t) & unkn(\mathsf{f}(t_1, \ldots, t_n)) &= \textstyle\bigcup_i unkn(t_i) \end{aligned}$$

2.2 PVS

The Prototype Verification System (PVS) [Owre and Shankar, 1999, Shankar et al., 2001] is a proof assistant based on higher-order logic, i.e., that allows to specify predicates and fuctions over functions or relations. It is designed to specify and verify properties of programs. The functional specification of PVS, based on LISP, provides a code that is more readable and logically meaningful than in imperative languages. The type system

presented is a simple type system extended with subtypes (analogous to subset relation) and dependent types; the specification is grouped in *theories*, which can be parameterised, giving also some level of polymorphism.

The proof engine is used to formalise the lemmas specified by PVS users and also to complete the type check by proving the proof obligations of Type Check Conditions (TCCs), that are automatically generated, but not always automatically proven. This includes proofs of termination with respect to recursive definitions, that is, the user provides a measure that decreases in each recursive call and a TCC is generated to complete the type check.

The PVS proof theory is based on the sequent calculus of Gentzen. A sequent is of the form $\Sigma \vdash_\Gamma \Lambda$, where Γ is the context that contains the information of types and Σ and Λ are the antecedent and consequent sets of formulas, respectively. One must understand a sequent as the conjunction of the formulas in the antecedent implying the disjunction of the formulas in the consequent.

In this section, from now on, the PVS *theory* `TRS` [Avelar et al., 2014, Galdino and Ayala-Rincón, 2009] will be used to illustrate the PVS rules and some data structure. This *theory* includes the *subtheory* `orthogonality` developed in a previous work and presented in [Rocha-Oliveira et al., 2016], that is a formalisation of the well know theorem of confluence of orthogonal systems in the context of TRSs. As mentioned in the introduction, a similar result for Nominal Rewriting Systems is discussed in this thesis.

The next codes are the definitions of joinability and confluence of an arbitrary binary relation $\rightarrow$ in the context of abstract rewriting. Notice that $\rightarrow$ here is a PVS variable and `RTC`($\rightarrow$) is the reflexive transitive closure of $\rightarrow$.

```
- joinable?(→)(x,y): bool = EXISTS z: RTC(→)(x,z) & RTC(→)(y,z)
- confluent?(→): bool = FORALL x,y,z: RTC(→)(x,y) & RTC(→)(x,z) =>
                                      joinable?(→)(y,z)
```

In the *subtheory* `results_confluence` of `TRS`, there is a lemma called `R1_Confl_iff_R2_Confl` that ensures that confluence of a relation $\rightarrow_1$ holds if, and only if, $\rightarrow_2$ is confluent, given that `RTC`($\rightarrow_1$)=`RTC`($\rightarrow_2$).

```
- R1_Confl_iff_R2_Confl: LEMMA RTC(→1) = RTC(→2) =>
                          ( confluent?(→1) <=> confluent?(→2) )
```

Following, one of the sequents is presented. Observe that the variable x is declared with the product type `[T,T]`, so that it is actually a pair.

```
- R1_Confl_iff_R2_Confl :
 {-1} FORALL (x: [T, T]): RTC(→1)(x) = RTC(→2)(x)
```

```
  |----
[1] (confluent?(→1) <=> confluent?(→2))
```

In PVS, the negative formulas are the antecedent and the positive ones are the consequent. In this proof, since the information of the formula `-1` is used several times (with different instances), then it is possible to use the command `(copy)` in order to duplicate it. Notice that proofs in PVS are developed from the main goal to the leaves. This way, the proof commands are intended to perform the opposite of the corresponding proof rules of the sequent calculus, leading each branch into axioms. For instance, `(copy)` corresponds to the structural rule of contraction.

$$\frac{a, a, \Sigma \vdash_\Gamma \Lambda}{a, \Sigma \vdash_\Gamma \Lambda}\ (\mathbf{C}\vdash) \qquad\qquad \frac{\Sigma \vdash_\Gamma a, a, \Lambda}{\Sigma \vdash_\Gamma a, \Lambda}\ (\mathbf{C})$$

One may have irrelevant information in one sequent in order to prove a consequent formula. In the next sequent, for instance, only the formulas `-1`, `-6` and `-7` are necessary to prove `1`.

```
 - R1_Confl_iff_R2_Confl.1 :
  {-1} joinable?(R1)(y, z)
  [-2] RTC(R2)(x, y)
  [-3] RTC(R2)(x, z)
  {-4} RTC(R1)(x, y)
  {-5} RTC(R1)(x, z)
  [-6] FORALL (x:  [T, T]): RTC(R1)(x) = RTC(R2)(x)
  [-7] FORALL (x:  [T, T]): RTC(R1)(x) = RTC(R2)(x)
    |----
  [1] joinable?(R2)(y, z)
```

So, one can apply the command `(hide -2 -3 -4 -5)` to have a more succint presentation of the sequent. This corresponds to the structural inference rule of weakening, in the sequent calculus.

$$\frac{\Sigma_1 \vdash_\Gamma \Lambda_1}{\Sigma_2 \vdash_\Gamma \Lambda_2}\ (\mathbf{W})\ \text{if } \Sigma_1 \subseteq \Sigma_2 \text{ and } \Lambda_1 \subseteq \Lambda_2$$

Another interesting command is the `(case)` that represents one application of the Cut rule in the sequent calculus.

$$\frac{(\tau(\Gamma)(a) \sim \texttt{bool})_\Gamma \qquad \Sigma, a \vdash_\Gamma \Lambda \qquad \Sigma \vdash_\Gamma a, \Lambda}{\Sigma \vdash_\Gamma \Lambda}\ (\textbf{Cut})$$

In the Cut rule, $(\tau(\Gamma)(a) \sim \texttt{bool})_\Gamma$ asserts that the type information in the context Γ designates a boolean type to the expression a, i.e., indeed, a can be introduced as an additional formula into a sequent. The notation used in such expression about types in PVS can be found in [Owre and Shankar, 1999]. The proof of the next lemma uses the command `(case)`.

- `R1_R2_RTC_R1_R2:` `LEMMA union(`$\to_1$`,`$\to_2$`)` $\subseteq$ `RTC(`$\to_1$`)` $\circ$ `RTC(`$\to_2$`)) &`
  `RTC(`$\to_1$`)` $\circ$ `RTC(`$\to_2$`)` $\subseteq$ `RTC(union(`$\to_1$`,` $\to_2$`))`

The case introduced is `(case "`$\to_1 \subseteq$ `union(`$\to_1$`,`$\to_2$`)")`; the proof is then split into two branches and the formula can be used in the antecedent by one side and one has to prove it in the consequent on the other side. Sometimes, the information introduced is trivial or hardly used, so it can be proved directly in the same proof. However, when the complexity to prove some formula is high, then it is better to prove it in a separate lemma. Thus to use the separate result one must use the command `(lemma)` to import it as a formula, but the semantics of the command is the same as the `(case)`, that is, it also corresponds to the Cut inference rule.

As mentioned above, a proof branch is complete when an axiom is applicable to the sequent. In fact, there is no PVS command corresponding to the axiomatic rules of the sequent calculus of Gentzen, because the proof assistant automatically identifies such cases and closes the branch. The axiomatic rules are (**Ax**), $(\texttt{FALSE} \vdash)$ and $(\vdash \texttt{TRUE})$.

$$\frac{}{\Sigma, a \vdash_\Gamma a, \Lambda}\ (\textbf{Ax}) \qquad \frac{}{\Sigma, \texttt{FALSE} \vdash_\Gamma \Lambda}\ (\texttt{FALSE} \vdash) \qquad \frac{}{\Sigma \vdash_\Gamma \texttt{TRUE}, \Lambda}\ (\vdash \texttt{TRUE})$$

The rule (**Ax**) asserts that a formula a can be derived from a and $(\vdash \texttt{TRUE})$ realise the presence of `TRUE` in the disjunction of the formulas of the consequent. The rule $(\texttt{FALSE} \vdash)$ is the absurd rule and it claims that the absurd proves anything.

The elimination of `IF-THEN-ELSE` from formulas is unusual because of the semantics of `IF` a `THEN` b `ELSE` c: it represents a conjunctive formula $(a$ `IMPLIES` $b) \wedge (\neg a$ `IMPLIES` $c)$. When it is in the consequent, the treatment of the sequent calculus is preserved in the sense that the proof is split into two cases to prove the subformulas separately. However, when the `IF-THEN-ELSE` is in the antecedent, it takes into consideration the law of the excluded middle and it assumes that $a \vee \neg a$, splitting the proof with a and b in one branch

and $\neg a$ and c in the other. Notice that $\neg a$ appears as a in the opposite side: if it is in the antecedent, then a becomes a positive formula and vice versa.

$$\frac{\Sigma, a \vdash_{\Gamma,a} b, \Lambda \qquad \Sigma \vdash_{\Gamma,\neg a} a, c, \Lambda}{\Sigma \vdash_{\Gamma} \texttt{IF}\, a\, \texttt{THEN}\, b\, \texttt{ELSE}\, c, \Lambda} (\vdash \texttt{IF}) \qquad \frac{\Sigma, a, b \vdash_{\Gamma,a} \Lambda \qquad \Sigma, c \vdash_{\Gamma,\neg a} a, \Lambda}{\Sigma, \texttt{IF}\, a\, \texttt{THEN}\, b\, \texttt{ELSE}\, c \vdash_{\Gamma} \Lambda} (\texttt{IF} \vdash)$$

Equality is defined in terms of the rules of reflexivity and replacement of expressions. Transitivity and symmetry can be derived so. The notation $a[e]$ in rules (**Refl**) and (**Repl**) denotes the occurrence of an expression e in a without free variables and it can extend to sets of formulas such as $\Lambda[e]$.

$$\frac{}{\Sigma \vdash_{\Gamma} a = a, \Lambda} (\textbf{Refl}) \qquad \frac{a = b, \Sigma[b] \vdash_{\Gamma} \Lambda[b]}{a = b, \Sigma[a] \vdash_{\Gamma} \Lambda[a]} (\textbf{Repl})$$

The proof of `R1_Confl_iff_R2_Confl` applies the command `(replace)` to the next sequent, which is related to the rule (**Repl**). The aim is to replace $\texttt{RTC}(\to_1)(x,y)$ and $\texttt{RTC}(\to_1)(x,z)$ for $\texttt{RTC}(\to_2)(x,y)$ and $\texttt{RTC}(\to_2)(x,y)$ in the formula `-1`, respectively. Such information comes from formulas `-4` and `-5`.

```
 - R1_Confl_iff_R2_Confl.1 :
  [-1] RTC(R1)(x, y) & RTC(R1)(x, z) => joinable?(R1)(y, z)
  [-2] RTC(R2)(x, y)
  [-3] RTC(R2)(x, z)
  [-4] RTC(R1)(x, y) = RTC(R2)(x, y)
  {-5} RTC(R1)(x, z) = RTC(R2)(x, z)
  [-6] FORALL (x:  [T, T]): RTC(R1)(x) = RTC(R2)(x)
    |----
  [1] joinable?(R2)(y, z)
```

The extensionality rules are also equality rules to stablish the equivalence between two function or product expressions. The rule (**FunExt**) introduces a Skolem constant s such that it does not occur free in Γ in order to evaluate the equivalence of two functions f and g whenever applying both to an arbitrary argument produces the same result.

$$\frac{\Sigma \vdash_{\Gamma, s:A} f(s) =_{B[s/x]} g(s), \Lambda}{\Sigma \vdash_{\Gamma} f =_{x:A \to B} g, \Lambda} (\textbf{FunExt}) \quad \Gamma(s) \text{ is undefined}$$

For instance, the lemma `R1_equal_R2` proves the equivalence between two relations.

```
- R1_equal_R2:  LEMMA subset?(→1,→2) & subset?(→2,RTC(→1)) =>
                              RTC(→1) = RTC(→2)
```

To prove this lemma, one may use the PVS commands (decompose-equality) or (apply-extensionality) related to rule (**FunExt**) in order to introduce arbitrary arguments to RTC($\rightarrow_1$) and RTC($\rightarrow_2$), obtaining the following sequent.

```
- R1_equal_R2 :
  [-1] subset?(→1,→2)
  [-2] subset?(→2,RTC(→1))
    |----
  {1} RTC(→1)(x1, x2) = RTC(→2)(x1, x2)
```

Observe that x_1 and x_2 are new for the previous context. The same technique can be used for products. In this case, to prove the equivalence between two products, one might split the proof, proving the equivalence between the corresponding coordinates in separate branches.

PVS also provides a mechanism for defining abstract datatypes of a class that includes all the "tree-like" recursive data structures which are freely generated by a number of constructor operations [Owre and Shankar, 1997]. For example, the abstract datatype of lists is generated by the constructors null and cons. In the context of the *theory* TRS, terms are defined with the constructors vars and app, that corresponds to variables and functional terms (application of a function symbol to n arguments, where n is the arity of the symbol).

```
- term[variable: TYPE+, symbol: TYPE+, arity: [symbol -> nat]] : DATATYPE
  BEGIN
   vars(v: variable): vars?
   app(f:symbol, args:{args:finseq[term] | |args|=arity(f)}): app?
  END term
```

The proof assistant automatically generates an induction scheme that allows to develop inductive proofs on the structure of terms. Such scheme is presented in the next code, where P is a variable for an arbitrary predicate over terms. In the PVS *theory* nominal unification, inductive proofs on the structure of nominal terms were widely used.

```
- term_induction: AXIOM
  FORALL (P: [term -> boolean]):
    ((FORALL (v1: variable): P(vars(v1))) AND
         (FORALL (f: symbol, SEQ: {args: finseq[term] | |args|=arity(f)}):
                (FORALL (i: below[|SEQ|]): P(SEQ(i))) IMPLIES
                                             P(app(f, SEQ))))
     IMPLIES (FORALL (t: term): P(t))
```

To prove a property for all terms, it is sufficient to prove it in the base case for arbitrary variables and to make the induction step, proving the property for a functional term by assuming it for the arguments of such term.

Additionally, PVS provides a partial strict well founded ordering ≪ over the elements of the datatype. In the case of terms, it is defined as below. Consider `subterm(x,y)` to mean the usual relation of subterms over trees and read the above expression as x is a subterm of y (maybe equal).

```
- <<: (strict_well_founded?[term]) =
   LAMBDA (x, y: term):
    CASES y OF
          vars(v): FALSE,
          app(f, SEQ): EXISTS (i: below[|SEQ|]): subterm(x, SEQ(i))
    ENDCASES
```

This ordering is very useful to define recursive functions depending on the datatypes, since such definitions require a decreasing measure to prove termination of the recursive evaluation. The next chapter will explore more of such ordering.

Another feature of PVS is the support for dependent types, i.e., it is possible to build types which depend on elements. For instance, in the *theory* `TRS`, a type to represent all positions of a specific term is presented.

```
- positions?(t: term): TYPE = {p: position | positionsOF(t)(p)}
```

In the code above, `positionsOF(t)` is a recursive predicate that evaluates if a position is indeed a position of the term `t`.

Chapter 3

Nominal Unification in PVS

Nominal unification is a problem that has been investigated and proved to be solvable in quadratic time [Calvès, 2010]. Algorithms that solve this problem are important tools for nominal rewriting, since they allow to compute rewrite steps and to check ambiguity between rewrite rules, as it will be explored in Chapter 4.

In this chapter, we present a functional specification of a new nominal unification algorithm and the formalisation of its correctness and completeness in the language of the higher-order proof assistant PVS [Shankar et al., 2001]. PVS was chosen because it has a large library about term rewriting systems ([Galdino and Ayala-Rincón, 2009, Galdino and Ayala-Rincón, 2010]). In this way, our *theory* `nominal unification` extends this background about rewriting and it serves as a channel to spread the nominal setting among PVS users, providing formalised basic results in this approach.

The presentation of this formalisation is accompanied with the whole PVS development for nominal unification, which includes specifications of all notions and definitions as well as formalisations of the proofs of all lemmas and theorems given in this chapter. The development is available in the PVS theory for term rewriting systems `trs.cic.unb.br`.

As discussed in the introduction, this formalisation differs from others in the form of the unification algorithm specification, that does not carry freshness contexts as parameters and it is in a style closer to functional presentations such as Robinson's first-order unification algorithm, which in turn have been formalised in a variety of proof assistants (e.g., [Avelar et al., 2014, Paulson, 1985]). Also, some differences in the formalised proofs were found, specially when proving transitivity of the relation $\approx_\alpha$ under a freshness context.

3.1 Specification

This section will present details of the PVS specification for the *theory* `nominal unification`, such as the data structure for terms, action of permutations, action of substitutions, freshness and α-equivalence.

The next specification in PVS represents nominal terms and allows us to have syntactical induction schemes generated automatically. It follows the grammar proposed in Definition 3 of Section 2.1.

```
term[atom:TYPE+, perm:TYPE+, variable:TYPE+, symbol:TYPE+ ]:DATATYPE
  BEGIN
      at (a: atom): atom?
      * (p: perm, V: variable): susp?
      unit: unit?
      pair (term1: term, term2: term): pair?
      abs (abstr: atom, body: term): abs?
      app (sym: symbol, arg: term): app?
  END term
```

In PVS, names and variables were specified as natural numbers in order to stress the fact that the sets which contain them are countably infinite and symbols of functions are specified as strings. This does not create any ambiguity because atomic terms have the constructor `at` in the head and unknowns only occur accompained with a permutation, both as arguments of the constructor "`*`". One must not be tempted to think that this last constructor is a function which applies the permutation to the unknown in a term in the class of `(susp?)`. However, the PVS variables used in the specifications and proofs of the *theory* `nominal unification` use a more standard notation as in Chapter 2; for instance, `a, b, c` range over atoms, `X, Y` range over unknowns and `f, g` range over symbols.

Permutations are specified as lists of pairs of atoms. The function `act` applies a permutation to an atom by the recursive action of the swappings that represent the permutation.

```
 - perm: TYPE = list[[atom,atom]]
 - act(pi:perm)(c): RECURSIVE atom =
    CASES pi OF
        null: c,
        cons((a,b),rest): LET d = act(rest)(c) IN
                          IF d = a THEN b
                          ELSIF d = b THEN a ELSE d ENDIF
     ENDCASES
    MEASURE pi BY <<
```

Remark 3.1. The necessity of 'measure' functions in PVS's recursive functions is for proving termination according to the operational semantics of termination of PVS. This measure on the parameters should decrease after each recursive call. In the previous function `act` the measure '$\ll$' represents the standard measure on the data structures of permutations (i.e., lists). In the code presented next, the measure $\ll$ represents the subterm relation. In some cases, as for these functions, the system can automatically verify the decrement of the measure provided, what is not always the case, as for Definition 16 in Section 3.4.

The function `ext` extends the action of permutations to terms homomorphically, i.e., it applies `act` to atoms and accumulates permutations in suspensions.

```
- ext(pi:perm)(t:term): RECURSIVE term =
      CASES t OF
            at(a): at(act(pi)(a)),
            *(pm, v): *(append(pi, pm), v),
            unit: unit,
            pair(t1,t2): pair(ext(pi)(t1),ext(pi)(t2)),
            abs(ab, bd): abs(act(pi)(ab), ext(pi)(bd)),
            app(sl, ag): app(sl, ext(pi)(ag))
     ENDCASES
    MEASURE t BY <<
```

The function `subs` has a pair `(X,s)` as arguments and `subs(X,s)` represents a nuclear substitution, which can be applied to a term `t`, substituting all occurences of `X` by `s` in `t`.

```
- subs(X,s)(t): RECURSIVE term =
     CASES t OF
           at(a): at(a),
           *(pm, Y): IF X = Y THEN ext(pm)(s) ELSE pm * Y ENDIF,
           unit: unit,
           pair(t1, t2): pair(subs(X,s)(t1), subs(X,s)(t2)),
           abs(a, bd): abs(a, subs(X,s)(bd)),
           app(f, ag): app(f, subs(X,s)(ag))
    ENDCASES
   MEASURE t BY <<
```

Substitutions are specified as lists of pairs of variables and terms. `Subs` applies the nuclear substitutions of a substitution `theta` recursively to a term `t`.

```
- Subs(theta)(t): RECURSIVE term =
     CASES theta OF
           null : t,
           cons(head,tail): subs(head)(Subs(tail)(t))
     ENDCASES
     MEASURE theta BY <<
```

The next lemma states the invariance of alternating the application of a permutation and a substitution on a term. For the sake of simplicity, the notation assumed in Chapter 2 will be used here.

Lemma 3.2. *For any term t, $\pi \bullet (t\theta) = (\pi \bullet t)\theta$.*

Proof. By induction on the length of θ and using an auxiliary result in the induction step that proves the same for a nuclear substitution which, in turn, is proved by induction on the structure of t. □

3.1.1 Freshness and α-equivalence

Following, we present the notions of freshness and α-equivalence as done in PVS. Those functions relate to Definition 7, but they are specified separately.

The freshness contexts were specified as lists of pairs in the shape `[atom,variable]`; however we are preserving the notation of sets from the previous chapter for simplicity. So, $a\#X \in \nabla$ means that (a, X) occurs in the list ∇. Also, ∇, Δ and π denote the code for the PVS variables `Nabla`, `Delta` and `pi` and π^{-1} represents the reverse list of the permutation π.

Definition 10 **(Freshness)** Let ∇, a and t be a freshness context, an atom and a term, respectively. So, we define the following function as the freshness predicate in a context ∇.

```
fresh(∇)(a,t):  RECURSIVE bool =
   CASES t OF
         at(b):   a ≠ b,
         π∗X:  act(π⁻¹)(a)#X ∈ ∇,
         unit:  TRUE,
         pair(t1,t2):  fresh(∇)(a,t1) AND fresh(∇)(a,t2),
         abs(b,t'):  IF a = b THEN TRUE
                    ELSE fresh(∇)(a,t') ENDIF,
         app(f,t'):  fresh(∇)(a,t')
```

```
    ENDCASES
    MEASURE t BY <<
```

Notation: If ∇ and Δ are freshness contexts, then $\nabla \vdash \Delta$ means that $\Delta \subseteq \nabla$ and $\nabla\Delta$ denotes $\nabla \cup \Delta$.

Example 3.3. $\texttt{fresh}(\{a\#X\})(a, \mathsf{f}(\texttt{at}(b), \texttt{id}{*}X))$ returns `TRUE` while `FALSE` is the result of $\texttt{fresh}(\varnothing)(a, \mathsf{f}(\texttt{at}(a), \texttt{id}{*}X))$ because a is not fresh in $\texttt{at}(a)$ neither in $\texttt{id}{*}X$ with the empty context. Only one failure would be enough to have `FALSE` in the second problem.

Now, with the specifications of permutation and freshness, α-equivalence can be specified in a formal way, as well.

Definition 11 **(α-equivalence)** Let ∇, t and s be a freshness context and two terms, respectively. So, we define the next function as the relation of α-equivalence in the context ∇.

```
alpha(∇)(t,s):  RECURSIVE bool =
    CASES t OF
        at(a):  s = at(a),
        π*X:  susp?(s) AND X = V(s) AND ∀a ∈ ds(π,p(s)): a#X ∈ ∇,
        unit:  s = unit,
        pair(t1,t2):  pair?(s) AND alpha(∇)(t1,term1(s)) AND
                                   alpha(∇)(t2,term2(s)),
        abs(a,t'):  abs?(s) AND IF a = abstr(s) THEN alpha(∇)(t',body(s))
                              ELSE fresh(∇)(a,body(s)) AND
                                   alpha(∇)(t',(a abstr(s))•body(s)) ENDIF,
        app(f,t'):  app?(s) AND f = sym(s) AND alpha(∇)(t',arg(s))
    ENDCASES
    MEASURE t BY <<
```

Here $\texttt{ds}(\pi, \pi')$ denotes the set $\{b \in \mathbb{A} \mid \texttt{act}(\pi)(b) \neq \texttt{act}(\pi')(b)\}$ (the difference set as in Definition 7).

3.2 A Direct Formalisation of Transitivity of α-equivalence

This section presents in detail the proof of transitivity for the relation $\texttt{alpha}(\nabla)$ presented. As discussed before, this formalisation differs from a previous formalisation in Isabelle/HOL [Urban, 2004] and from an attempt to improve such formalisation [Urban, 2010]. To do so, some preliminar formalised results are presented now.

We consider the notation from Section 2.1, i.e., the specifications $\texttt{fresh}(\nabla)(a,t)$ and $\texttt{alpha}(\nabla)(t,s)$ are represented as $\nabla \vdash a\#t$ and $\nabla \vdash t \approx_\alpha s$, respectively.

The following two auxiliary lemmas express invariance of derivability in the previous calculus under the action of permutations and weakening of freshness contexts. They are basic results in the context of nominal and can be found in [Fernández and Gabbay, 2007], for example. The next lemma is proved in detail here in order to show the kind of inductive proof that we have developed.

Lemma 3.4. *Let ∇, a and t be a freshness context, an atom and a term, respectively. For all permutation π, we have $\nabla \vdash a\#t \Leftrightarrow \nabla \vdash \pi(a)\#\pi \bullet t$.*

Proof. It is proved by induction on the structure of t.

- $t = \bar{b}$: $\nabla \vdash a\#\bar{b}$ if and only if $a \neq b$. Also, $a \neq b$ if and only if $\pi(a) \neq \pi(b)$, because otherwise, $\pi^{-1}(\pi(a)) = \pi^{-1}(\pi(b))$ would hold. Finally, $\pi(a) \neq \pi(b)$ holds if and only if $\nabla \vdash \pi(a)\#\overline{\pi(b)}$.
- $t = \pi' \cdot X$: $\nabla \vdash a\#\pi' \cdot X$ if and only if $\pi'^{-1}(a)\#X \in \nabla$. Additionally, $(\pi \circ \pi')^{-1}(\pi(a)) = \pi'^{-1} \circ (\pi^{-1} \circ \pi)(a) = \pi'^{-1}(a)$. Therefore, $\pi^{-1}(a) \in \nabla$ iff $(\pi \circ \pi')^{-1}(\pi(a))\#X \in \nabla$ iff $\nabla \vdash \pi(a)\#(\pi \circ \pi') \cdot X$.
- $t = [b]t'$: If $a = b$, then $\nabla \vdash a\#[a]t'$ and $\nabla \vdash \pi(a)\#[\pi(a)]\pi \bullet t'$ trivially. Otherwise, by induction hypothesis (IH), $\nabla \vdash a\#t'$ iff $\nabla \vdash \pi(a)\#\pi \bullet t'$ iff $\nabla \vdash \pi(a)\#[\pi(b)]\pi \bullet t'$.
- For the cases $t = \mathsf{f}\, t'$ and $t = (t_1, t_2)$ the proof follows trivially by IH. The case $t = ()$ is even easier because the permutation has no effect over it and any atom is fresh in the unit.

□

In the formalisation itself, some extra details have to be treated such as the associativity of the action of permutations and the inverse character of the reverse list of a permutation. Such ordinary details are not considered here for brevity.

Lemma 3.5. *If $\Delta \vdash \nabla$ and $\nabla \vdash a\#t$, then $\Delta \vdash a\#t$.*

Proof. The proof is by induction on the structure of t, following the scheme similar to the one of the previous lemma. □

The next four auxiliary lemmas relate α-equivalence, freshness and the action of permutations. The first one expresses preservation of freshness by α-equivalent terms; the second one, alternation of the action of a permutation and its inverse on α-equivalent terms; the third one, invariance of α-equivalence under the action of a permutation; and, the fourth one, preservation of α-equivalence of a term under the action of permutations whose difference set is fresh in the term.

Lemma 3.6. $\nabla \vdash a\#s$ *and* $\nabla \vdash s \approx_\alpha t$ *implies* $\nabla \vdash a\#t$.

Lemma 3.7. $\nabla \vdash s \approx_\alpha \pi \bullet t \Rightarrow \nabla \vdash \pi^{-1} \bullet s \approx_\alpha t$.

Lemma 3.8. $\nabla \vdash s \approx_\alpha t \Leftrightarrow \nabla \vdash \pi \bullet s \approx_\alpha \pi \bullet t$.

Lemma 3.9. $\nabla \vdash ds(\pi_1, \pi_2)\#t$ *implies* $\nabla \vdash \pi_1 \bullet t \approx_\alpha \pi_2 \bullet t$.

Lemmas 3.6-3.9 are proved by induction on s, applying Lemma 3.4. For Lemma 3.7, Lemma 3.6 is applied.

The treatment of the results presented so far is the same as in previous papers ([Urban et al., 2004, Fernández and Gabbay, 2007, Urban, 2010]) and their complete formalisations are available in the accompanying PVS development.

The proof of the next lemma is shown in detail because, at this point, the formalisation differs from the one given in [Urban, 2004] and reported in [Urban, 2010]. The proof engine of PVS provides already the inductive scheme over the structure of nominal terms, which are specified as an abstract datatype presented at the beginning of Section 3.1. Such induction scheme is discussed at the end of Section 2.2 for terms in the *theory* `TRS`.

Lemma 3.10 (Transitivity of α-equivalence)**.** *The relation* $\approx_\alpha$ *is transitive under a given context* ∇*, i.e.,* $\nabla \vdash t_1 \approx_\alpha t_2$ *and* $\nabla \vdash t_2 \approx_\alpha t_3$ *imply* $\nabla \vdash t_1 \approx_\alpha t_3$.

Proof. The proof is by induction on the structure of t_1.

- $t_1 = \bar{a}$: then by definition of $\approx_\alpha$, $t_2 = t_3 = \bar{a}$.
- $t_1 = \pi_1 \cdot X$: so $t_2 = \pi_2 \cdot X$ and $t_3 = \pi_3 \cdot X$. We need to prove that $ds(\pi_1, \pi_3)\#X \subseteq \nabla$. So, take c such that $\pi_1(c) \neq \pi_3(c)$. There are two cases: if $\pi_1(c) = \pi_2(c)$, then $\pi_2(c) \neq \pi_3(c)$ and $c\#X \in \nabla$ for $ds(\pi_2, \pi_3)\#X \subseteq \nabla$; if $\pi_1(c) \neq \pi_2(c)$, then $c\#X \in \nabla$ because $ds(\pi_1, \pi_2)\#X \subseteq \nabla$.
- $t_1 = ()$ implies $t_2 = ()$ and $t_3 = ()$.
- $t_1 = (s_1, s_2)$: then $t_2 = (u_1, u_2)$ and $t_3 = (w_1, w_2)$. By induction hypothesis (IH), $\nabla \vdash s_1 \approx_\alpha w_1$ and $\nabla \vdash s_2 \approx_\alpha w_2$.
- $t_1 = \mathsf{f}\, s$: then $t_2 = \mathsf{f}\, u$ and $t_3 = \mathsf{f}\, w$. By IH, $\nabla \vdash s \approx_\alpha w$.
- $t_1 = [a]s$: then $t_2 = [b]u$ and $t_3 = [c]w$. It is necessary to compare the abstractors:
  - $a = b = c$: thus the result follows by IH trivially.
  - $a = b \neq c$: by definition, $\nabla \vdash s \approx_\alpha u$ and $\nabla \vdash u \approx_\alpha (b\ c) \bullet w$ and $\nabla \vdash b\#w$. By IH, $\nabla \vdash s \approx_\alpha (b\ c) \bullet w$. As $a = b$, then freshness condition is satisfied to a as well.

- $a \neq b = c$: we have $\nabla \vdash a\#u$, $\nabla \vdash s \approx_\alpha (a\ c) \bullet u$ and $\nabla \vdash u \approx_\alpha w$. By Lemma 3.8, $\nabla \vdash (a\ c) \bullet u \approx_\alpha (a\ c) \bullet w$ and, by IH, $\nabla \vdash s \approx_\alpha (a\ c) \bullet w$. By Lemma 3.4, $\nabla \vdash c\#(a\ c) \bullet u$ and $\nabla \vdash c\#(a\ c) \bullet w$ by Lemma 3.6. Finally, again by Lemma 3.4, $\nabla \vdash a\#w$.
- $b \neq a = c$: it is known that $\nabla \vdash s \approx_\alpha (b\ c) \bullet u$ and $\nabla \vdash u \approx_\alpha (b\ c) \bullet w$. Then, $\nabla \vdash (b\ c) \bullet u \approx_\alpha w$ by Lemma 3.7. By IH, $\nabla \vdash s \approx_\alpha w$.
- $a \neq b \neq c \neq a$: it is necessary to prove that $\nabla \vdash s \approx_\alpha (a\ c) \bullet w$ and $\nabla \vdash a\#w$. Let us prove first freshness: by definition of $\approx_\alpha$, $\nabla \vdash a\#u$ and $\nabla \vdash u \approx_\alpha (b\ c) \bullet w$. By Lemma 3.6, $\nabla \vdash a\#(b\ c) \bullet w$ and, by Lemma 3.4($\Leftarrow$), $\nabla \vdash a\#w$. Now lets prove α-equivalence: by hypothesis, $\nabla \vdash s \approx_\alpha (a\ b) \bullet u$, $\nabla \vdash u \approx_\alpha (b\ c) \bullet w$ and $\nabla \vdash b\#w$. By Lemma 3.8, $\nabla \vdash (a\ b) \bullet u \approx_\alpha (a\ b)(b\ c) \bullet w$. As $ds((a\ b)(b\ c), (a\ c)) = \{a, b\}$ and both atoms are fresh in w, then $\nabla \vdash (a\ b)(b\ c) \bullet w \approx_\alpha (a\ c) \bullet w$ by Lemma 3.9. Now, applying IH twice, one obtains $\nabla \vdash s \approx_\alpha (a\ c) \bullet w$.

□

Note that the critical point in this proof is the abstraction, particularly when all the abstractors differ. This is due to the asymmetry of rule ($\approx_\alpha[\mathbf{b}]$) in Table 2.1. The previous lemma was also presented in [Urban et al., 2004, Fernández and Gabbay, 2007], but in [Urban, 2010], a weak equivalence notion (Definition 12) is used as an intermediate relation to contour the problem with the abstraction case. However, auxiliary lemmas similar to the ones presented here were necessary in [Urban, 2010], in addition to other technical results to deal specifically with this weak equivalence (some of those additional lemmas in [Urban, 2010] are particular cases of transitivity). In the current formalisation, weak equivalence was not needed and the abstractions were treated as given in the five cases in the proof of Lemma 3.10.

Definition 12 (Weak-equivalence) Given two terms s, t, they are said to be **weak equivalent** (notation: $s \sim t$) whenever there exists a derivation of $s \sim t$ using the following rules:

$$\bar{a} \sim \bar{a}\ (\sim_{\mathbf{a}}) \qquad \frac{ds(\pi, \pi') = \varnothing}{\pi \cdot X \sim \pi' \cdot X}\ (\sim_{\mathbf{X}}) \qquad () \sim ()\ (\sim_{()})$$

$$\frac{s_1 \sim t_1 \quad s_2 \sim t_2}{(s_1, s_2) \sim (t_1, t_2)}\ (\sim_{\mathbf{pair}}) \qquad \frac{s \sim t}{[a]s \sim [a]t}\ (\sim_{[\mathbf{a}]}) \qquad \frac{s \sim t}{\mathsf{f}\ s \sim \mathsf{f}\ t}\ (\sim_{\mathsf{f}})$$

In the previous definition, observe that when $s \sim t$, then s and t differ only in possible representations of permutations π and π' in suspensions. Even so, the action of those permutations must be equal. Thus, the relation $\sim$ actually is closer to syntactic equality than to α-equivalence. To obtain transitivity of $\approx_\alpha$ using this definition, several auxiliary steps are necessary, among others, proving that $\sim$ is invariant under the action of permutations, preservation of freshness by weak-equivalent terms, etc. These lemmas are similar to the previously mentioned for $\approx_\alpha$. In addition, it is necessary to prove that, under a freshness context Δ, $(\approx_\alpha \circ \sim) \subseteq \approx_\alpha$, which is the key property for concluding transitivity of $\approx_\alpha$. All this work is unnecessary in our approach.

Lemma 3.11. (Equivalence) $\approx_\alpha$ *is an equivalence relation under any context* ∇*.*

Proof. Transitivity is guaranteed by Lemma 3.10. Reflexivity ($\nabla \vdash t \approx_\alpha t$) and symmetry ($\nabla \vdash t \approx_\alpha s$ implies $\nabla \vdash s \approx_\alpha t$) are easy to verify through an inductive proof on the structure of t. The interesting case is the proof of symmetry for abstractions with different abstractors. In this case, $\nabla \vdash [a]t' \approx_\alpha [b]s'$ means $\nabla \vdash t' \approx_\alpha (a\,b) \bullet s'$ and $\nabla \vdash a\#s'$. Applying $(a\,b)$ to the freshness, we obtain $\nabla \vdash b\#(a\,b) \bullet s'$ and, by Lemma 3.6, $\nabla \vdash b\#t'$. Now, by IH, $\nabla \vdash (a\,b) \bullet s' \approx_\alpha t'$ and, by Lemma 3.7, $\nabla \vdash s' \approx_\alpha (a\,b) \bullet t'$. This proves $\nabla \vdash [b]s' \approx_\alpha [a]t'$. ☐

Notice that, unlike the proofs given in [Urban et al., 2004, Urban, 2010], this formalised proof of symmetry does not use transitivity. Thus, these two properties are independent from each other.

3.3 Minimal Freshness Contexts

A solution for a unification problem is a pair (∇, θ) of a freshness context and a substitution (see Section 3.4). A nominal unification algorithm should generate "most general solutions" with respect to an ordering "$\leq$" as in the first-order case (see Definition 18). In the current formalisation, a function was specified that can compute a minimal freshness context ∇ which derives a freshness problem $a\#t$ when possible, i.e., $\nabla \vdash a\#t$ and ∇ is a subset of any other context Δ such that $\Delta \vdash a\#t$.

In the next function, the measure "<<" denotes the proper subterm relation that is generated by PVS when the abstract data structure specified for terms is type-checked. As for the example in Remark 3.1, termination with respect to this measure can be automatically verified.

Definition 13 Let a be an atom and t be a term. Define the function $\langle _\#_ \rangle_{sol}$ that takes as input the pair (a, t) and outputs a freshness context and a Boolean, as

follows:

$$
\begin{array}{lrl}
\langle a\#t\rangle_{sol} := & \mathit{CASES\ OF}\ t: & \\
 & \bar{b}: & (\varnothing, a \neq b), \\
 & \pi \cdot X: & (\{\pi^{-1} \bullet a\#X\}, \mathit{True}), \\
 & (): & (\varnothing, \mathit{True}), \\
 & (t_1, t_2): & \mathit{LET}\ (\Delta_1, \mathtt{b}_1) = \langle a\#t_1\rangle_{sol},\ (\Delta_2, \mathtt{b}_2) = \langle a\#t_2\rangle_{sol} \\
 & & \quad \mathit{IN\ IF}\ \mathtt{b}_1 = \mathtt{b}_2 = \mathit{True\ THEN}\ (\Delta_1\Delta_2, \mathit{True}) \\
 & & \qquad\quad \mathit{ELSE}\ (\varnothing, \mathit{False}), \\
 & [b]t': & \mathit{IF}\ a = b\ \mathit{THEN}\ (\varnothing, \mathit{True})\ \mathit{ELSE}\ \langle a\#t'\rangle_{sol}, \\
 & \mathsf{f}\ t': & \langle a\#t'\rangle_{sol} \\
\mathit{MEASURE} << & &
\end{array}
$$

The function above was taken from the transformation rules related to the unification algorithm in [Urban et al., 2004]. The difference is that here the freshness solutions are obtained separately from the substitutions which solve the equational problems in the unification algorithm. In this way, it is clear that the freshness constraints cannot modify the substitution that solves the problem, although they can restrict the validity of a unification problem.

The following lemma formalises the correctness of the previous definition.

Lemma 3.12 (Correctness of $\langle _\#_\rangle_{sol}$)**.** *Take* $(\Delta, \mathtt{b}) = \langle a\#t\rangle_{sol}$. *Then,*

(i) $\mathtt{b} = \mathit{True} \Rightarrow \Delta \vdash a\#t$, *and*

(ii) for any ∇, $\nabla \vdash a\#t \Rightarrow \mathtt{b} = \mathit{True}$ *and* $\nabla \vdash \Delta$.

Proof. The proof is by induction on the structure of t. The interesting case is when $t = (t_1, t_2)$, because in this case we need to have the same context in the derivations $\nabla \vdash a\#t_1$ and $\nabla \vdash a\#t_2$. However, the function $\langle _\#_\rangle_{sol}$ returns minimal contexts Δ_1 and Δ_2 to t_1 and t_2, respectively. For this reason, Δ_1 and Δ_2 have to be joined when computing $\langle _\#_\rangle_{sol}$. Then, using Lemma 3.5, it is possible to enlarge the contexts into the derivations $\Delta_1\Delta_2 \vdash a\#t_1$ and $\Delta_1\Delta_2 \vdash a\#t_2$ in order to derive the freshness for the pair. □

This function is crucial to build independently a freshness context for a whole nominal unification problem from its partial solutions, and it is used in the recursive treatment for the case of abstractions and pairs as it will be explained in the next section.

Notation: The function $\langle\cdot\rangle_{sol}$ can be generalised to sets of instantiated freshness constraints. In particular, $\langle \nabla\theta\rangle_{sol} = (\Delta, \mathit{True})$, where Δ is the union of all the freshness

contexts computed by $\langle a\#(id \cdot X)\theta\rangle_{sol}$, for each $a\#X \in \nabla$, if every subproblem is consistent, and $\langle\nabla\theta\rangle_{sol} = (\varnothing, False)$ otherwise.

The notation $\Delta \vdash \nabla\theta$ states that $\Delta \vdash a\#(id \cdot X)\theta$ is derivable for all $a\#X \in \nabla$.

3.4 Nominal unification algorithm

In order to construct a nominal unification algorithm as a recursive function in the specification language of PVS, it is necessary to provide a recognisable answer in cases of failure, because PVS does not allow partial functions. To deal with failure, our algorithm will return triplets of the form $(\nabla, \theta, \mathtt{b})$, which are a freshness context, a substitution and a Boolean, respectively, instead of pairs of the form (∇, θ). The triplet of the form $(\varnothing, Id, False)$ identifies failure cases and triplets of the form $(\nabla, \theta, True)$ successful cases with solutions of the form (∇, θ). For the sake of efficiency, in failure cases, the freshness context and the substitution are cleared into $\varnothing$ and Id respectively. If any branch fails, then it is not worth to carry partial solutions throughout recursive calls.

Definition 14 (Unifiable terms and unifiers) Two terms t, s are said to be **unifiable** if there exists a context ∇ and a substitution θ such that $\nabla \vdash t\theta \approx_\alpha s\theta$. Under these conditions, the pair (∇, θ) is called a **unifier** of t and s.

Definition 15 The **depth** of a term is computed by the following function:

$$depth(\bar{a}) = depth(\pi \cdot X) = depth(\,()\,) = 0 \qquad depth([a]t) = 1 + depth(t)$$
$$depth((t_1, t_2)) = 1 + max(depth(t_1), depth(t_2)) \qquad depth(\mathsf{f}\ t) = 1 + depth(t)$$

The function *depth* is used as component of the lexicographic measure provided to ensure termination of the nominal unification algorithm presented below.

Definition 16 (Nominal Unification Function) Let t, s be two nominal terms. Then, we define the function *unify* as below:

$$
\begin{array}{rl}
\mathit{unify}(t,s) := & IF\ s = \pi_s{\cdot}X_s\ AND\ X_s \notin \mathit{Vars}(t)\ THEN\ (\varnothing, [X_s \mapsto \pi_s^{-1} \bullet t], True) \\
& ELSE \\
& CASES\ OF\ (t,s): \\
(\pi_t \cdot X,\ \pi_s \cdot X): & (ds(\pi_t,\pi_s)\#X,\ Id,\ True), \\
(\pi_t \cdot X_t,\ s): & IF\ X_t \notin \mathit{Vars}(s)\ THEN\ (\varnothing, [X_t \mapsto \pi_t^{-1} \bullet s], True), \\
(\bar{a},\bar{a}): & (\varnothing,\ Id,\ True), \\
((),()): & (\varnothing,\ Id,\ True), \\
((t_1,t_2),(s_1,s_2)): & LET\ (\nabla_1,\ \theta_1,\ \mathtt{b}_1) = \mathit{unify}(t_1,s_1), \\
& \quad (\nabla_2,\ \theta_2,\ \mathtt{b}_2) = \mathit{unify}(t_2\theta_1,\ s_2\theta_1), \\
& \quad (\nabla_3,\ \mathtt{b}_3) = \langle \nabla_1\theta_2 \rangle_{sol} \\
& IN\ (\nabla_2\nabla_3,\ \theta_1\theta_2,\ \mathtt{b}_1 \wedge \mathtt{b}_2 \wedge \mathtt{b}_3), \\
([a]t',[b]s'): & IF\ a = b\ THEN\ \mathit{unify}(t',\ s') \\
& ELSE\ LET\ (\nabla_1,\ \theta,\ \mathtt{b}_1) = \mathit{unify}(t',\ (a\,b) \bullet s'), \\
& \quad (\nabla_2,\ \mathtt{b}_2) = \langle a\#s'\theta \rangle_{sol} \\
& IN\ (\nabla_1\nabla_2,\ \theta,\ \mathtt{b}_1 \wedge \mathtt{b}_2), \\
(\mathsf{f}\ t',\mathsf{f}\ s'): & \mathit{unify}(t',\ s'), \\
ELSE: & (\varnothing,\ Id,\ False) \\
& MEASURE\ lex(|\mathit{Vars}(t,s)|,\ \mathit{depth}(t))
\end{array}
$$

The measure function provided (see Remark 3.1) is lexicographic, with first component the number of variables in the unification problem and second component the *depth* of the first term of the unification problem.

The next remarks explain how the function $\langle _\#_ \rangle_{sol}$ correctly builds the necessary contexts for the abstraction and pair cases avoiding passing as parameter the freshness contexts, as done in unification mechanisms based on transformation rules (cf. [Urban, 2004]). In these remarks, terms are considered unifiable.

Remark 3.13. In case of pairs, $(\nabla_2\nabla_3, \theta_1\theta_2)$ has to be a unifier for (t_1, t_2) and (s_1, s_2), i.e., $\nabla_2\nabla_3 \vdash t_1\theta_1\theta_2 \approx_\alpha s_1\theta_1\theta_2$ and $\nabla_2\nabla_3 \vdash t_2\theta_1\theta_2 \approx_\alpha s_2\theta_1\theta_2$. Initially, *unify* builds the unifier (∇_1, θ_1) for t_1 and s_1. Afterwards, (∇_2, θ_2) is computed as a unifier for $t_2\theta_1$ and $s_2\theta_1$. If $\langle \nabla_1\theta_2 \rangle_{sol} = (\nabla_3, True)$, then $\nabla_1 \vdash t_1\theta_1 \approx_\alpha s_1\theta_1$ implies $\nabla_3 \vdash t_1\theta_1\theta_2 \approx_\alpha s_1\theta_1\theta_2$. Finally, since $\nabla_2 \vdash t_2\theta_1\theta_2 \approx_\alpha s_2\theta_1\theta_2$, weakening the contexts we obtain the desired unifier.

Remark 3.14. When unifying two abstractions with different abstractors, the answer $(\nabla_1\nabla_2, \theta)$ has to be a unifier for $[a]t$ and $[b]s$. Indeed, initially the recursive call $\mathit{unify}(t, (a\,b) \bullet s)$ provides a unifier (∇_1, θ) for this problem, if it is possible. Hence, $\nabla_1 \vdash t\theta \approx_\alpha (a\,b) \bullet s\theta$, but not necessarily ∇_1 would be able to derive $a\#s\theta$. Then, $\langle _\#_ \rangle_{sol}$ computes the minimal context ∇_2 which derives $a\#s\theta$ separately. Joining both contexts, the derivation $\nabla_1\nabla_2 \vdash [a]t\theta \approx_\alpha [b]s\theta$ can be completed.

Example 3.15. Take the problem of unifying (X, X) and $((a\,b) \cdot X, a)$. First, one unifies X and $(a\,b) \cdot X$. The result is the substitution Id and the context $\{a\#X, b\#X\}$. Then, to unify $X\,Id$ and $a\,Id$, we need the substitution $[X \mapsto a]$ and the empty context $\varnothing$. Then, $\{a\#X, b\#X\}$ is updated with $[X \mapsto a]$, and $\langle a\#a \rangle_{sol}$ returns failure.

Formalisation of termination of the function *unify* is not obtained automatically and requires human intervention to show that $lex(|\mathit{Vars}(t, s)|,\ \mathit{depth}(t))$ decreases in each recursive call. Observe that there are recursive calls in the cases of pairs, abstractions and applications. In the last two cases one advances on the structure of the first (and second) terms calling recursively a problem with the same number of variables, but smaller *depth*. The same happens for the first recursive call in the case of pairs. For the second recursive call of the case of pairs, when $\mathit{unify}(t_2\theta_1,\ s_2\theta_1)$ is computed, if $\theta_1 \neq Id$, the number of variables in the problem decreases for the nature of the nuclear substitutions generated in suspensions. So it is necessary to prove that the substitutions generated by *unify* have a special characterisation, as asserted in the next lemma.

Definition 17 (Type $\boldsymbol{Subs(s)}$ substitutions) The substitution $[X_1 \mapsto t_1] \ldots [X_n \mapsto t_n]$ is said to be of type $\mathit{Subs}(s)$ if

$$\bigcup_{i=1}^{n} \mathit{Vars}((X_i, t_i)) \subseteq \mathit{Vars}(s) \text{ and } X_i \notin \mathit{Vars}(t_i), \forall i = 1, \ldots, n.$$

Lemma 3.16 (Decrement of variables for substitutions of type $\boldsymbol{Subs(s)}$)**.** *Let θ be a substitution of type $\mathit{Subs}(s)$.*

(i) $\mathit{Vars}(t\theta) \subseteq \mathit{Vars}((t, s))$.

(ii) $\theta \neq Id$ *implies that* $|\mathit{Vars}(t\theta)| < |\mathit{Vars}((t, s))|$.

Proof. By induction on the length of θ.

(i) If $\theta = Id$, then obviously $\mathit{Vars}(t) \subseteq \mathit{Vars}((t, s))$. If $\theta = \theta'[X \mapsto u]$, then $t\theta = (t\theta')[X \mapsto u]$. By IH, $\mathit{Vars}(t\theta') \subseteq \mathit{Vars}((t, s))$. As $X \notin \mathit{Vars}(u)$, it is known

that $\mathit{Vars}(t\theta) = \mathit{Vars}(t\theta'[X \mapsto u]) = \mathit{Vars}((t\theta', u)) \setminus \{X\} \subseteq \mathit{Vars}((t, s)) \setminus \{X\} \subseteq \mathit{Vars}((t, s))$.

(ii) From (i), $\mathit{Vars}(t\theta'[X \mapsto u]) \subseteq \mathit{Vars}((t, s)) \setminus \{X\}$. Since $X \in \mathit{Vars}(s)$, the cardinality indeed decreases, i.e., $|\mathit{Vars}((t, s)) \setminus \{X\}| = |\mathit{Vars}((t, s))| - 1$.

$\square$

Lemma 3.17 (Type of substitutions built by *unify*)**.** *If* $\mathit{unify}(t, s) = (\nabla, \theta, \mathtt{b})$*, then the substitution* θ *is of type* $\mathit{Subs}((t, s))$*.*

Proof. This is easily checked observing the nuclear substitutions generated in the cases of suspensions. Note that, one condition to build $[X \mapsto \pi^{-1} \bullet u]$, for instance, is $X \notin \mathit{Vars}(u)$. $\square$

The last two lemmas ensure termination for the function $unify$:

Corollary 3.18 (Termination of *unify*)**.** *The function unify is total.*

Notation: It is said that $\Delta \vdash \theta \approx_\alpha \vartheta$ if, for any Y, $\Delta \vdash (id \cdot Y)\theta \approx_\alpha (id \cdot Y)\vartheta$, for substitutions θ and ϑ.

An auxiliary lemma regarding the action of α-equivalent substitutions over a term is necessary for the formalisation of the completeness of the unification algorithm and it is presented below.

Lemma 3.19. $\Delta \vdash \theta \approx_\alpha \vartheta$ *implies* $\Delta \vdash t\theta \approx_\alpha t\vartheta$*, for all term* t*.*

Proof. By induction on the structure of t. $\square$

The next results are the most difficult part of the formalisation (fully available at trs.cic.unb.br). Soundness and completeness formalisations follow the same inductive proof technique and the analysis of cases are also analogous. Thus, we focus only on completeness.

Lemma 3.20 (Soundness)**.** *Let* $(\nabla, \theta, \mathtt{b})$ *be the solution for* $\mathit{unify}(t, s)$*. If* $\mathtt{b} = True$*, then* (∇, θ) *is a unifier of* t *and* s*.*

Proof. The proof is by induction on $lex(|\mathit{Vars}((t, s))|, depth(t))$. It follows the cases distributed in the recursive definition of *unify* (Definition 16). $\square$

The previous lemma alone is not enough in the sense that, if the algorithm returns always $False$, then no unifier is provided, even to unifiable terms. The next theorem

guarantees that the algorithm actually gives a unifier whenever the terms are unifiable and the answer is the most general unifier.

Definition 18 (More general solutions) Let ∇, Δ be two contexts and ϑ, θ two substitutions. Then $(\nabla, \vartheta) \leq (\Delta, \theta)$ if there exists μ such that

$$\Delta \vdash \nabla\mu \text{ and } \Delta \vdash \vartheta\mu \approx_\alpha \theta.$$

If (∇, ϑ) is the least unifier for a unification problem according to "$\leq$", then it is a **most general unifier** (mgu).

Theorem 3.21 (Completeness). *Let* $(\nabla, \vartheta, \mathtt{b})$ *be the solution for* $\mathit{unify}(t, s)$. *If there exists any other solution* (Δ, θ) *for the unification problem, i.e.,* $\Delta \vdash t\theta \approx_\alpha s\theta$, *then* $\mathtt{b} = \mathit{True}$ *and* $(\nabla, \vartheta) \leq (\Delta, \theta)$.

Proof. The proof is by induction on $lex(|\mathit{Vars}(t, s)|,\ \mathit{depth}(t))$. There are some cases to consider: either t or s are suspensions or both have the same structure, that is, t and s are units or abstractions, for instance. That is due to the α-equivalence between $t\theta$ and $s\theta$ and the fact that θ cannot change the structure of a term, unless when acting over suspensions. The proof follows distinguishing cases according to the form (t, s). Below, we present the cases where s is a suspension, both are pairs, and both are abstractions; these are the most interesting cases.

- $(t, \pi \cdot X)$ and $X \notin \mathit{Vars}(t)$: so $\Delta \vdash t\theta \approx_\alpha (\pi \cdot X)\theta = \pi \bullet (X\theta)$ by Lemma 3.2. We need to prove $(\varnothing, [X \mapsto \pi^{-1} \bullet t]) \leq (\Delta, \theta)$. By definition of $\leq$, it is necessary to provide μ such that $\forall Y :\ \Delta \vdash Y[X \mapsto \pi^{-1} \bullet t]\mu \approx_\alpha Y\theta$. Instantiate it with θ.
  - $Y \neq X$ implies $\Delta \vdash Y[X \mapsto \pi^{-1} \bullet t]\theta = Y\theta \approx_\alpha Y\theta$.
  - $Y = X$: $\Delta \vdash t\theta \approx_\alpha \pi \bullet (X\theta)$ implies $\Delta \vdash \pi^{-1} \bullet (t\theta) \approx_\alpha X\theta$, by Lemma 3.7. As $X[X \mapsto \pi^{-1} \bullet t]\theta = \pi^{-1} \bullet t\theta$, the α-equivalence is derivable.

- $((t_1, t_2), (s_1, s_2))$: by hypothesis, $\Delta \vdash t_1\theta \approx_\alpha s_1\theta$ and $\Delta \vdash t_2\theta \approx_\alpha s_2\theta$.

  By IH, $\mathit{unify}(t_1, s_1) = (\nabla_1, \vartheta_1, \mathit{True})$ and $(\nabla_1, \vartheta_1) \leq (\Delta, \theta)$, i.e.,

  $$\text{there exists } \mu \text{ such that } \Delta \vdash \nabla_1\mu \text{ and } \Delta \vdash \vartheta_1\mu \approx_\alpha \theta.$$

  By Lemma 3.19, transitivity and symmetry, $\Delta \vdash t_2\vartheta_1\mu \approx_\alpha s_2\vartheta_1\mu$, that is, (Δ, μ) is a unifier for $t_2\vartheta_1$ and $s_2\vartheta_1$.

Using IH again, with $\mathit{unify}(t_2\vartheta_1, s_2\vartheta_1) = (\nabla_2, \vartheta_2, True)$, we obtain $\Delta \vdash \nabla_2\tilde{\mu}$ and $\Delta \vdash \vartheta_2\tilde{\mu} \approx_\alpha \mu$ for some $\tilde{\mu}$.

As $\mathit{unify}((t_1, t_2), (s_1, s_2)) = (\nabla_1\vartheta_2\nabla_2, \vartheta_1\vartheta_2, \mathtt{b})$, all we need to prove is that $\Delta \vdash \vartheta_1\vartheta_2\tilde{\mu} \approx_\alpha \theta$ and $\Delta \vdash \nabla_1\vartheta_2\tilde{\mu}$ (because $\Delta \vdash \nabla_2\tilde{\mu}$ follows by IH).

By Lemma 3.19, for any variable Y, it is possible to derive

$$\Delta \vdash (id \cdot Y\vartheta_1)\vartheta_2\tilde{\mu} \approx_\alpha (id \cdot Y\vartheta_1)\mu \approx_\alpha id \cdot Y\theta.$$

So, by transitivity, $\Delta \vdash \vartheta_1\vartheta_2\tilde{\mu} \approx_\alpha \theta$ holds.

Finally, as $\Delta \vdash \vartheta_2\tilde{\mu} \approx_\alpha \mu$ and $\Delta \vdash \nabla_1\mu$, then $\Delta \vdash \nabla_1\vartheta_2\tilde{\mu}$ by Lemmas 3.6 and 3.19.

- $([a]t', [b]s')$: by premisse, $\Delta \vdash a\#s'\theta$ and $\Delta \vdash t'\theta \approx_\alpha (a\ b) \bullet (s'\theta)$; by Lemma 3.2, the latter term is equal to $((a\ b) \bullet s')\theta$.

  By IH, $\mathit{unify}(t', (a\ b) \bullet s') = (\nabla_1, \vartheta, True)$ and $(\nabla_1, \vartheta) \leq (\Delta, \theta)$, i.e.,

  $$\text{there is } \mu \text{ such that } \Delta \vdash \nabla_1\mu \text{ and } \Delta \vdash \vartheta\mu \approx_\alpha \theta.$$

  By Lemma 3.6, $\Delta \vdash a\#s'\theta$ implies $\Delta \vdash a\#s'\vartheta\mu$. As μ cannot eliminate any inconsistency in "$a\#s'\vartheta$", then $\Delta \vdash a\#s'\vartheta$.

  By Lemma 3.12, as $\langle _\#_ \rangle_{sol}$ is complete, so $\langle a\#s'\vartheta \rangle_{sol} = (\nabla_2, True)$.

  Thus, the algorithm computes $\mathit{unify}([a]t', [b]s') = (\nabla_1\nabla_2, \vartheta, True)$. To show that $(\nabla_1\nabla_2, \vartheta) \leq (\Delta, \theta)$, we only need to see that $\Delta \vdash \nabla_2\mu$. Finally, since $(\nabla_2, True) = \langle a\#s'\vartheta \rangle_{sol}$ and $\Delta \vdash a\#s'\vartheta\mu$, then the result follows by Lemma 3.12.

□

Example 3.22. The notions of β and η-reduction for the λ-calculus can be defined using a nominal rewriting system [Fernández and Gabbay, 2007]. The formal notion of nominal rewriting will be introduced in the next chapter as well. In this example, the signature contains term-formers λ of arity 1, and *app* and *subst* of arity 2. Below, application is denoted by juxtaposition and $subst([a]X, Y)$ is written $X[a \mapsto Y]$ as usual (syntactic sugar). Freshness contexts are used in rewrite rules to express conditions on the matching

substitutions used to generate the rewrite relation.

$$\begin{array}{lrlcl}
(\mathsf{Beta}) & & \vdash\ (\lambda[a]X)Y & \to & X[a \mapsto Y] \\
(\mathsf{Eta}) & b\#Z & \vdash\ \lambda[b](Z\,b) & \to & Z \\
(\sigma_{app}) & & \vdash\ (XX')[a \mapsto Y] & \to & X[a \mapsto Y]X'[a \mapsto Y] \\
(\sigma_{var}) & & \vdash\ a[a \mapsto X] & \to & X \\
(\sigma_{lam}) & b\#Y & \vdash\ (\lambda[b]X)[a \mapsto Y] & \to & \lambda[b](X[a \mapsto Y]) \\
(\sigma_{\epsilon}) & a\#X & \vdash\ X[a \mapsto Y] & \to & X
\end{array}$$

To analyse one of the overlaps between (Beta) and (Eta), we can compute $\mathit{unify}((\lambda[a]X)Y, Z\,b) = (\varnothing, [Y \mapsto b][Z \mapsto \lambda[a]X], True)$ and apply the resulting substitution to the freshness context $\{b\#Z\}$, obtainig $(\{b\#X\}, True)$. In the case that the version $a\#Z \vdash \lambda[a](Z\,a) \to Z$ of (Eta) is chosen, then the solution of $\mathit{unify}((\lambda[a]X)Y, Z\,a)$ is $(\varnothing, [Y \mapsto a][Z \mapsto \lambda[a]X], True)$ and $\langle\{a\#Z\}[Y \mapsto a][Z \mapsto \lambda[a]X]\rangle_{sol} = (\varnothing, True)$.

Chapter 4

Ambiguity of Nominal Rules

In Chapter 3, we focused on the formalisation of properties related to unification modulo α-equality, i.e., the process of finding a substitution that makes two terms "equal" with respect to $\approx_\alpha$. On the other side, we may think under an equational reasoning, where sets of nominal axioms give rise to theories of equality for nominal terms. When introducing a rewrite theory that presents a nominal algebra, the semantical equality between terms can be analysed by verifying their normal forms in the rewriting system whenever it is terminating and confluent. That occurs because, under these two premises, equivalent elements have a unique normal form in such rewrite theory. This relation of nominal rewriting and nominal algebra is well explored in [Fernández and Gabbay, 2010].

This way, termination and confluence are essential properties to be explored because, despite their undecidability, there are conditions under which they can be guaranteed. In this chapter, our attention is devoted to investigate some criteria that ensure (local) confluence in nominal rewrite theories by examining overlaps of nominal rewrite rules. The differences regarding the same criteria in TRS's will be highlighted.

4.1 Nominal Rewriting

This section introduces the main concepts related with nominal rewriting, including the nominal rewriting relation itself, confluence, closedness of terms in context and rules and the closed rewriting relation. In the next chapter, another notion of typed rewriting relation will be introduced.

Definition 19 A **rewrite judgement** is a tuple $\nabla \vdash l \to r$ of a freshness context and two terms. We may write '$\varnothing \vdash$' as '$\vdash$'.

A **rewrite theory** $\mathsf{R} = (\Sigma, Rw)$ is a pair of a signature Σ and a possibly infinite set of rewrite judgements Rw in that signature; we call these **rewrite rules**.

A rewrite rule $\nabla \vdash l \to r$ is **left-linear** if each unknown occurs at most once in l.

The Example 3.22 at the end of the previous chapter is a set of left-linear rewrite rules.

The *equivariant closure* of a set of rules defined next is needed to generate the rewrite relation (Definition 21; see [Fernández and Gabbay, 2007, Fernández and Gabbay, 2010] for more details).

Definition 20 The **equivariant closure** of a set Rw of rewrite rules is the closure of Rw by the meta-action of permutations, that is, it is the set of all the permutative variants of rules in Rw. We write *eq-closure*(Rw) for the equivariant closure of Rw.

Below we write $\Delta \vdash (\phi_1, \ldots, \phi_n)$ for the judgements $\Delta \vdash \phi_1, \ldots, \Delta \vdash \phi_n$.

Definition 21 **Nominal rewriting:** Let $\mathsf{R} = (\Sigma, Rw)$ be a rewrite theory. The **one-step rewrite relation** $\Delta \vdash s \overset{\mathsf{R}}{\to} t$ is the least relation such that for every $(\nabla \vdash l \to r) \in Rw$, position p, permutation π, and substitution θ,

$$\frac{\Delta \vdash \left({}^{\pi}\nabla\theta, \quad s|_p \approx_\alpha {}^{\pi}l\theta, \quad s[p \leftarrow {}^{\pi}r\theta] \approx_\alpha t\right)}{\Delta \vdash s \overset{\mathsf{R}}{\to} t} \; (\mathbf{Rew}_{\nabla \vdash \mathbf{l} \to \mathbf{r}})$$

The notation $\Delta \vdash s \to_{\langle R,p,\pi,\theta \rangle} t$ highlights the fact that the rewrite step from s to t occurs with some specific rule R, position p, permutation π and substitution θ, under the freshness context Δ.

The **rewrite relation** $\Delta \vdash_{\mathsf{R}} s \to t$ is the reflexive transitive closure of the one-step rewrite relation, that is, the least relation that includes the one-step rewrite relation and such that:

- for all Δ and s: $\Delta \vdash s \approx_\alpha s'$ implies $\Delta \vdash_{\mathsf{R}} s \to s'$; and
- for all Δ, s, t, u: $\Delta \vdash_{\mathsf{R}} s \to t$ and $\Delta \vdash_{\mathsf{R}} t \to u$ implies $\Delta \vdash_{\mathsf{R}} s \to u$.

If $\Delta \vdash_{\mathsf{R}} s \to t$ holds, we say that s rewrites to t in the context Δ.

The rewrite relation is defined in a freshness context since it takes into account α-equivalence, which depends on freshness information for the term unknowns.

Example 4.1. Here some rules of the Example 3.22 are recalled. In the following rewrite theory, we can derive $\vdash_{\mathsf{R}} (\lambda[a]a)Y \to Y$ and also $a\#Z \vdash_{\mathsf{R}} (\lambda[a]Z)Y \to Z$.

$$
\begin{array}{lrlcl}
(\mathsf{Beta}) & & \vdash (\lambda[a]X)Y & \to & X[a \mapsto Y] \\
(\sigma_{app}) & & \vdash (XX')[a \mapsto Y] & \to & X[a \mapsto Y]X'[a \mapsto Y] \\
(\sigma_{var}) & & \vdash a[a \mapsto X] & \to & X \\
(\sigma_{lam}) & b\#Y & \vdash (\lambda[b]X)[a \mapsto Y] & \to & \lambda[b](X[a \mapsto Y]) \\
(\sigma_{\epsilon}) & a\#X & \vdash X[a \mapsto Y] & \to & X
\end{array}
$$

Definition 22 A rewrite theory R is **terminating** if there are no infinite rewriting sequences, i.e., there is no term in context from which infinite rewriting steps can be performed. It is **locally confluent** if $\Delta \vdash s \overset{\mathsf{R}}{\to} u$ and $\Delta \vdash s \overset{\mathsf{R}}{\to} v$ implies that there exists w such that $\Delta \vdash_{\mathsf{R}} u \to w$ and $\Delta \vdash_{\mathsf{R}} v \to w$. It is **confluent** when, if $\Delta \vdash_{\mathsf{R}} s \to t$ and $\Delta \vdash_{\mathsf{R}} s \to t'$, then there exists u such that $\Delta \vdash_{\mathsf{R}} t \to u$ and $\Delta \vdash_{\mathsf{R}} t' \to u$.

We call the situation $\Delta \vdash s \overset{\mathsf{R}}{\to} u$ and $\Delta \vdash s \overset{\mathsf{R}}{\to} v$ a **peak**.

Remark 4.2. Since the definition of the rewriting relation generated by a rewrite theory $\mathsf{R} = (\Sigma, Rw)$ takes into account permuted variants of rules (via the use of the permutation π in the one-step rewrite relation, see Definition 21), it is not necessary to include permuted variants of rules in Rw. For convenience, in the rest of the thesis we assume that for any $R \in Rw$, if R and ${}^{\pi}R$ are both in Rw then $\pi = id$; in other words, Rw does not contain permuted variants of the same rule.

According to Definition 21, to generate a rewrite step we need to solve an equivariant matching problem, that is, we need to find a permutation and a substitution such that $\Delta \vdash s|_p \approx_{\alpha} {}^{\pi}l\theta$. This problem is decidable, but exponential over the number of different atoms of the terms in context [Cheney, 2004]. However, for *closed rules* [Fernández and Gabbay, 2007], a simpler matching problem of the form $\Delta \vdash s|_p \approx_{\alpha} l\theta$, called nominal matching [Urban et al., 2004], suffices to generate the rewrite relation. Nominal matching is decidable and unitary [Urban et al., 2004] and efficient (it can be solved in linear time [Calvès and Fernández, 2010, Calvès, 2010]).

Closed rules roughly correspond to rules without free atoms, where rewriting cannot change the binding status of an atom. They are the counterpart of rules in standard higher-order rewriting formats (see [Domínguez and Fernández, 2014]). Below we first

recall the definition of nominal matching and then give a structural definition and an operational characterisation of closed terms.

Definition 23 A **term-in-context** is a pair $\Delta \vdash s$ of a freshness context and a term. A **nominal matching problem** is a pair of terms-in-context

$$(\nabla \vdash l) \; {}_?{\approx} \; (\Delta \vdash s) \quad \text{where} \quad \mathit{unkn}(\nabla \vdash l) \cap \mathit{unkn}(\Delta \vdash s) = \varnothing.$$

A **solution** to this problem is a substitution θ such that $\Delta \vdash \nabla\theta$, $\Delta \vdash l\theta \approx_\alpha s$, and $\mathit{dom}(\theta) \subseteq \mathit{unkn}(\nabla \vdash l)$.

The following structural definition of closedness follows [Clouston, 2007] and [Domínguez and Fernández, 2014].

Definition 24 Call a term-in-context $\Delta \vdash t$ **closed** when

1. every occurrence of an atom subterm a in t is under an abstraction of a;
2. if $\pi \cdot X$ occurs under an abstraction of $\pi \cdot a$ then any occurrence of $\pi' \cdot X$ occurs under an abstraction of $\pi' \cdot a$ or $a \# X \in \Delta$;
3. for any pair $\pi_1 \cdot X, \pi_2 \cdot X$ occurring in t, and $a \in ds(\pi_1, \pi_2)$, if neither $\pi_1 \cdot X$ nor $\pi_2 \cdot X$ occurs in the scope of an abstraction of $\pi_1 \cdot a$ or $\pi_2 \cdot a$, respectively, then $a \# X \in \Delta$.

Call $R = (\nabla \vdash l \to r)$ **closed** when $\nabla \vdash (l, r)$ is closed.[1]

It is easy to check whether a term is closed, using nominal matching and a freshened variant of the term [Fernández and Gabbay, 2007] (see Proposition 4.3 below).

Definition 25 A **freshened variant** $t^{\text{\tiny N}}$ of a nominal term t is a term with the same structure as t, except that the atoms and unknowns are replaced by 'fresh' atoms and unknowns with respect to t and to some atoms and unknowns from other syntax, which must always be specified. We omit an inductive definition.

Similarly, if ∇ is a freshness context then $\nabla^{\text{\tiny N}}$ denotes a freshened variant of ∇, i.e., if $a \# X \in \nabla$ then $a^{\text{\tiny N}} \# X^{\text{\tiny N}} \in \nabla^{\text{\tiny N}}$, where $a^{\text{\tiny N}}$ and $X^{\text{\tiny N}}$ are chosen fresh for the atoms and unknowns appearing in ∇.

We may extend this to other syntax, like equality and rewrite judgements.

Note that if $\nabla^{\text{\tiny N}} \vdash l^{\text{\tiny N}} \to r^{\text{\tiny N}}$ is a freshened variant of $\nabla \vdash l \to r$ then $\mathit{unkn}(\nabla^{\text{\tiny N}} \vdash l^{\text{\tiny N}} \to r^{\text{\tiny N}}) \cap \mathit{unkn}(\nabla \vdash l \to r) = \varnothing$.

Proposition 4.3. *A term-in-context $\nabla \vdash l$ is closed if, and only if, there exists a solution for the matching problem*

$$(\nabla^{\text{И}} \vdash l^{\text{И}}) \quad {}_?{\approx} \quad (\nabla, atms(\nabla^{\text{И}}, l^{\text{И}}) \# unkn(\nabla, l) \vdash l). \tag{4.1}$$

Due to the link between closedness of terms-in-context and solvability of a nominal matching problem, made explicit by the proposition above, the definition of closed rewriting (Definition 26) is based on nominal matching instead of using equivariant matching as in Definition 21.

Definition 26 Given a rewrite rule $R = (\nabla \vdash l \to r)$ and a term-in-context $\Delta \vdash s$, write $\Delta \vdash s \overset{R}{\to}_c t$ when there is some $R^{\text{И}}$ a freshened variant of R (so, fresh for R, Δ, s, and t), position p and substitution θ such that

$$\Delta, atms(R^{\text{И}}) \# unkn(\Delta, s, t) \vdash (\nabla^{\text{И}}\theta,\ s|_p \approx_\alpha l^{\text{И}}\theta,\ s[p \leftarrow r^{\text{И}}\theta] \approx_\alpha t). \tag{4.2}$$

We call this (one-step) **closed rewriting**.

The **closed rewrite relation** $\Delta \vdash_{\mathsf{R}} s \to_c t$ is the reflexive transitive closure of the one-step closed rewrite relation.

The previous definition is similar to Definition 21, but notice that the freshness context is extended.

Example 4.4. Any rule with free atoms, such as $\vdash \mathsf{f}(a, a) \to a$, is not closed (it is impossible to match it with a freshened variant). The rule $R = \vdash [a]\mathsf{f}(a, X) \to 0$ is closed, since taking a freshened version $R^{\text{И}} = \vdash [b]\mathsf{f}(b, Y) \to 0$, it is possible to solve the matching problem $(\ \vdash ([b]\mathsf{f}(b, Y), 0))\ {}_?{\approx}\ (b\#X \vdash ([a]\mathsf{f}(a, X), 0))$ with the substitution $\theta = [Y \mapsto (a\ b) \cdot X]$. Notice that $b\#X \vdash [b]\mathsf{f}(b, (a\ b) \cdot X) \approx_\alpha [a]\mathsf{f}(a, X)$.

We refer to [Fernández and Gabbay, 2007, Fernández and Gabbay, 2010] for more examples and properties of closed rewriting.

To compute overlaps of rules, a nominal unification algorithm may be used, as explored in Chapter 3.

Definition 27 A **nominal unification problem** is a set of freshness constraints and pairs of terms, written $\{a_1\#t_1, \ldots, a_k\#t_k, s_1\ {}_?{\approx}_?\ u_1, \ldots, s_m\ {}_?{\approx}_?\ u_m\}$. It is unifiable if there exists a **solution** $\langle \Gamma, \theta \rangle$ (freshness context and substitution) such that $\Gamma \vdash (a_1\#t_1\theta, \ldots, a_k\#t_k\theta, s_1\theta \approx_\alpha u_1\theta, \ldots, s_m\theta \approx_\alpha u_m\theta)$. In this case, $\langle \Gamma, \theta \rangle$ is said to be a **unifier** for the problem.

Compare this definition to Definition 14, where unification problems are defined only for equational constraints. In Definitions 16 and 13, there are algorithms to solve equality and freshness problems for a unification problem respectively.

4.2 Confluence of Nominal Rewriting

In this section we consider two well-known criteria for confluence of first-order rewriting based on the notion of overlapping rewrite steps [Baader and Nipkow, 1998]. They can be extended to nominal rewrite theories, but it is necessary to add some conditions.

4.2.1 Critical Pair Criterion and Orthogonality

The notion of overlap has been extended from the first-order setting to the nominal rewriting setting [Fernández and Gabbay, 2007]. In the first-order case, overlaps are computed by unification of a left-hand side of a rule R_1 with a non-variable subterm of the left-hand side of a rule R_2 (which could be a copy of R_1 with renamed variables, in which case the subterm must be strict, that is, overlaps at the root between a left-hand side and its copy are not considered). With nominal rules the nominal rewrite relation is generated by the *equivariant closure* of a set of rules (see Definitions 20 and 21) so we must consider permuted variants of rules, and use nominal unification instead of first-order unification. This is Definition 28, which follows [Fernández and Gabbay, 2007]:

Definition 28 (Overlaps and CPs) Let $R_i = \nabla_i \vdash l_i \to r_i$ $(i = 1, 2)$ be copies of rewrite rules in *eq-closure*(Rw) (so R_1 and R_2 could be copies of the same rule), where $unkn(R_1) \cap unkn(R_2) = \varnothing$, as usual. If the nominal unification problem $\nabla_1 \cup \nabla_2 \cup \{l_2 \;{}_?{\approx}_?\; l_1|_p\}$ has a most general solution $\langle \Gamma, \theta \rangle$ for some position p, then we say that R_1 **overlaps** with R_2, and we call the pair of terms-in-context $\Gamma \vdash (r_1\theta, l_1\theta[p \leftarrow r_2\theta])$ a **critical pair**. If p is a variable position, or if R_1 and R_2 are identical modulo renaming of variables and $p = \epsilon$, then we call the overlap and critical pair **trivial**, otherwise we call it **non-trivial**.

The critical pair $\Gamma \vdash (r_1\theta, l_1\theta[p \leftarrow r_2\theta])$ is **joinable** if there is a term u such that $\Gamma \vdash_{\mathsf{R}} r_1\theta \to u$ and $\Gamma \vdash_{\mathsf{R}} (l_1\theta[p \leftarrow r_2\theta]) \to u$.

We distinguish between different kinds of overlaps and critical pairs:

Definition 29 (Permutative Overlaps and CPs) Let $R_i = \nabla_i \vdash l_i \to r_i$ $(i = 1, 2)$ be copies of rewrite rules in *eq-closure*(Rw), such that there is an overlap. If R_2 is a copy of ${}^{\pi}R_1$, we say that the overlap is **permutative**. We call a permutative

overlap at the root position **root-permutative**. We call an overlap that is not trivial and not root-permutative **proper**. We use the same terminology to classify critical pairs; e.g. we call a critical pair generated by a permutative overlap **permutative**.

A permutative overlap indicates that there is a critical pair generated by a rule and one of its permuted variants.

Note that only the root-permutative overlaps where π is id are trivial. While overlaps at the root between variable-renamed versions of first-order rules can be discarded (they generate equal terms), in nominal rewriting we must also consider overlaps at the root between permuted variants of rules. Indeed, they do not necessarily produce the same result, as the following example shows (see also [Suzuki et al., 2015]).

Example 4.5. Consider $R = (\vdash f(X) \to f([a]X))$. There is an overlap at the root between this rule and its variant ${}^{(a\ b)}R = (\vdash f(X) \to f([b]X))$, i.e., a root-permutative overlap, which is not trivial. It generates the critical pair $\vdash (f([a]X), f([b]X))$. Note that the terms $f([a]X)$ and $f([b]X)$ are not α-equivalent. This theory is not confluent; we have for instance:

$$\begin{array}{ccc} & f(a) & \\ {\scriptstyle \langle R,\epsilon,id,[X\mapsto a]\rangle}\swarrow & & \searrow{\scriptstyle \langle R,\epsilon,(a\ b),[X\mapsto a]\rangle} \\ f([a]a) & \not\approx_\alpha & f([b]a) \end{array}$$

Definition 30 introduces *uniformity.* In [Fernández and Gabbay, 2007] a Critical Pair Lemma was proved for uniform nominal rewrite theories, that joinability of non-trivial critical pairs implies local confluence; confluence follows by Newman's Lemma if the theory is terminating. Uniformity features in this chapter in Theorem 4.6. Intuitively, uniformity means that if a is not free in s and s rewrites to t then a is not free in t.

Definition 30 (Uniformity) We call a nominal rewrite theory $\mathsf{R} = (\Sigma, Rw)$ **uniform** [Fernández and Gabbay, 2007] when if $\Delta \vdash_{\mathsf{R}} s \to t$ and $\Delta, \Delta' \vdash a\#s$ for some Δ', then $\Delta, \Delta' \vdash a\#t$.

Note that in the Critical Pair Lemma of [Fernández and Gabbay, 2007], joinability is assumed for all non-trivial critical pairs. Joinability of proper critical pairs is insufficient for local confluence, even for a uniform theory: the rule in Example 4.5 is uniform. However, an additional condition allows us to prove that uniform rewrite theories with joinable proper critical pairs are locally confluent. Recall the notion of α-stability

from [Suzuki et al., 2015]:

Definition 31 **(α-stability)** Call a rewrite rule $R = \nabla \vdash l \to r$ **α-stable** when, for all $\Delta, \pi, \theta, \theta'$, $\Delta \vdash \nabla\theta, {}^{\pi}\nabla\theta', l\theta \approx_{\alpha} {}^{\pi}l\theta'$ implies $\Delta \vdash r\theta \approx_{\alpha} {}^{\pi}r\theta'$.

A rewrite theory $\mathsf{R} = (\Sigma, Rw)$ is **α-stable** if every rule in Rw is α-stable.

α-stability is hard to check in general because of the quantification over all θ and θ'. α-stability is related to *closedness* (Definition 24): we show in Section 4.2.2 that closed rules are α-stable. The reverse implication does not hold: for example $\vdash f(a) \to a$ is α-stable but not closed.

Theorem 4.6 (Critical Pair Lemma for uniform α-stable theories)**.** *Let* $\mathsf{R} = (\Sigma, Rw)$ *be a uniform rewrite theory where all the rewrite rules in Rw are α-stable. If every proper critical pair is joinable, then* R *is locally confluent.*

Proof. We consider cases. There are four kinds of peaks:

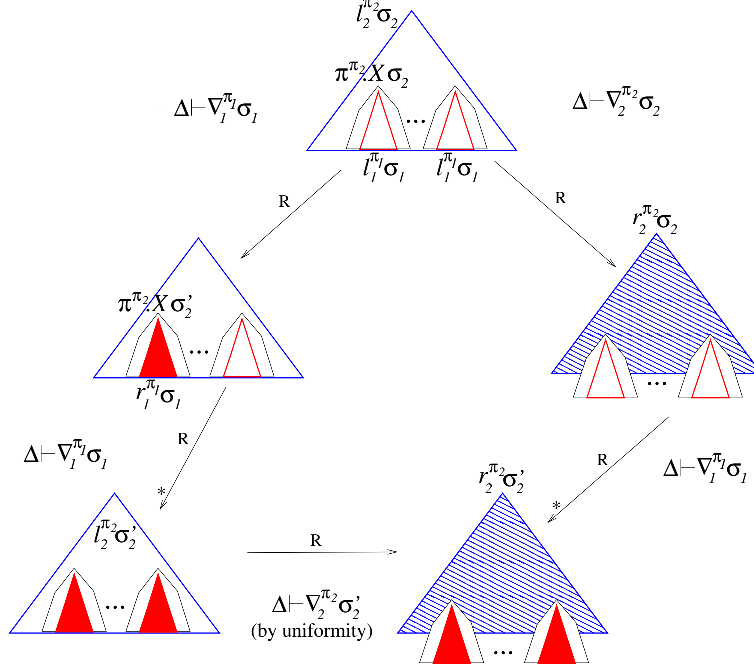
- *If the rewrite steps occur at disjoint positions,* then the peak is trivially joinable by applying the same rules, permutations and substitutions.
- *If the peak is an instance of a proper critical pair* (joinable by assumption), then it is joinable since rewriting is compatible with instantiation [Fernández and Gabbay, 2007, Theorem 49].
- *If the peak is generated by an overlap at a variable position,* without loss of generality assume $\nabla \vdash s \approx_{\alpha} {}^{\pi_1}l_1\theta_1$ and s occurs inside ${}^{\pi_2}l_2\theta_2$ under an instance of an unknown $({}^{\pi_2}\pi \cdot X)\theta_2$ (see Figure 4.1). Then we can change the action of θ_2 over X, replacing s by t, such that $\nabla \vdash t \approx_{\alpha} {}^{\pi_1}r_1\theta_1$, as it is done in the first-order case. Here we rely on uniformity to ensure that no free atoms are introduced by the rewrite step, so freshness constraints are preserved when replacing s by t.
- *If there is a root-permutative overlap* then joinability follows by α-stability. □

Definition 32 Call a rewrite theory $\mathsf{R} = (\Sigma, Rw)$ **orthogonal** when all the rules in Rw are left-linear and there are no non-trivial critical pairs.

Call $\mathsf{R} = (\Sigma, Rw)$ **quasi-orthogonal** when all rules are left-linear and there are no proper overlaps.

So orthogonal theories are left-linear and can have trivial overlaps only, whereas quasi-orthogonal theories are left-linear and can have trivial overlaps and root-permutative overlaps (Definition 29).

Figure 4.1: Critical Pair Lemma - case of overlap at a variable position



Orthogonal theories were defined in [Fernández and Gabbay, 2007]. Quasi-orthogonal theories were defined in [Suzuki et al., 2015] and called orthogonal (we changed the name here to avoid confusion). Orthogonality implies confluence for uniform nominal rewrite theories [Fernández and Gabbay, 2007]. Quasi-orthogonality is insufficient for confluence of uniform theories; see Example 4.5. If a theory is uniform, quasi-orthogonal, and α-stable, then it is confluent [Suzuki et al., 2015].

4.2.2 Criterion for α-stability

This section presents closedness as a sufficient condition for α-stability. Closedness is easy to check using a nominal matching algorithm (see Proposition 4.3).

An easy technical lemma will be useful, that substitutions that coincide modulo α on the unknowns in a term yield α-equivalent instances, and vice-versa (i.e., if the instances are α-equivalent, the substitutions must coincide modulo α on the unknowns of the term):

Lemma 4.7. $\Delta \vdash t\theta \approx_\alpha t\theta' \Leftrightarrow \forall X \in unkn(t).\Delta \vdash X\theta \approx_\alpha X\theta'$.

The direction ($\Leftarrow$) of Lemma 4.7 is a particularisation of Lemma 3.19 because only the variables in t are required to have the same image by both substitutions. It is also available in [Fernández and Gabbay, 2010] (Lemma 5.12).

Theorem 4.8. *If R is a closed rule, then R is α-stable.*

Proof. It is sufficient to prove the following property: $R = \nabla \vdash l \to r$ closed, $\Delta \vdash s \approx_\alpha l\vartheta \to r\vartheta$ and $\Delta \vdash s \approx_\alpha {}^\pi l\vartheta' \to {}^\pi r\vartheta'$ imply $\Delta \vdash r\vartheta \approx_\alpha {}^\pi r\vartheta'$.

The matching problems $(\nabla^{\text{N}} \vdash (l^{\text{N}}, r^{\text{N}}))\ {}_?{\approx}\ (\nabla, atms(R^{\text{N}})\#unkn(R) \vdash (l,r))$ and $(\nabla^{\text{N}} \vdash (l^{\text{N}}, r^{\text{N}}))\ {}_?{\approx}\ ({}^\pi\nabla, atms(R^{\text{N}})\#unkn(R) \vdash ({}^\pi l, {}^\pi r))$ are solvable with solutions θ and ${}^\pi\theta$, respectively, insofar as R is closed. Hence, we can infer:

- $\nabla, atms(R^{\text{N}})\#unkn(R) \vdash \nabla^{\text{N}}\theta,\ (l^{\text{N}}\theta, r^{\text{N}}\theta) \approx_\alpha (l, r)$
- ${}^\pi\nabla, atms(R^{\text{N}})\#unkn(R) \vdash \nabla^{\text{N}\pi}\theta,\ (l^{\text{N}\pi}\theta, r^{\text{N}\pi}\theta) \approx_\alpha ({}^\pi l, {}^\pi r)$
- $\Delta \vdash \nabla\vartheta, {}^\pi\nabla\vartheta', l\vartheta \approx_\alpha {}^\pi l\vartheta' \implies \Delta, atms(R^{\text{N}})\#unkn(R\vartheta) \vdash l^{\text{N}}\theta\vartheta \approx_\alpha l^{\text{N}\pi}\theta\vartheta'$

From Lemma 4.7 ($\Rightarrow$), it follows that $\forall X \in unkn(l^{\text{N}}) : \Delta, atms(R^{\text{N}})\#unkn(R\vartheta) \vdash X\theta\vartheta \approx_\alpha X^\pi\theta\vartheta'$.

Since $unkn(r^{\text{N}}) \subseteq unkn(l^{\text{N}})$, Lemma 4.7 ($\Leftarrow$) can be used to demonstrate the equivalences

$$\Delta, atms(R^{\text{N}})\#unkn(R\vartheta) \vdash r^{\text{N}}\theta\vartheta \approx_\alpha r\vartheta, r^{\text{N}\pi}\theta\vartheta' \approx_\alpha {}^\pi r\vartheta', r^{\text{N}}\theta\vartheta \approx_\alpha r^{\text{N}\pi}\theta\vartheta'$$

and, finally, $\Delta, atms(R^{\text{N}})\#unkn(R\vartheta) \vdash r\vartheta \approx_\alpha {}^\pi r\vartheta'$ is obtained by transitivity. Notice that atoms in $atms(R^{\text{N}})$ do not appear in $r\vartheta, {}^\pi r\vartheta'$, so that the previous judgement can be strengthened taking only Δ as context. □

4.3 Better Criteria for Confluence of Closed Rewriting

In this section we study confluence of closed rewriting (Definition 26). Closed rewriting uses freshened versions of rules and nominal matching, instead of the computationally more expensive equivariant matching used in Definition 21. Closed rewriting is complete for equational reasoning if the axioms are closed [Fernández and Gabbay, 2010].

The following three lemmas state properties of closed rules and closed rewriting, and will be useful for Theorems 4.12 and 4.14. The first two state that if a rule has no free atoms then its freshness context can be extended to obtain a closed rule, and closed rewriting with either rule is equivalent. The third lemma states that a rule with free atoms generates an empty closed rewriting relation.

Lemma 4.9. *Let $R = \nabla \vdash l \to r$ be a rule such that every occurrence of an atom subterm a in l or r is under the scope of an abstraction of a (i.e., no atom occurs free as*

a subterm in R). Then there exists a minimal context $\Delta \subseteq atms(R)\#unkn(R)$ such that $\Delta, \nabla \vdash l \to r$ is closed.

Proof. By definition of closed term (see Definition 24), we must check:

1. Every occurrence of an atom subterm a is under an abstraction of a.

2. If $\pi \cdot X$ occurs under an abstraction of $\pi \cdot a$, then any occurrence of $\pi' \cdot X$ is in the scope of an abstraction of $\pi' \cdot a$ or $a\#X \in \nabla \cup \Delta$.

3. For any pair $\pi_1 \cdot X, \pi_2 \cdot X$ occurring in R and $a \in ds(\pi_1, \pi_2)$, if $\pi_1 \cdot a$ and $\pi_1 \cdot a$ are not abstracted over the respective occurrences of X, then $a\#X \in \nabla \cup \Delta$.

The first point holds by assumption. For the second and third points, if $a\#X \notin \nabla$ it is sufficient to include $a\#X$ in Δ. □

Lemma 4.10. *Suppose $R = \nabla \vdash l \to r$ and $R' = \Delta, \nabla \vdash l \to r$ are rules such that R has no free atom-subterms and $\Delta \subseteq atms(R)\#unkn(R)$ is the minimal set of freshness constraints that makes R' closed. Then, $\Gamma \vdash s \overset{R}{\to}_c t \;\Leftrightarrow\; \Gamma \vdash s \overset{R'}{\to}_c t$.*

Proof. The left-to-right direction. If $\Gamma \vdash s \overset{R}{\to}_c t$, then $\Gamma, atms(R^{\text{И}})\#unkn(\Gamma, s, t) \vdash s \overset{R^{\text{И}}}{\to} t$, i.e., there is θ such that

$$\Gamma, atms(R^{\text{И}})\#unkn(\Gamma, s, t) \vdash s|_p \approx_\alpha l^{\text{И}}\theta, t \approx_\alpha s[p \leftarrow r^{\text{И}}\theta], \nabla^{\text{И}}\theta.$$

Since $atms(R) = atms(R')$, it suffices to show that $\Gamma, atms(R^{\text{И}})\#unkn(\Gamma, s, t) \vdash \Delta^{\text{И}}\theta$ to obtain $\Gamma, atms(R^{\text{И}})\#unkn(\Gamma, s, t) \vdash s \overset{R'^{\text{И}}}{\to} t$ as required.

To prove $\Gamma, atms(R^{\text{И}})\#unkn(\Gamma, s, t) \vdash \Delta^{\text{И}}\theta$, observe that $a^{\text{И}}\#X^{\text{И}}$ is in $\Delta^{\text{И}}$ if $\pi_1^{\text{И}} \cdot X^{\text{И}}$ and $\pi_2^{\text{И}} \cdot X^{\text{И}}$ occur in $(l^{\text{И}}, r^{\text{И}})$ and at least one of the following holds:

- $\pi_1^{\text{И}} \cdot a^{\text{И}}$ is abstracted over $\pi_1^{\text{И}} \cdot X^{\text{И}}$ and $\pi_2^{\text{И}} \cdot a^{\text{И}}$ is not abstracted over $\pi_2^{\text{И}} \cdot X^{\text{И}}$. We know

$$\Gamma, atms(R^{\text{И}})\#unkn(\Gamma, s, t) \vdash \pi_2^{\text{И}} \cdot a^{\text{И}}\#(s|_p, t|_p),\ (s|_p, t|_p) \approx_\alpha (l^{\text{И}}\theta, r^{\text{И}}\theta).$$

  Then, since $\pi_2^{\text{И}} \cdot a^{\text{И}}$ is not abstracted over $\pi_2^{\text{И}} \cdot X^{\text{И}}$, the same freshness context allows us to derive $\pi_2^{\text{И}} \cdot a^{\text{И}}\#\pi_2^{\text{И}} \cdot X^{\text{И}}\theta$ and, consequently, $a^{\text{И}}\#X^{\text{И}}\theta$.

- $a^{\text{И}}$ is in $ds(\pi_1^{\text{И}}, \pi_2^{\text{И}})$ and neither $\pi_1^{\text{И}} \cdot a^{\text{И}}$ nor $\pi_2^{\text{И}} \cdot a^{\text{И}}$ are abstracted over the respective occurrences of $X^{\text{И}}$. The same argument is valid in this case.

The right-to-left direction. If $\Gamma \vdash s \overset{R'}{\to}_c t$, then $\Gamma, atms(R^{\text{И}})\#unkn(\Gamma, s, t) \vdash s \overset{R'^{\text{И}}}{\to} t$, i.e., there is θ such that

$$\Gamma, atms(R'^{\text{И}})\#unkn(\Gamma, s, t) \vdash s|_p \approx_\alpha l^{\text{И}}\theta, t \approx_\alpha s[p \leftarrow r^{\text{И}}\theta], \nabla^{\text{И}}\theta, \Delta^{\text{И}}\theta.$$

So $atms(R) = atms(R')$. It follows that $\Gamma, atms(R^{\text{и}})\#unkn(\Gamma, s, t) \vdash s \overset{R^{\text{и}}}{\to} t$. $\square$

Lemma 4.11. *Suppose $R = \nabla \vdash l \to r$ is a nominal rule and there exist Δ, s, t and a closed-rewriting step $\Delta \vdash s \overset{R}{\to}_c t$. Then every occurrence of an atom subterm a in l or r is under an abstraction of a (i.e., no atom occurs free as a subterm in R).*

Proof. By contradiction. Assume R has a free atom subterm a; without loss of generality, we assume $l|_q = a$ (if it occurs in r we reason in the same way). By definition of closed-rewriting, there exists $R^{\text{и}}$, a fresh variant of R, such that $\Delta, atms(R^{\text{и}})\#unkn(\Delta, s, t) \vdash s|_p \approx_\alpha l^{\text{и}}\theta, t \approx_\alpha s[p{\leftarrow}r^{\text{и}}\theta], \nabla^{\text{и}}\theta$. But $l^{\text{и}}|_q = a^{\text{и}}$ is free, and $a^{\text{и}}$ does not occur in s, contradicting $\Delta, atms(R^{\text{и}})\#unkn(\Delta, s, t) \vdash s|_p \approx_\alpha l^{\text{и}}\theta$. $\square$

The following definitions of *fresh overlap* and *fresh critical pair* will be used to derive sufficient conditions for confluence of closed rewriting.

Definition 33 (Fresh Overlaps and CPs) Let $R_i = \nabla_i \vdash l_i \to r_i$ $(i = 1, 2)$ be freshened versions of rewrite rules in Rw (R_1 and R_2 could be two freshened versions of the same rule), where $unkn(R_1) \cap unkn(R_2) = \varnothing$, as usual. If the nominal unification problem $\nabla_1 \cup \nabla_2 \cup \{l_2 \;{}_?{\approx}_?\; l_1|_p\}$ has a most general solution $\langle \Gamma, \theta \rangle$ for some position p, then we say that R_1 **fresh overlaps** with R_2, and we call the pair of terms-in-context $\Gamma \vdash (r_1\theta, l_1\theta[p{\leftarrow}r_2\theta])$ a **fresh critical pair**.

If p is a variable position, or if R_1 and R_2 are equal modulo renaming of variables and $p = \epsilon$, then we call the overlap and critical pair **trivial**.

If R_1 and R_2 are freshened versions of the same rule and $p = \epsilon$, then we call the overlap and critical pair **fresh root-permutative**.

A fresh overlap (resp. fresh critical pair) that is not trivial and not root-permutative is **proper**.

The fresh critical pair $\Gamma \vdash (r_1\theta, l_1\theta[p{\leftarrow}r_2\theta])$ is **joinable** if there is a term u such that $\Gamma \vdash_{\mathsf{R}} r_1\theta \to_c u$ and $\Gamma \vdash_{\mathsf{R}} (l_1\theta[p{\leftarrow}r_2\theta]) \to_c u$.

Definition 34 Call a rewrite theory $\mathsf{R} = (\Sigma, Rw)$ **fresh quasi-orthogonal** when all rules are left-linear and there are no proper fresh critical pairs.

Theorem 4.12 (Critical Pair Lemma for Closed Rewriting)**.**
Let $\mathsf{R} = (\Sigma, Rw)$ be a rewrite theory where every proper fresh critical pair is joinable. Then the closed rewriting relation generated by R is locally confluent.

Proof. Since rules with free atom-subterms do not generate closed rewriting steps (Lemma 4.11), without loss of generality we can assume that the rules in Rw do not have free atom-subterms. Consider $\mathsf{R}' = (\Sigma, Rw')$ the closed rewrite theory obtained by extending the freshness contexts of rules in Rw, as described in Lemma 4.9. Then, by Lemma 4.10, the closed rewriting relation generated by R is equivalent to the one generated by R'. Thus, joinability of proper fresh critical pairs in R implies joinability of proper fresh critical pairs in R' and it suffices to prove local confluence for the closed rewriting relation generated by R'. Also note that since all rules in Rw' are closed, they are uniform and α-stable (Theorem 4.8).

We consider the kinds of peaks that may arise:

- *If the rewrite steps defining the peak occur at disjoint positions* then the peak is trivially joinable by applying the same rules and substitutions.

- *If the peak is generated by an overlap at a variable position* then consider $R_1 = \nabla_1^{\text{И}} \vdash l_1^{\text{И}} \to r_1^{\text{И}}$ and $R_2 = \nabla_2^{\text{И}} \vdash l_2^{\text{И}} \to r_2^{\text{И}}$ freshened versions of two rules (see Figure 4.1, but here we do not need permuted versions for the rules are already freshened). Let Δ be the context used to rewrite $l_2^{\text{И}}\theta_2$ with R_1 and R_2. Without loss of generality, we assume $\Delta, atms(R_1)\#unkn(\Delta, s) \vdash \nabla_1^{\text{И}}\theta_1, s \approx_\alpha l_1^{\text{И}}\theta_1, t \approx_\alpha r_1^{\text{И}}\theta_1$ and s occurs inside $l_2^{\text{И}}\theta_2$ under an instance of an unknown $(\pi^{\text{И}} \cdot X^{\text{И}})\theta_2$. Then we can change the action of θ_2 over $X^{\text{И}}$, replacing s by t, such that $\nabla_1 \vdash t \approx_\alpha {}^{\pi_1}r_1\theta_1$, as it is done in the first-order case. Here we rely on the assumption of uniformity, which ensures that no free atoms are introduced by the rewrite step, therefore no freshness constraint will be violated when replacing s by t.

- *If there are freshened rules $R_1 = \nabla_1^{\text{И}} \vdash l_1^{\text{И}} \to r_1^{\text{И}}$ and $R_2 = \nabla_2^{\text{И}} \vdash l_2^{\text{И}} \to r_2^{\text{И}}$ and a term-in-context $\Delta \vdash s$, such that there is a rewrite step at position p_1 in s using R_1 and at position p_2 using R_2* then $\Delta, \Gamma_1 \vdash \nabla_1^{\text{И}}\theta, l_1^{\text{И}}\theta \approx_\alpha s|_{p_1}$ and $\Delta, \Gamma_2 \vdash \nabla_2^{\text{И}}\theta', l_2^{\text{И}}\theta' \approx_\alpha s|_{p_2}$. Without loss of generality we assume that $p_2 = p_1 q$. Since the sets of variables in the freshened rules are disjoint, without loss of generality we can assume $dom(\theta) \cap dom(\theta') = \emptyset$, and define the substitution $\vartheta = \theta \circ \theta'$ such that $dom(\vartheta) = dom(\theta) \cup dom(\theta')$. Then, $\Delta, \Gamma_1, \Gamma_2 \vdash \nabla_1^{\text{И}}\vartheta, \nabla_2^{\text{И}}\vartheta, l_1^{\text{И}}|_q\vartheta \approx_\alpha l_2^{\text{И}}\vartheta$. Therefore the unification problem $\nabla_1^{\text{И}}, \nabla_2^{\text{И}}, l_1^{\text{И}}|_q \;{}_?{\approx}_?\; l_2^{\text{И}}$ has a solution. Hence, by Definition 33, there is a fresh critical pair between R_1 and R_2. Observe that, if $q = \epsilon$ and R_1 is a permuted copy of R_2 (equal or not), then the terms of divergence t_1 and t_2 are α-equivalent by triviality or α-stability. If the critical pair is proper it is joinable by assumption. Therefore the peak is joinable since the rewriting relation is compatible with instantiation [Fernández and Gabbay, 2007, Theorem 49].

□

Since it is sufficient to consider just one freshened version of each rule when computing overlaps of closed rules, the number of fresh critical pairs for a rewrite theory with a finite number of rules is finite. Thus, Theorem 4.12 provides an effective criterion for local confluence, similar to the criterion for first-order systems.

We can deduce from Theorem 4.12 that the closed rewriting relation for the closed theory defining explicit substitution in Example 4.1 (i.e., all the rules except Beta) is locally confluent: every proper fresh critical pair is joinable. If we consider also the rule (Beta) then the system is not locally confluent. This does not contradict the previous theorem, because there is a proper fresh critical pair between (Beta) and (σ_{app}), obtained from $\varnothing \vdash ((\lambda[a]X)Y)[b \mapsto Z]$, which is not joinable:

$$\varnothing \vdash (((\lambda[a]X)[b \mapsto Z])(Y[b \mapsto Z]), (X[a \mapsto Y])[b \mapsto Z]).$$

Next we consider criteria for confluence based on (quasi-) orthogonality. The following lemma is used in the proof of confluence.

Lemma 4.13. *Let* $\mathsf{R} = (\Sigma, Rw)$ *be a closed rewrite theory.*
$\Delta \vdash_{\mathsf{R}} s \rightarrow_c t$ *if, and only if, there exist* $R_1, \ldots, R_n \in Rw$ *such that* $\Delta, atms(R_1^{\text{И}}, \ldots, R_n^{\text{И}}) \# unkn(\Delta, s) \vdash_{\mathsf{R}} s \rightarrow t$.

Proof. In both directions, the proof is by induction on the number of steps in $\Delta \vdash_{\mathsf{R}} s \rightarrow_c t$ and $\Delta, atms(R_1^{\text{И}}, \ldots, R_n^{\text{И}}) \# unkn(\Delta, s) \vdash_{\mathsf{R}} s \rightarrow t$, respectively. From left to right, the result follows by definition of closed rewriting. In the other direction, it is necessary to consider closedness of rules. Any version of $R \in Rw$ can be used in one step $\Delta, atms(R_1^{\text{И}}, \ldots, R_n^{\text{И}}) \# unkn(\Delta, s) \vdash s \xrightarrow{\mathsf{R}} v$. So, the version $R^{\text{И}}$ freshened with respect to $\Delta, atms(R_1^{\text{И}}, \ldots, R_n^{\text{И}})$ and all the terms in the rewrite sequence could be taken in this step. Weakening the freshness context, $\Delta, atms(R_1^{\text{И}}, \ldots, R_n^{\text{И}}, R^{\text{И}}) \# unkn(\Delta, s) \vdash s \xrightarrow{\mathsf{R}} v$ is obtained. Since the atoms of $R_1^{\text{И}}, \ldots, R_n^{\text{И}}$ do not occur in $\Delta, R^{\text{И}}$ and in the terms of the rewrite sequence, the freshness context can be strengthened into $\Delta, atms(R^{\text{И}}) \# unkn(\Delta, s) \vdash s \xrightarrow{\mathsf{R}} v$. Thus, $\Delta \vdash s \xrightarrow{\mathsf{R}}_c v$ is reached. □

Theorem 4.14. *If* R *is a fresh quasi-orthogonal rewrite theory, then the closed rewriting relation generated by* R *is confluent.*

Proof. As in the previous theorem, we prove confluence for the closed rewriting relation generated by $\mathsf{R}' = (\Sigma, Rw')$, where Rw' is obtained by extending the freshness contexts to close the rules of Rw which do not have free atom-subterms (see Lemmas 4.11, 4.9 and 4.10). Since all rules in Rw' are closed, they are also uniform and α-stable.

Now we can proceed in the usual way (see, e.g., [Baader and Nipkow, 1998, Rocha-Oliveira and Ayala-Rincón, 2012, Fernández and Gabbay, 2007]), by proving the

diamond property for a parallel closed-rewriting relation (simultaneous closed rewriting steps at disjoint positions). The proof proceeds by analysis of peaks: When overlaps occur under instances of variables, we use uniformity to ensure that when we change the substitution, the rewrite step is still possible. Joinability of root-permutative overlaps is a consequence of α-stability for the rules are closed.

Alternatively, we can prove confluence by reducing to a previous result for standard nominal rewriting, using the previous lemma: Consider a peak $\Delta \vdash_{\mathsf{R}'} s \to_c t$ and $\Delta \vdash_{\mathsf{R}'} s \to_c v$. By Lemma 4.13 ($\Rightarrow$), there exist $R_1, \ldots, R_n \in Rw'$ such that $\Delta, atms(R_1^{\text{и}}, \ldots, R_n^{\text{и}})\#unkn(\Delta, s) \vdash_{\mathsf{R}'} s \to t$ and $\Delta, atms(R_1^{\text{и}}, \ldots, R_n^{\text{и}})\#unkn(\Delta, s) \vdash_{\mathsf{R}'} s \to v$. Theorem 28 of [Suzuki et al., 2015] guarantees confluence with the context $\Delta, atms(R_1^{\text{и}}, \ldots, R_n^{\text{и}})\#unkn(\Delta, s)$, since for closed theories our notion of fresh-quasi-orthogonality coincides with the notion of orthogonality defined in [Suzuki et al., 2015] (in this case, it does not matter which permuted version is used to obtain a proper critical pair). Using Lemma 4.13($\Leftarrow$), we obtain confluence of $\Delta \vdash _ \stackrel{\mathsf{R}'}{\to}_c _$. □

Example 4.15. Consider a signature for first-order logic, with term-formers $\neg$, $\forall$ and $\exists$ of arity 1, and $\wedge, \vee$ of arity 2 (as usual we write them infix). The following closed rules can be used to simplify formulas:

$$\vdash \neg(X \wedge Y) \to \neg(X) \vee \neg(Y) \quad \text{and} \quad b\#X \vdash \neg(\forall[a]X) \to \exists[b]\neg((b\ a) \cdot X).$$

Why write $\exists[b]\neg((b\ a) \cdot X)$ on the right-hand side above, instead of the α-equivalent $\exists[a]\neg(X)$? We could: these are equivalent—in a nominal context. The version above directly translates the corresponding CRS rule (see [Domínguez and Fernández, 2014]) which, following Barendregt's convention, must use *different* names for bound variables in a rule. Theorem 4.14 tells us that the closed rewriting relation generated by the theory in Example 4.15 is confluent. This theory is closed, but forbidden by ASP restrictions because of the permutation $(b\ a)$ on the right-hand side.

The criteria for local confluence given in Theorem 4.12 and for confluence given in Theorem 4.14 for closed rewriting are easy to check using a nominal unification algorithm: just compute overlaps for the set of rules obtained by taking one freshened copy of each given rule. For comparison, the criteria given in [Fernández and Gabbay, 2007] and [Suzuki et al., 2015] require the computation of critical pairs for permutative variants of rules, which needs equivariant unification (exponential). Theorems 4.12 and 4.14 apply even if the rules are not closed, as long as we use closed rewriting. Consider the uniform rules $\vdash f(a) \to 0$ and $\vdash g(f(b)) \to 0$. These rules have no non-trivial fresh overlap, and closed rewriting is confluent, but the standard rewriting relation is not confluent, since

the term $g(f(a))$ rewrites to both $g(0)$ and 0. Using closed rewriting, the term $g(f(a))$ is a normal form.

Chapter 5

Nominal Essential Intersection Types

The Essential Intersection Type System in the context of the λ-calculus was explored in [van Bakel, 1995], stating the main advantages with respect to the BCD Intersection Type System. They include the fact that the set of typings assignable to a term is a proper subset with respect to the set in the BCD system and the direct relationship between terms and skeletons of type derivations. Beside that, the Essential System presents the closedness of typability under η-reduction, unlike the Strict Intersection Type System presented in [van Bakel, 1992].

The syntax for nominal types used here was inspired by the Essential Intersection Type System presented in [van Bakel and Fernández, 1997], where the system assigns types to first-order terms. There, only sorts of types, variables and arrow types are considered and, as in Definition 2, the arity of function symbols is fixed. Here, we extend it with abstraction types and allow type constructors to have arguments.

In this chapter, we present an Intersection Type System satisfying some basic properties; as expected, it was proved the equivariance of type derivations (renaming with permutations), invariance of typings for α-equivalent terms and subject reduction.

5.1 Types, ordering and operations

The next definition introduces the set of types considered in this chapter.

Definition 35 Let $\mathcal{C}, \mathcal{V}$ be a set of type constructors, each one with a fixed arity, and a countably infinite set of type variables, respectively. The set $\mathcal{T}_s$ is defined containing the **strict types**, which are generated by:

$$\tau ::= \varphi \mid \tau_1 \cap \cdots \cap \tau_k \to \tau \mid [\tau_1 \cap \cdots \cap \tau_k]\tau \mid \mathtt{C}(\tau_1, \ldots, \tau_n),$$

where $\varphi \in \mathcal{V}$, $\mathtt{C} \in \mathcal{C}$ and n is the arity of $\mathtt{C}$ and k might be possibly 0. Define $\mathcal{T}_S$ as the set of **intersection types**, which are built using types in $\mathcal{T}_s$ as

$$\sigma ::= \tau_1 \cap \cdots \cap \tau_k, \quad k \geq 0.$$

The symbol ω represents the intersection of zero strict types.

Notation. $\bigcap_{i=1}^{n} \tau_i \in \mathcal{T}_S$ whenever $\tau_i \in \mathcal{T}_s$ and, in particular, for $n = 1$, $\bigcap_{i=1}^{n} \tau_i \in \mathcal{T}_s$. The set $Vars(\sigma)$ contains $\varphi \in \mathcal{V}$ that occurs in σ.

In this chapter, we extend the signature of terms as presented in Definition 2. A functional symbol $\mathsf{f} \in \Sigma$ is accompanied by its arity $n \geq 0$ and a **type declaration** denoted by Σ_{f}, which is any type in $\mathcal{T}_S$.

Notice that, in this grammar of types, we add a structure for abstraction types with respect to the one in [van Bakel and Fernández, 1997]. In essence, these types behave as arrow types in the sense that contravariance in the "domain" is supposed when comparing arrow types as well as abstraction types (cf. Table 5.1) and intersections immediately in the right-hand side are prohibited for both.

Example 5.1. Let Π be a ternary symbol of a signature Σ and $[\mathtt{nat}]\varphi \to \mathtt{nat} \to \mathtt{nat} \to \varphi$ be the type declaration of Π. Consider $\prod_{i=n}^{k} X$ the syntactic sugar of $\prod([i]X, n, k)$. This type declaration is supposed to enforce the first argument of Π to be an abstraction type and the second and third ones to be naturals. The type φ must be the resulting type of a term that is a full instance of Π respecting the declaration.

Definition 36 A **type environment** Γ is a finite set of **type annotations** of the form $X : \sigma$ or $a : \sigma$, where each X/a appears only in one annotation. The type annotations of X and a in Γ are denoted by Γ_X and Γ_a, respectively.

The connective $\bowtie$ represents an **updating** in an environment Γ. In this way, if a and X are not annotated in Γ, then $\Gamma \bowtie a : \sigma$ and $\Gamma \bowtie X : \sigma$ denote $\Gamma \cup \{a : \sigma\}$ and $\Gamma \cup \{X : \sigma\}$, respectively. Otherwise, they respectively denote $\Gamma \setminus \Gamma_a \cup \{a : \sigma\}$ and $\Gamma \setminus \Gamma_X \cup \{X : \sigma\}$.

The meta-action of permutation (Definition 5) can be extended to environments: ${}^{\pi}\Gamma = \{\pi(a) : \Gamma_a \mid a \in \Gamma\} \cup \{X : \Gamma_X \mid X \in \Gamma\}$.

Pairs $\langle \Gamma, \sigma \rangle$ of an environment Γ and a type σ will be called *typings* with respect to a term, whose definition will be introduced formally in Definition 41. In Table 5.1, a partial order is defined for types, environments and this kind of pairs.

The lemma that follows is taken from [Kamareddine and Nour, 2007], where the difference between our types and theirs is the presence of abstraction types here. Since, the

Table 5.1: Relation $\leq$ between types

$(\leq_{\cap_\mathbf{E}})\ \tau_1 \cap \cdots \cap \tau_n \leq \tau_i\ ,\forall n \geq 1$	$(\leq_\mathbf{Trans})\ \dfrac{\sigma \leq \tau \qquad \tau \leq \rho}{\sigma \leq \rho}$	$(\leq_\mathbf{abs})\ \dfrac{\sigma \leq \gamma \qquad \rho \leq \tau}{[\gamma]\rho \leq [\sigma]\tau}$
$(\leq_{\cap_\mathbf{I}})\ \dfrac{\sigma \leq \tau_1 \qquad \ldots \qquad \sigma \leq \tau_n}{\sigma \leq \tau_1 \cap \cdots \cap \tau_n}$		$(\leq_\rightarrow)\ \dfrac{\sigma \leq \gamma \qquad \rho \leq \tau}{\gamma \rightarrow \rho \leq \sigma \rightarrow \tau}$
$(\leq_\mathbf{\Gamma})\ \dfrac{\forall y \in \Gamma : (y \in \Phi) \wedge (\Phi_y \leq \Gamma_y) \qquad y \in \mathbb{A} \cup \mathbb{V}}{\Phi \leq \Gamma}$		$(\leq_\mathbf{pair})\ \dfrac{\Gamma' \leq \Gamma \qquad \sigma \leq \sigma'}{\langle \Gamma, \sigma \rangle \leq \langle \Gamma', \sigma' \rangle}$

ordering $\leq$ does not mix abstraction and arrow types in rules of Table 5.1 and both are contravariant in the domain, the verification that the properties for "_ $\rightarrow$ _" hold for "[_]_" is straightforward. These properties will be used in the next section.

Lemma 5.2 ($\leq$-Inversion lemma [Kamareddine and Nour, 2007])**.**

1. *$\gamma \leq \sigma$ implies that $\gamma = \bigcap_{i=1}^{n} \tau_i$ and $\sigma = \bigcap_{j=1}^{m} \tau'_j$ and, for all $1 \leq j \leq m$, there exists $1 \leq i \leq n$ such that $\tau_i \leq \tau'_j$.*

2. *$[\gamma]\tau \leq \sigma$ implies that $\sigma = \bigcap_{i=1}^{n} [\gamma_i]\rho_i$, where $\gamma_i \leq \gamma$ and $\tau \leq \rho_i$ for all $1 \leq i \leq n$.*

Now, some operations on types are introduced, namely *lifting*, *substitution* and *expansion*. They are used in the next section similarly to the way they were originally presented in [van Bakel and Fernández, 1997], while designating types for functional terms. Beyond that, such operations are used to achieve typings of terms from a *principal* one. These notions will be presented later in this chapter.

Definition 37 An operation of **lifting** $L_{\langle\langle \Gamma,\sigma\rangle,\langle \Gamma',\sigma'\rangle\rangle} : \mathcal{T}_S \rightarrow \mathcal{T}_S$ is defined by a pair of pairs $\langle\langle \Gamma,\sigma\rangle,\langle \Gamma',\sigma'\rangle\rangle$ such that $\langle \Gamma,\sigma\rangle \leq \langle \Gamma',\sigma'\rangle$, following the rules:

- $L_{\langle\langle \Gamma,\sigma\rangle,\langle \Gamma',\sigma'\rangle\rangle}(\sigma) = \sigma'$,
- $L_{\langle\langle \Gamma,\sigma\rangle,\langle \Gamma',\sigma'\rangle\rangle}(\gamma) = \gamma$, if $\sigma \neq \gamma$,
- $L_{\langle\langle \Gamma,\sigma\rangle,\langle \Gamma',\sigma'\rangle\rangle}(\Phi) = \Phi \bowtie \{y : \gamma' \in \Gamma' \mid y : \gamma \in \Gamma \cap \Phi$ or $y \notin \Gamma \cup \Phi\}$, where $y \in \mathbb{A} \cup \mathbb{V}$,
- $L_{\langle\langle \Gamma,\sigma\rangle,\langle \Gamma',\sigma'\rangle\rangle}(\langle \Phi, \gamma\rangle) = \langle (L_{\langle\langle \Gamma,\sigma\rangle,\langle \Gamma',\sigma'\rangle\rangle}(\Phi)), (L_{\langle\langle \Gamma,\sigma\rangle,\langle \Gamma',\sigma'\rangle\rangle}(\gamma))\rangle$.

Observe that, if $\Phi = \Gamma$, then $L_{\langle\langle \Gamma,\sigma\rangle,\langle \Gamma',\sigma'\rangle\rangle}(\Gamma) = \Gamma'$.

Example 5.3. Consider $L = L_{\langle\langle\{X:\varphi_1\cap\varphi_2\},\varphi_1\cap\varphi_2\rangle,\langle\{X:\varphi_1\cap\varphi_2\cap\varphi_3\},\varphi_2\rangle\rangle}$. So, one can obtain a lifted pair as follows on the pair below:

$$L(\langle\{X:\varphi_1\cap\varphi_2\},\varphi_1\cap\varphi_2\rangle) = \langle\{X:\varphi_1\cap\varphi_2\cap\varphi_3\},\varphi_2\rangle$$

However, if the lifting is applied to some different environment, then the result may vary:

$$L(\{X:\varphi_1\cap\varphi_2, a:\rho\}) = \{X:\varphi_1\cap\varphi_2\cap\varphi_3, a:\rho\},$$

$$L(\varnothing) = \{X:\varphi_1\cap\varphi_2\cap\varphi_3\},$$

$$L(\{X:\varphi_1\cap\varphi_5\}) = \{X:\varphi_1\cap\varphi_5\}.$$

In any case, we have an environment smaller than the original with respect to $\leq$.

Definition 38 A **substitution** $(\varphi \mapsto \alpha) : \mathcal{T}_S \to \mathcal{T}_S$, where φ is a type variable and $\alpha \in \mathcal{T}_s \cup \{\omega\}$, is defined as:

1. $(\varphi \mapsto \alpha)(\varphi) = \alpha$,
2. $(\varphi \mapsto \alpha)(\varphi') = \varphi'$, if $\varphi \neq \varphi'$,
3. $(\varphi \mapsto \alpha)(\mathtt{C}(\tau_1,\dots,\tau_n)) = \omega$, if $(\varphi \mapsto \alpha)(\tau_i) = \omega$, for some $i = 1,\dots,n$,
4. $(\varphi \mapsto \alpha)(\mathtt{C}(\tau_1,\dots,\tau_n)) = \mathtt{C}((\varphi \mapsto \alpha)(\tau_1),\dots,(\varphi \mapsto \alpha)(\tau_n))$, if $(\varphi \mapsto \alpha)(\tau_i) \neq \omega$, for all $i = 1,\dots,n$,
5. $(\varphi \mapsto \alpha)(\sigma \to \tau) = \omega$, if $(\varphi \mapsto \alpha)(\tau) = \omega$,
6. $(\varphi \mapsto \alpha)(\sigma \to \tau) = (\varphi \mapsto \alpha)(\sigma) \to (\varphi \mapsto \alpha)(\tau)$, if $(\varphi \mapsto \alpha)(\tau) \neq \omega$,
7. $(\varphi \mapsto \alpha)([\sigma]\tau) = \omega$, if $(\varphi \mapsto \alpha)(\tau) = \omega$,
8. $(\varphi \mapsto \alpha)([\sigma]\tau) = [(\varphi \mapsto \alpha)(\sigma)](\varphi \mapsto \alpha)(\tau)$, if $(\varphi \mapsto \alpha)(\tau) \neq \omega$,
9. $(\varphi \mapsto \alpha)(\tau_1 \cap \cdots \cap \tau_n) = (\varphi \mapsto \alpha)(\rho_1) \cap \cdots \cap (\varphi \mapsto \alpha)(\rho_m)$, where $\{\rho_1,\dots,\rho_m\} = \{\tau_i \in \{\tau_1,\dots,\tau_n\} \mid (\varphi \mapsto \alpha)(\tau_i) \neq \omega\}$,
10. $(\varphi \mapsto \alpha)(\Gamma) = \{y : (\varphi \mapsto \alpha)(\gamma) \mid y:\gamma \in \Gamma\}$,
11. $(\varphi \mapsto \alpha)(\langle\Gamma,\gamma\rangle) = \langle(\varphi \mapsto \alpha)(\Gamma), (\varphi \mapsto \alpha)(\gamma)\rangle$.

Example 5.4. Take a substitution $S = (\varphi_1 \mapsto \texttt{nat})$. So, we can instantiate the following pair as follows:

$$S(\langle\{X : [\varphi_1]\varphi_2\}, \varphi_1\rangle) = \langle\{X : [\texttt{nat}]\varphi_2\}, \texttt{nat}\rangle.$$

For $S' = (\varphi_2 \mapsto \omega) \circ S$, we must be careful:

$$S'(\varphi_1 \cap \varphi_2) = \texttt{nat},$$

$$S'(\varphi_1 \to \varphi_2) = \omega.$$

The next lemma asserts the compatibility of substitutions with the relation $\leq$.

Lemma 5.5 (Compatibility of substitutions with $\leq$)**.** *Let S be a type substitution.*

1. *$\sigma \leq \gamma$ implies $S(\sigma) \leq S(\gamma)$.*
2. *$\Phi \leq \Gamma$ implies $S(\Phi) \leq S(\Gamma)$.*

Proof. 1. The proof is by induction on the derivation of $\sigma \leq \gamma$. The interesting cases are when the rules $(\leq_{\mathbf{abs}})$ or $(\leq_{\to})$ are the last used ones. Since those cases are very similar, only the case for $(\leq_{\to})$ will be presented here.

We have $\sigma = \sigma' \to \rho$ and $\gamma = \gamma' \to \tau$. As premises of the rule, it holds that $\gamma' \leq \sigma'$ and $\rho \leq \tau$. By IH, $S(\gamma') \leq S(\sigma')$ and $S(\rho) \leq S(\tau)$. If $S(\rho) = \omega$, then $S(\tau) = \omega$ by Lemma 5.2 item (1). In this case, $S(\sigma' \to \rho) = \omega \leq \omega = S(\gamma' \to \tau)$. Otherwise, $S(\sigma' \to \rho) = S(\sigma') \to S(\rho) \leq S(\gamma') \to S(\tau) = S(\gamma' \to \tau)$ using rule $(\leq_{\to})$.

2. It follows by item 1 and by definition of rule $(\leq_{\mathbf{\Gamma}})$.

□

The next two definitions were based on [van Bakel, 2011], but here we also deal with the abstraction types and the types with constructors.

Definition 39 The **last type variable set** of a type τ in $\mathcal{T}_s$ is denoted by $LV(\tau)$ and defined by:

1. $LV(\varphi)$ is $\{\varphi\}$,
2. $LV(\texttt{C}(\tau_1, \ldots, \tau_n))$ is $\bigcup_{i=1}^{n} LV(\tau_i)$,
3. $LV(\sigma \to \tau)$ and $LV([\sigma]\tau)$ is $LV(\tau)$.

The notion of last type variable set is used to define the operation of type expansion in the next definition. Notice that, in the case of arrow and abstraction types, we focus on

the type of the right-hand side, since they require attention in the sense that intersections cannot be introduced there directly.

Definition 40 For all $\psi \in \mathcal{T}_s$, $k \geq 2$, environment Γ and type σ, the quadruple $\langle \psi, k, \Gamma, \sigma \rangle$ determines an **expansion** $Exp_{\langle \psi,k,\Gamma,\sigma \rangle} : \mathcal{T}_S \to \mathcal{T}_S$, which proceeds as follows:

1. The set of type variables $\mathcal{V}_\psi(\Gamma, \sigma)$ affected by $Exp_{\langle \psi,k,\Gamma,\sigma \rangle}$ is built by:
   a) If φ occurs in ψ, then $\varphi \in \mathcal{V}_\psi(\Gamma, \sigma)$,
   b) If $\tau \in \mathcal{T}_s$ is a subtype of Γ or σ and the intersection between $LV(\tau)$ and $\mathcal{V}_\psi(\Gamma, \sigma)$ is not empty, then all other type variables of τ are in $\mathcal{V}_\psi(\Gamma, \sigma)$.
2. Suppose $\mathcal{V}_\psi(\Gamma, \sigma) = \{\varphi_1, \ldots, \varphi_m\}$. Take $m \times k$ different type variables $\varphi_1^1, \ldots, \varphi_1^k, \ldots, \varphi_m^1, \ldots, \varphi_m^k$, disjoint of the variables in Γ and σ. For each $i = 1, \ldots, k$, consider a substitution S_i such that $S_i(\varphi_j) = \varphi_j^i$, for all $j = 1, \ldots, m$.
3. $Exp_{\langle \psi,k,\Gamma,\sigma \rangle}(\gamma)$ is computed traversing γ top-down and replacing every subtype τ of γ by $S_1(\tau) \cap \cdots \cap S_k(\tau)$ whenever $LV(\tau)$ intersects $\mathcal{V}_\psi(\Gamma, \sigma)$, i.e., for $Ex = Exp_{\langle \psi,k,\Gamma,\sigma \rangle}$,

$$
\begin{array}{lll}
Ex(\tau_1 \cap \cdots \cap \tau_n) & = & \bigcap_{i=1}^n Ex(\tau_i) \\
Ex(\tau) & = & \bigcap_{i=1}^k (S_i(\tau)), \quad \text{if } LV(\tau) \text{ intersects } \mathcal{V}_\psi(\Gamma, \sigma) \\
Ex(\gamma' \to \rho) & = & Ex(\gamma') \to Ex(\rho), \quad \text{if } LV(\rho) \text{ does not intersect } \mathcal{V}_\psi(\Gamma, \sigma) \\
Ex([\gamma']\rho) & = & [Ex(\gamma')]Ex(\rho), \quad \text{if } LV(\rho) \text{ does not intersect } \mathcal{V}_\psi(\Gamma, \sigma) \\
Ex(\mathtt{C}(\tau_1, \ldots, \tau_n)) & = & \mathtt{C}(Ex(\tau_1), \ldots, Ex(\tau_n)), \quad \text{if } LV(\mathtt{C}(\tau_1, \ldots, \tau_n)) \text{ does not intersect } \mathcal{V}_\psi(\Gamma, \sigma) \\
Ex(\varphi) & = & \varphi, \quad \text{if } \varphi \text{ is not in } \mathcal{V}_\psi(\Gamma, \sigma).
\end{array}
$$

Example 5.6. When it is desirable to add polymorphism in the domain of an arrow or abstraction type, the intersection can be used without problem because our grammar allows intersection on the left-hand side of such types. However, if the polymorphism is expected in the right-hand side, some special treatment is necessary. For example, to expand φ_1 in $[\varphi_1]\varphi_2$ one could obtain $[\varphi_3 \cap \varphi_4]\varphi_2$. On the other hand, to expand φ_2 in the same type, it is necessary to expand the entire type in the sense that we cannot add

an intersection in the right-hand side. So the result would be $[\varphi_3]\varphi_5 \cap [\varphi_4]\varphi_6$.

The next lemma is a technical result to prove the compatibility of type expansion with the ordering $\leq$.

Lemma 5.7. *For $\tau, \rho \in \mathcal{T}_s$, $\tau \leq \rho$ implies $LV(\tau) = LV(\rho)$.*

Proof. By induction on the derivation of $\tau \leq \rho$. Lets analyse the cases of transitivity and abstraction.

($\leq_{\mathbf{Trans}}$) In this case, there exists σ such that $\tau \leq \sigma \leq \rho$. By Lemma 5.2(1), $\sigma = \bigcap_{i=1}^{k} \sigma_i$ such that $\tau \leq \sigma_i$, for all $i = 1, \ldots, k$, and $\sigma_i \leq \rho$, for some i. Thus, by IH, it follows that $LV(\tau) = LV(\sigma_i)$, for all i. In this way, $LV(\sigma_i) = LV(\rho)$ for all i, by IH and because all $LV(\sigma_i)$ are equal. The result follows by transitivity.

($\leq_{\mathbf{abs}}$) $\tau = [\sigma]\tau'$, $\rho = [\gamma]\rho'$ and the following assertions are valid: $\gamma \leq \sigma$ and $\tau' \leq \rho'$. By IH, $LV(\tau') = LV(\rho')$. So, by definition of last type variable set, $LV(\tau) = LV(\rho)$.

□

The Lemma 5.8 and its proof is also presented in [van Bakel and Fernández, 1997] and it states the compatibility of expansion with the ordering $\leq$.

Lemma 5.8 (Compatibility of Expansion with $\leq$)**.** *Let $E = Exp_{\langle\psi,k,\Gamma,\sigma\rangle}$ be a type expansion.*

1. *$\sigma \leq \gamma$ implies $E(\sigma) \leq E(\gamma)$.*
2. *$\Phi \leq \Gamma$ implies $E(\Phi) \leq E(\Gamma)$.*

Proof. The proof is by induction on the derivation of $\sigma \leq \gamma$. Only the case ($\leq_{\mathbf{abs}}$) will be presented here.

We have $\sigma = [\sigma']\rho$ and $\gamma = [\gamma']\tau$. As premises of the rule, it holds that $\gamma' \leq \sigma'$ and $\rho \leq \tau$. By Lemma 5.7, we have only two cases:

- If $LV([\sigma']\rho)$ and $LV([\gamma']\tau)$ intersect $\mathcal{V}_\psi(\Gamma, \sigma)$, then $E([\sigma']\rho) = \bigcap_{i=1}^{k} S_i([\sigma']\rho)$ and $E([\gamma']\tau) = \bigcap_{i=1}^{k} S_i([\gamma']\tau)$. By Lemma 5.5, $S_i([\sigma']\rho) \leq S_i([\gamma']\tau)$ for all $i = 1, \ldots, k$, and the result follows.
- If $LV([\sigma']\rho)$ does not intersect $\mathcal{V}_\psi(\Gamma, \sigma)$ so neither does $LV([\gamma']\tau)$ and vice versa. By IH, $E(\gamma') \leq E(\sigma')$ and $E(\rho) \leq E(\tau)$, what implies that $E([\sigma']\rho) = [E(\sigma')]E(\rho) \leq [E(\gamma')]E(\tau) = E([\gamma']\tau)$, as it was supposed to be proven.

□

5.2 Type Inference System and Basic Properties

This section presents the system of type derivation rules that is explored in this thesis and the fundamental properties about it, which include the preservation of types for α-equivalent terms.

Definition 41 A **type judgement** is a tuple of a type environment, a term and a type, denoted by $\Gamma \vdash t : \gamma$. A type judgement $\Gamma \vdash t : \gamma$ is **derivable** if it can be deduced following the rules in Table 5.2; if so, then $\langle \Gamma, \gamma \rangle$ is called a **typing** of t. In Table 5.2, all types are strict except for the σ's that can be intersection types while C ranges over **chains of type operations**, that is a sequence $\langle O_1, \ldots, O_k \rangle$ of type operations of lifting, substitution and expansion that apply to types as follows:

$$C(\sigma) = \langle O_1, \ldots, O_k \rangle(\sigma) = O_1(\ldots(O_k(\sigma))\ldots).$$

Here liftings, substitutions and expansions do not have a determined order in chains. The empty chain is denoted by Id.

Table 5.2: Type System

$$(\mathcal{T}_{\mathsf{a}})\ \frac{\sigma \leq \tau}{\Gamma \bowtie a : \sigma \vdash a : \tau} \qquad (\mathcal{T}_{\mathbf{X}})\ \frac{\sigma \leq \tau}{\Gamma \bowtie X : \sigma \vdash \pi \cdot X : \tau}$$

$$(\mathcal{T}_{[\mathsf{a}]})\ \frac{\Gamma \bowtie a : \sigma \vdash t : \tau}{\Gamma \vdash [a]t : [\sigma]\tau} \qquad (\mathcal{T}_{\cap})\ \frac{\Gamma \vdash t : \tau_1 \quad \ldots \quad \Gamma \vdash t : \tau_k}{\Gamma \vdash t : \tau_1 \cap \cdots \cap \tau_k}, k \neq 1$$

$$(\mathcal{T}_{\mathsf{f}})\ \frac{\sigma_1 \to \cdots \to \sigma_n \to \tau = C(\Sigma_{\mathsf{f}}) \quad \Gamma \vdash t_1 : \sigma_1 \quad \ldots \quad \Gamma \vdash t_n : \sigma_n}{\Gamma \vdash \mathsf{f}(t_1, \ldots, t_n) : \tau}$$

The rule $(\mathcal{T}_{\cap})$ can be an axiom in the case that $k = 0$. This makes possible that any term is typable with type ω.

The next lemma is an inversion lemma on typing $\langle \Gamma, \sigma \rangle$ with respect to the structure of the corresponding term t.

Lemma 5.9 (Generation Lemma). *1. If $\Gamma \vdash a : \sigma$, then $\sigma = \omega$ or there exists $a : \gamma \in \Gamma$ such that $\gamma \leq \sigma$.*

2. If $\Gamma \vdash \pi \cdot X : \sigma$, then $\sigma = \omega$ or there exists $X : \gamma \in \Gamma$ such that $\gamma \leq \sigma$.

3. *If* $\Gamma \vdash [a]t : \sigma$*, then* $\sigma = \bigcap_{i=1}^{n}[\rho_i]\tau_i$ *and* $\Gamma \bowtie a : \rho_i \vdash t : \tau_i$*.*

4. *If* $\Gamma \vdash \mathsf{f}(t_1, \ldots, t_n) : \sigma$*, then* $\sigma = \bigcap_{j=1}^{k} \rho_j$ *and there are chains of operations* C_j*,* $1 \leq j \leq k$*, such that* $C_j(\Sigma_{\mathsf{f}}) = \gamma_{j_1} \to \cdots \to \gamma_{j_n} \to \rho_j$*,*

$$\Gamma \vdash t_1 : \gamma_{j_1} \ , \ \ldots \ , \ \Gamma \vdash t_n : \gamma_{j_n}, \ \ \forall j = 1, \ldots, k.$$

Proof. 1. The proof is by induction on the derivation of types. The only rules that can be the last ones to be used are $(\mathcal{T}_{\mathbf{a}})$ and $(\mathcal{T}_{\cap})$. If it is $(\mathcal{T}_{\mathbf{a}})$ so the statement is obtained because there is $a : \gamma$ annotated in Γ with $\gamma \leq \sigma$. If the rule is $(\mathcal{T}_{\cap})$, then $\sigma = \sigma_1 \cap \cdots \cap \sigma_n$ and the deductions $\Gamma \vdash a : \sigma_i$ are valid for all $1 \leq i \leq n$. If $n = 0$ then $\sigma = \omega$. If $n \neq 0$, the IH can be used and there is a type annotation $a : \gamma \in \Gamma$ such that $\gamma \leq \sigma_i$, what implies that $\gamma \leq \sigma$ by rule $(\leq_{\cap\mathbf{I}})$. Notice that the same Γ is used in all the n subderivations.

2. The proof is similar to the previous item, but the rules that can be for last ones to be used are $(\mathcal{T}_{\mathbf{X}})$ or $(\mathcal{T}_{\cap})$.

3. The rules that can be used for last are $(\mathcal{T}_{[\mathbf{a}]})$ or $(\mathcal{T}_{\cap})$. If the rule is $(\mathcal{T}_{[\mathbf{a}]})$, then $\sigma = [\rho]\tau$. If the rule is $(\mathcal{T}_{\cap})$, then $\sigma = \delta_1 \cap \cdots \cap \delta_n$ and the deductions $\Gamma \vdash [a]t : \delta_i$ hold for every $1 \leq i \leq n$. Since each δ_i is a strict type, the rule $(\mathcal{T}_{\cap})$ cannot be applied again. So, each $\delta_i = [\rho_i]\tau_i$, as analysed first.

4. The only possible rules to apply are $(\mathcal{T}_{\mathsf{f}})$ and $(\mathcal{T}_{\cap})$. If $(\mathcal{T}_{\cap})$ is applied, then $\sigma = \bigcap_{j=1}^{k} \rho_j$, with k derivations. Since each ρ_j is a strict type, only the $(\mathcal{T}_{\mathsf{f}})$ rule could be applied before. So, the conditions for each ρ_j are supplied, as in the statement of the lemma.

□

Following, the operations of lifting, substitution and expansion are proved to be sound in the type system.

Lemma 5.10 (Soundness of Lifting)**.** *If* $\Gamma \vdash t : \tau$ *and* $\langle \Gamma, \tau \rangle \leq \langle \Gamma', \tau' \rangle$*, then* $\Gamma' \vdash t : \tau'$*.*

Proof. By induction on the derivation of $\Gamma \vdash t : \tau$.

- Rule $(\mathcal{T}_{\mathbf{a}})$: $\Gamma'_a \leq \Gamma_a \leq \tau \leq \tau'$. So, $\Gamma' \vdash a : \tau'$.
- Rule $(\mathcal{T}_{\mathbf{X}})$: $\Gamma'_X \leq \Gamma_X \leq \tau \leq \tau'$. Thus, $\Gamma' \vdash \pi \cdot X : \tau'$.
- Rule $(\mathcal{T}_{[\mathbf{a}]})$: We have $\Gamma \bowtie a : \sigma \vdash t' : \rho$. If $[\sigma]\rho \leq \tau'$, then $\tau' = \bigcap_{i=1}^{m}[\sigma'_i]\rho'_i$ by Lemma 5.2(2) and, for each i, $\sigma'_i \leq \sigma$ and $\rho \leq \rho'_i$. So, by IH, $\Gamma' \bowtie a : \sigma'_i \vdash t' : \rho'_i$. Thus, using rules $(\mathcal{T}_{[\mathbf{a}]})$ and $(\mathcal{T}_{\cap})$, we obtain $\Gamma' \vdash [a]t' : \tau'$.

- Rule $(\mathcal{T}_\cap)$: $\tau = \tau_1 \cap \cdots \cap \tau_n \leq \tau'$, $\Gamma \vdash t : \tau_j$ for each $j = 1, \ldots, n$ and, by Lemma 5.2(1), $\tau' = \bigcap_{i=1}^{m} \tau'_i$ such that, for all $i = 1, \ldots, m$, there exists $j = 1, \ldots, n$ satisfying $\tau_j \leq \tau'_i$. By IH, $\Gamma' \vdash t : \tau'_i$ and, applying $(\mathcal{T}_\cap)$, one obtains $\Gamma' \vdash t : \tau'$.

- Rule $(\mathcal{T}_\mathsf{f})$: There exists a chain C of operations such that $C(\Sigma_\mathsf{f}) = \sigma_1 \to \cdots \to \sigma_n \to \tau$ and $\Gamma \vdash t_i : \sigma_i$, for all $i = 1, \ldots, n$. This implies that $\Gamma' \vdash t_i : \sigma_i$ by IH. Since $\sigma_1 \to \cdots \to \sigma_n \to \tau \leq \sigma_1 \to \cdots \to \sigma_n \to \tau'$ (because $\tau \leq \tau'$), the lifting $L = L_{\langle\langle \varnothing, \sigma_1 \to \cdots \to \sigma_n \to \tau \rangle, \langle \varnothing, \sigma_1 \to \cdots \to \sigma_n \to \tau' \rangle\rangle}$ is well defined and $L \circ C(\Sigma_\mathsf{f}) = \sigma_1 \to \cdots \to \sigma_n \to \tau'$, what gives us $\Gamma' \vdash \mathsf{f}(t_1, \ldots, t_n) : \tau'$.

□

Lemma 5.11 (Soundness of type substitution). *Let S be a type substitution and $\Gamma \vdash t : \sigma$ a derivable judgement. So, $S(\Gamma) \vdash t : S(\sigma)$ has a type derivation that is exactly as the original with the substitution S applied in each node.*

Proof. Since the case when $S(\sigma) = \omega$ is trivial, we consider $S(\sigma) \neq \omega$. The proof proceeds by induction on the derivation of $\Gamma \vdash t : \sigma$.

- Rule $(\mathcal{T}_\mathbf{a})$: $t = a$ and $\Gamma_a \leq \sigma$. Since $S(\Gamma_a) \leq S(\sigma)$ by Lemma 5.5, we obtain $S(\Gamma) \vdash a : S(\sigma)$.

- Rule $(\mathcal{T}_\mathbf{X})$: Similar to the previous item.

- Rule$(\mathcal{T}_{[\mathbf{a}]})$: $t = [a]t'$, $\sigma = [\gamma]\tau$ and $\Gamma \bowtie a : \gamma \vdash t' : \tau$. By IH, $S(\Gamma) \bowtie a : S(\gamma) \vdash t' : S(\tau)$. Since $S([\gamma]\tau) \neq \omega$, one has $S[\tau] \neq \omega$ and $S([\gamma]\tau) = [S(\gamma)]S(\tau)$. In this way, $S(\Gamma) \vdash [a]t' : S([\gamma]\tau)$.

- Rules $(\mathcal{T}_\cap)$ and $(\mathcal{T}_\mathsf{f})$: these cases are developed in [van Bakel and Fernández, 1997].

□

Lemma 5.12 (Soundness of type expansion). *Let E be a type expansion and $\Gamma \vdash t : \sigma$ a derivable judgement. So, $E(\Gamma) \vdash t : E(\sigma)$ is also derivable.*

Proof. Suppose w.l.o.g. that $\sigma \in \mathcal{T}_s$. In the proof of Theorem 4.4.3 of [van Bakel and Fernández, 1997], most of the proof is done, including the case when $E(\sigma) \notin \mathcal{T}_s$. We only need to add the case of rule $(\mathcal{T}_{[\mathbf{a}]})$ when $E(\sigma) \in \mathcal{T}_s$.

In such case, one has $\sigma = [\sigma']\tau$, $t = [a]t'$ and $\Gamma \bowtie a : \sigma' \vdash t' : \tau$. By IH, we obtain $E(\Gamma) \bowtie a : E(\sigma') \vdash t' : E(\tau)$. Applying rule $(\mathtt{abs})$, it holds that $E(\Gamma) \vdash [a]t' : [E(\sigma')]E(\tau)$. If the last type variable set of τ intersected $\mathcal{V}_\psi(\Gamma, \sigma)$, $E([\sigma']\tau)$ would be an intersection, what contradicts $E(\sigma) \in \mathcal{T}_s$. So, $E([\sigma']\tau) = [E(\sigma')]E(\tau)$, what concludes the proof. □

The proposition that follows states that atoms which are fresh in a term are not important in a typing (they can be removed or updated in the environment) and atoms and unkowns that do not occur in the environment are fresh (names) or do not occur in the term.

Proposition 5.13. *1. If $\Gamma \vdash t : \sigma$ and there exists some Δ such that $\Delta \vdash a\#t$, then $\Gamma \bowtie a : \tau \vdash t : \sigma$. (Type weakening).*

2. If $\Gamma \bowtie a : \tau \vdash t : \sigma$ and there exists some Δ such that $\Delta \vdash a\#t$, then $\Gamma \vdash t : \sigma$. (Type strengthening)

Proof. By induction on deductions.

1. - Rule $(\mathcal{T}_{\mathbf{a}})$: $\Gamma \bowtie b : \gamma \vdash b : \sigma$ with $\gamma \leq \sigma$, $a \notin atms(b)$ and $\Delta \vdash a\#b$. So, $\Gamma \bowtie b : \gamma \bowtie a : \tau \vdash b : \sigma$ using rule $(\mathcal{T}_{\mathbf{a}})$.
   - Rule $(\mathcal{T}_{\mathbf{X}})$: similarly to the previous case, $\Gamma \bowtie X : \gamma \bowtie a : \tau \vdash \pi \cdot X : \sigma$ with $\gamma \leq \sigma$.
   - Rule $(\mathcal{T}_{[\mathbf{a}]})$: $t = [b]t'$, $\sigma = [\sigma']\sigma''$ and $\Gamma \bowtie b : \sigma' \vdash t' : \sigma''$. One has to consider 2 cases. If $a = b$, then $\Gamma \bowtie a : \tau \bowtie a : \sigma' = \Gamma \bowtie b : \sigma'$ and we have the same valid judgement. If $a \neq b$, then $\Delta \vdash a\#t'$ (or $a \notin atms(t')$) and, by IH, $\Gamma \bowtie b : \sigma' \bowtie a : \tau \vdash t' : \sigma''$.
   - Rule $(\mathcal{T}_{\cap})$: it can be obtained applying IH in the cases that $\sigma \neq \omega$. If $\sigma = \omega$, the derivation is trivial.
   - Rule $(\mathcal{T}_{\mathsf{f}})$: it holds directly applying the induction hypothesis.
2. Similar to the weakening.

□

The following lemma will be specially used in the context of "typed rewrite steps" in Section 5.3, where renamed versions of rules are allowed.

Lemma 5.14 (Meta Level Equivariance of Type Derivations)**.** *$\Gamma \vdash t : \tau$ is derivable if and only if so is ${}^{\pi}\Gamma \vdash {}^{\pi}t : \tau$, where the permutation acts the same way throughout the derivation, applied to each node.*

Proof. By induction on type deductions.

- Rule $(\mathcal{T}_{\mathbf{a}})$: If $\sigma \leq \tau$, then $\Gamma \bowtie a : \sigma \vdash a : \tau$ if and only if ${}^{\pi}\Gamma \bowtie \pi(a) : \sigma \vdash {}^{\pi}a : \tau$.
- Rule $(\mathcal{T}_{\mathbf{X}})$: the annotation of variables is not changed by permutations.

- Rule $(\mathcal{T}_{[\mathbf{a}]})$: By IH, $\Gamma \bowtie a : \sigma \vdash t' : \rho$ if and only if ${}^{\pi}\Gamma \bowtie \pi(a) : \sigma \vdash {}^{\pi}t' : \rho$ and the statement is completed by applying the rule $(\mathcal{T}_{[\mathbf{a}]})$.
- Rule$(\mathcal{T}_{\cap})$: By IH, $\Gamma \vdash t : \tau_i$ for all $i = 1, \ldots, m$ if and only if ${}^{\pi}\Gamma \vdash {}^{\pi}t : \tau_i$ and this case is finished by applying the rule $(\mathcal{T}_{\cap})$.
- Rule $(\mathcal{T}_{\mathsf{f}})$: There is a chain C such that $C(\Sigma_{\mathsf{f}}) = \sigma_1 \to \cdots \to \sigma_n \to \tau$ and, for all $i = 1, \ldots, n$, $\Gamma \vdash t_i : \sigma_i$. By IH, ${}^{\pi}\Gamma \vdash {}^{\pi}t_i : \sigma_i$. So, ${}^{\pi}\Gamma \vdash {}^{\pi}\mathsf{f}(t_1, \ldots, t_n) : \tau$. The reverse is analogous.

□

The next lemma explains why we do not mind about the types that are annotated for the atoms which occur in permutations, since permutations do not change types, whenever the same renaming is used in the type environment.

Lemma 5.15 (Object Level Equivariance of Typing Derivations)**.** $\Gamma \vdash t : \tau$ *is derivable if and only if so is* ${}^{\pi}\Gamma \vdash \pi \bullet t : \tau$*, where the permutation acts the same way throughout the derivation, applied to each node.*

Proof. By induction on type deductions.

- Rule $(\mathcal{T}_{\mathbf{a}})$: If $\sigma \leq \tau$, then $\Gamma \bowtie a : \sigma \vdash a : \tau$ if and only if ${}^{\pi}\Gamma \bowtie \pi(a) : \sigma \vdash \pi(a) : \tau$.
- Rule $(\mathcal{T}_{\mathbf{X}})$: the annotation of variables is not changed by permutations.
- Rule $(\mathcal{T}_{[\mathbf{a}]})$: By IH, $\Gamma \bowtie a : \sigma \vdash t' : \rho$ if and only if ${}^{\pi}\Gamma \bowtie \pi(a) : \sigma \vdash \pi \bullet t' : \rho$ and the statement is completed by applying the rule $(\mathcal{T}_{[\mathbf{a}]})$.
- Rule$(\mathcal{T}_{\cap})$: By IH, $\Gamma \vdash t : \tau_i$ for all $i = 1, \ldots, m$ if and only if ${}^{\pi}\Gamma \vdash \pi \bullet t : \tau_i$ and this case is finished by applying the rule $(\mathcal{T}_{\cap})$.
- Rule $(\mathcal{T}_{\mathsf{f}})$: There is a chain C such that $C(\Sigma_{\mathsf{f}}) = \sigma_1 \to \cdots \to \sigma_n \to \tau$ and, for all $i = 1, \ldots, n$, $\Gamma \vdash t_i : \sigma_i$. By IH, ${}^{\pi}\Gamma \vdash \pi \bullet t_i : \sigma_i$. So, ${}^{\pi}\Gamma \vdash \pi \bullet \mathsf{f}(t_1, \ldots, t_n) : \tau$. The reverse is analogous.

□

Example 5.16. Consider $\Gamma = \{X : \mathtt{nat}, a : \mathtt{nat}\}$ and the derivable type judgement $\Gamma \vdash +(a, X) : \mathtt{nat}$, where $\Sigma_{+} = \mathtt{nat} \to \mathtt{nat} \to \mathtt{nat}$. So, it is also possible to derive ${}^{(a\,b)}\Gamma \vdash +(b, (a\,b) \cdot X) : \mathtt{nat}$, that is the result of applying $(a\,b)$ to the judgement in the object level.

$$(\mathcal{T}_{\mathsf{f}})\ \frac{(\mathcal{T}_{\mathbf{a}})\ \dfrac{}{\Gamma \vdash a : \mathtt{nat}} \qquad (\mathcal{T}_{\mathbf{X}})\ \dfrac{}{\Gamma \vdash X : \mathtt{nat}}}{\Gamma \vdash +(a, X) : \mathtt{nat}} \quad \leadsto \quad \frac{\dfrac{}{{}^{(a\,b)}\Gamma \vdash b : \mathtt{nat}}\ (\mathcal{T}_{\mathbf{a}}) \qquad \dfrac{}{{}^{(a\,b)}\Gamma \vdash (a\,b) \cdot X : \mathtt{nat}}\ (\mathcal{T}_{\mathbf{X}})}{{}^{(a\,b)}\Gamma \vdash +(b, (a\,b) \cdot X) : \mathtt{nat}}\ (\mathcal{T}_{\mathsf{f}})$$

The Lemma 5.17 asserts that two α-equivalent terms have the same typings, what is very reasonable since such terms represent the same class of objects.

Lemma 5.17 (α-equivalence preserves types)**.** *If* $\Gamma \vdash t : \sigma$ *is derivable and* $\Delta \vdash t \approx_\alpha s$ *for some* Δ*, then* $\Gamma \vdash s : \sigma$ *is also derivable.*

Proof. Induction on the derivation of α-equivalence.

- Rule $(\approx_\alpha \mathbf{a})$: $t = s = a$: it is straightforward.
- Rule $(\approx_\alpha \mathbf{X})$:$t = \pi \cdot X$ and $s = \pi' \cdot X$: since $\Gamma \vdash \pi \cdot X : \sigma$, by the Generation Lemma (5.9), there exists γ such that $X : \sigma \in \Gamma$ and $\gamma \leq \sigma$.Thus, $\Gamma \vdash \pi' \cdot X : \sigma$ by applying rule $(\mathcal{T}_\mathbf{X})$.
- Rule $(\approx_\alpha[\mathbf{a}])$ or $(\approx_\alpha[\mathbf{b}])$:$t = [a]t'$ and $s = [b]s'$: we know that $\sigma = \cap_{i=1}^{n}[\sigma_i]\gamma_i$ and that $\Gamma \bowtie a : \sigma_i \vdash t' : \gamma_i$ for all $i = 1, \ldots, n$, by the Generation Lemma (5.9). If $a = b$, then $\Delta \vdash t' \approx_\alpha s'$ and, by IH, $\Gamma \bowtie b : \sigma_i \vdash s' : \gamma_i$ and it follows that $\Gamma \vdash [b]s' : \tau$. If $a \neq b$, then $\Delta \vdash t' \approx_\alpha (a\, b) \bullet s', a\#s', (a\, b) \bullet t' \approx_\alpha s', b\#t'$. Lets consider the more complex case when $a, b \in \Gamma$. Thus,

$\Gamma \setminus \{b : \Gamma_b\} \bowtie a : \sigma_i \vdash t' : \gamma_i,$	by Lemma 5.13(2) - strengthening,
$\Gamma \setminus \{a : \Gamma_a, b : \Gamma_b\} \bowtie a : \sigma_i \vdash t' : \gamma_i,$	that is the same judgement,
$\Gamma \setminus \{a : \Gamma_a, b : \Gamma_b\} \bowtie a : \sigma_i \bowtie b : \sigma_i \vdash t' : \gamma_i,$	by Lemma 5.13(1) - weakening,
$\Gamma \setminus \{a : \Gamma_a, b : \Gamma_b\} \bowtie a : \sigma_i \bowtie b : \sigma_i \vdash (a\, b) \bullet t' : \gamma_i,$	by Lemma 5.15
$\Gamma \setminus \{a : \Gamma_a, b : \Gamma_b\} \bowtie a : \sigma_i \bowtie b : \sigma_i \vdash s' : \gamma_i,$	by induction hypothesis,
$\Gamma \setminus \{a : \Gamma_a, b : \Gamma_b\} \bowtie a : \sigma_i \vdash [b]s' : [\sigma_i]\gamma_i,$	applying rule $(\mathcal{T}_{[\mathbf{a}]})$,
$\Gamma \setminus \{a : \Gamma_a, b : \Gamma_b\} \vdash [b]s' : [\sigma_i]\gamma_i,$	by Lemma 5.13(2) - strengthening,
$\Gamma\setminus \vdash [b]s' : [\sigma_i]\gamma_i,$	by Lemma 5.13(1) - weakening,
$\Gamma\setminus \vdash [b]s' : \sigma,$	applying rule $(\mathcal{T}_\cap)$.

- Rule $(\approx_\alpha \mathsf{f})$: $t = \mathsf{f}(t_1, \ldots, t_n)$: this case follows by item 4 of Generation Lemma (5.9) and IH, applying rules $(\mathcal{T}_\mathsf{f})$ and $(\mathcal{T}_\cap)$ at the end.

□

Example 5.18. Consider a signature where $\Sigma_\forall = ([\varphi]\texttt{bool}) \to \texttt{bool}$, $\Sigma_{\mathsf{even}} = \texttt{nat} \to \texttt{bool}$ and $\Sigma_+ = \varphi \to \varphi \to \varphi$ (+ will be written infixed). By the following derivation, the term

$\forall[a]\mathsf{even}(a + X)$ has $\langle\{X : \mathtt{nat}\}, \mathtt{bool}\rangle$ as a typing.

$$
\dfrac{\dfrac{\dfrac{\dfrac{X : \mathtt{nat}, a : \mathtt{nat} \vdash a : \mathtt{nat}\ (\mathcal{T}_{\mathbf{a}}) \qquad X : \mathtt{nat}, a : \mathtt{nat} \vdash X : \mathtt{nat}\ (\mathcal{T}_{\mathbf{X}})}{X : \mathtt{nat}, a : \mathtt{nat} \vdash a + X : \mathtt{nat}}(\mathcal{T}_{\mathsf{f}})}{X : \mathtt{nat}, a : \mathtt{nat} \vdash \mathsf{even}(a + X) : \mathtt{bool}}(\mathcal{T}_{\mathsf{f}})}{X : \mathtt{nat} \vdash [a]\mathsf{even}(a + X) : [\mathtt{nat}]\mathtt{bool}}(\mathcal{T}_{[\mathbf{a}]})}{X : \mathtt{nat} \vdash \forall[a]\mathsf{even}(a + X) : \mathtt{bool}}(\mathcal{T}_{\mathsf{f}})
$$

The same typing is also valid to $\forall[b]\mathsf{even}(b + (a\,b) \cdot X)$, by Lemma 5.17, because $b\#X \vdash \forall[a]\mathsf{even}(a + X) \approx_\alpha \forall[b]\mathsf{even}(b + (a\,b) \cdot X)$.

The Subterm Lemma presented next shows that a derivation of a judgement whose type is different from ω contains typings for the subterms of the term in question.

Lemma 5.19 (Subterm lemma)**.** *If $\Gamma \vdash t : \sigma$ is derivable with a derivation $\mathcal{D}$ without any subterm typed with ω and s is a proper subterm of t, then there is a subtree of $\mathcal{D}$ that derives $\Gamma' \vdash s : \gamma$ for some Γ' and γ such that Γ' is an extension of Γ.*

Proof. By induction on the type derivation of $\Gamma \vdash t : \sigma$.

- Rules $(\mathcal{T}_{\mathbf{a}})$ and $(\mathcal{T}_{\mathbf{X}})$ are trivial because there is no proper subterm of an atom or a suspended variable.
- Rule $(\mathcal{T}_{[\mathbf{a}]})$: We have $\Gamma \bowtie a : \rho \vdash t' : \delta$, $t = [a]t'$ and $\sigma = [\rho]\delta$. If $s = t'$, it is done. If not, then s is a proper subterm of t' and, by IH, there are Γ' and γ such that $\Gamma' \vdash s : \gamma$ is a subtree in the derivation of $\Gamma \bowtie a : \rho \vdash t' : \delta$.
- Rule $(\mathcal{T}_{\cap})$: $\sigma = \tau_1 \cap \cdots \cap \tau_k$ and, for each $i = 1, \ldots, k$, $\Gamma \vdash t : \tau_i$. By IH, there are Γ_i's and γ_i's such that $\Gamma_i \vdash s : \gamma_i$ is a subtree of the derivation of $\Gamma \vdash t : \tau_i$. In this branch, it is important to observe that $k \neq 0$; otherwise, the IH would not be possible.
- Rule $(\mathcal{T}_{\mathsf{f}})$: $t = \mathsf{f}(t_1, \ldots, t_n)$, there is a chain C such that $C(\Sigma_{\mathsf{f}}) = \sigma_1 \to \cdots \to \sigma_n \to \sigma$ and $\Gamma \vdash t_i : \sigma_i$ for all $i = 1, \ldots, n$. If $s = t_i$ for some t_i, then the result follows. Otherwise, s is a proper subterm of some t_i and the result is obtained by IH.

□

5.3 Typed Matching and Typed Rewrite Relation

This section introduces the notion of typed matching, that is crucial for a rewrite relation which considers types. This matching is supposed to instantiate type variables as well

as unknowns. In that way, the pattern of a matching must include a term in context, as done in [Fernández and Gabbay, 2007], and a typing of such term. The term and type substitutions should then satisfy the conditions that are expressed in each leaf with unknowns of the type derivation. The notion of *variable environment* introduced next isolates what information is relevant to consider, eliminating the type annotations of atoms that are fresh for some unknown.

> **Definition 42** If $\Gamma \vdash \pi \cdot X : \sigma$ is a leaf in the type derivation $\mathcal{D}$ of $\Gamma' \vdash t : \gamma$, then ${}^{\pi^{-1}}\Gamma \setminus \{a : \rho \mid \Delta \vdash a\#X \text{ or } \rho = \omega\}$ is a **variable environment** of X in $\Delta \vdash \mathcal{D}$.

Notice that the atoms that cannot occur free in the instance of a variable are not considered in a variable environment as well as atoms which are annotated with ω, because they do not add information.

Example 5.20. Let $[\texttt{nat}]\varphi \to [\texttt{string}]\varphi \to \texttt{real}$ be the type declaration of a binary symbol $\texttt{g}$. Consider the judgement $X : \texttt{nat} \vdash \texttt{g}([a]X, [a](a\,b)\cdot X) : \texttt{real}$ with the following derivation:

$$\mathcal{D} = \cfrac{\cfrac{X : \texttt{nat}, a : \texttt{nat} \vdash X : \texttt{nat}(\mathcal{T}_{\mathbf{X}})}{X : \texttt{nat} \vdash [a]X : [\texttt{nat}]\texttt{nat}}\,(\mathcal{T}_{[\mathsf{a}]}) \quad \cfrac{X : \texttt{nat}, a : \texttt{string} \vdash (a\,b)\cdot X : \texttt{nat}(\mathcal{T}_{\mathbf{X}})}{X : \texttt{nat} \vdash [a](a\,b)\cdot X : [\texttt{string}]\texttt{nat}}\,(\mathcal{T}_{[\mathsf{a}]})}{X : \texttt{nat} \vdash \texttt{g}([a]X, [a](a\,b)\cdot X) : \texttt{real}}\,(\mathcal{T}_{\mathsf{f}})$$

If we observe the leaf on the left-hand side of $\mathcal{D}$, then the substitution $[X \mapsto a]$ respects the environment $\{X : \texttt{nat}, a : \texttt{nat}\}$ maintaining the type in the instance of X. On the right-hand side, a is annotated with type $\texttt{string}$ in the environment; applying the inverse of $(a\,b)$, which is itself, to the environment, we obtain $\{X : \texttt{nat}, b : \texttt{string}\}$, that is a variable environment of X in $\varnothing \vdash \mathcal{D}$. Notice that enlarging this environment with $a : \texttt{nat}$ allows us to proceed with the instance of the term keeping the designated type.

In the case that the context is not empty, for instance $\{a\#X\}$, then the variable environments of X in $\{a\#X\} \vdash \mathcal{D}$ would be $\{X : \texttt{nat}\}$ and $\{X : \texttt{nat}, b : \texttt{string}\}$.

> **Definition 43** Let $\Phi \vdash l : \tau$ be a type judgement with derivation $\mathcal{D}$. The typed matching problem $(\Phi \Vdash \nabla \vdash l : \tau)^? \approx_\alpha (\Gamma \Vdash \Delta \vdash s)$, with $unkn(\Phi, \nabla, l)$ and $Vars(\Phi, \tau)$ disjoint from $unkn(\Gamma, \Delta, s)$ and $Vars(\Gamma)$, has solution (C, θ, π) with a chain of operations C, a substitution θ and a permutation π whenever:
>
> 1. $\Delta \vdash {}^{\pi}\nabla\theta,\ {}^{\pi}l\theta \approx_\alpha s$;

2. $C({}^{\pi}\Phi)|_{\mathbb{A}} \subseteq \Gamma$, i.e., the atoms of Φ renamed with π have the annotations changed only by C in Γ; and

3. ${}^{\pi'^{-1}}\Gamma \bowtie C(\Phi') \vdash X\theta : C(\Phi)_X$, for every variable environment Φ' of X in ${}^{\pi}\Delta \vdash {}^{\pi}\mathcal{D}$, with π' the corresponding permutation in each leaf. Notice that $\Phi_X = \Phi'_X$ because renamings do not change annotations of unknowns.

Example 5.21. Take the symbol $\mathtt{g}$ as in Example 5.20. The following typed matching problem

$$(X : \mathtt{nat} \Vdash b\#X \vdash \mathtt{g}([a]X, [a](a\,b)\cdot X) : \mathtt{real})\, {}^{?}{\approx_\alpha}\, (b : \mathtt{nat} \Vdash \varnothing \vdash \mathtt{g}([b]b, [c]b))$$

has a solution $(Id, id, [X \mapsto a])$, i.e., $\varnothing \vdash b\#a$, $\mathtt{g}([a]a, [a]b) \approx_\alpha \mathtt{g}([b]b, [c]b)$ and, taking the variables environments in $b\#X \vdash \mathcal{D}$ (Example 5.20), $b : \mathtt{nat}, X : \mathtt{nat}, a : \mathtt{nat} \vdash a : \mathtt{nat}$ is derivable as well as $a : \mathtt{nat}, X : \mathtt{nat}, b : \mathtt{string} \vdash a : \mathtt{nat}$.

Example 5.22. Consider $\prod$ with the type declaration of Example 5.1 and take the binary symbol $/$ (with $\Sigma_/ = \mathtt{nat} \to \mathtt{nat} \to \mathtt{rat}$ and infixed notation), the unary symbol s (with $\Sigma_\mathsf{s} = \mathtt{nat} \to \mathtt{nat}$) and the nullary symbol $\mathsf{0}$ (with $\Sigma_\mathsf{0} = \mathtt{nat}$). The following typed matching problem

$$\left(X : \mathtt{real} \cap \mathtt{rat} \cap \mathtt{nat}, Y : \mathtt{nat}, Z : \mathtt{nat} \Vdash i\#X \vdash \prod_{i=Y}^{Z} X : \mathtt{real} \cap \mathtt{rat} \cap \mathtt{nat}\right) {}^{?}{\approx_\alpha}$$

$$\left(m : \mathtt{nat} \Vdash \varnothing \vdash \prod_{i=1}^{m} \left(\frac{\mathsf{s(0)}}{\mathsf{s(s(0))}}\right)\right)$$

has a solution $\left(L_{\langle\langle\varnothing,\mathtt{real}\cap\mathtt{rat}\cap\mathtt{nat}\rangle,\langle\varnothing,\mathtt{rat}\rangle\rangle}, id, \left[X \mapsto \frac{\mathsf{s(0)}}{\mathsf{s(s(0))}}\right] \circ [Y \mapsto \mathsf{s(0)}] \circ [Z \mapsto m]\right)$. Observe that the instance of the term satisfies the freshness constraint of the pattern in the empty context.

The next lemma is inspired by the presentation given in [Fairweather, 2014].

Lemma 5.23 (Typed Nominal Pattern Matching Lemma)**.** *If* $\Phi \vdash l : \sigma$ *and* $(\Phi \Vdash \nabla \vdash l : \sigma)\, {}^{?}{\approx_\alpha}\, (\Gamma \Vdash \Delta \vdash s) = (C, \theta, \pi)$*, then* $\Gamma \vdash {}^{\pi}l\theta : C(\sigma)$*.*

Proof. Firstly, by meta-level equivariance (Lemma 5.14), soundness of type operations in C (Lemmas 5.10, 5.11 and 5.12) and weakening (Lemma 5.13(1)), $\Gamma \vdash {}^{\pi}l : C(\sigma)$ is derivable. To include the term-substitution θ, we proceed by induction on the derivation of this last judgement. Assume that $C(\sigma) \neq \omega$, since the opposite would be achieved using rule $(\mathcal{T}_\cap)$.

- Rule ($\mathcal{T}_{\mathbf{a}}$): this part is trivial because θ has no effect on ${}^{\pi}l$.

- Rule ($\mathcal{T}_{\mathbf{X}}$): ${}^{\pi}l = \pi' \cdot X$. By Condition 3 of Definition 43,

$$^{\pi'^{-1}}\Gamma \bowtie C(^{\pi'^{-1}\circ\pi}\Phi \setminus \{a:\rho \mid {}^{\pi}\nabla \vdash a\#X\}) \vdash X\theta : C(^{\pi}\Phi)_X,$$

  what implies that

$$\Gamma \bowtie C(^{\pi}\Phi \setminus \{a:\rho \mid {}^{\pi}\nabla \vdash a\#X\}) \vdash \pi' \cdot X\theta : C(^{\pi}\Phi)_X$$

  using object-level equivariance (Lemma 5.15). Since $C(^{\pi}\Phi)|_{\mathbb{A}} \subseteq \Gamma$ and $C(^{\pi}\Phi)_X \leq C(\sigma)$ (by Lemmas 5.5 and 5.8 and the definition of lifting), the derivation of $\Gamma \vdash \pi' \cdot X\theta : C(\sigma)$ follows easily by Lemma 5.10.

- Rule ($\mathcal{T}_{[\mathbf{a}]}$): $l = [a]l'$ and $C(\sigma) = [\sigma']\tau$. We have $\Gamma \bowtie \pi(a) : \sigma' \vdash {}^{\pi}l' : \tau$ and, since (C, θ, π) is also a solution for the matching problem $(\Phi \bowtie a : \sigma' \Vdash \nabla \vdash l' : \tau)\, {}^{?}\!\!\approx_{\alpha} (\Gamma \bowtie a : \sigma' \Vdash \Delta \vdash (a\, b) \bullet s')$ for $s = [b]s'$, so one has $\Gamma \bowtie \pi(a) : \sigma' \vdash {}^{\pi}l'\theta : \tau$ by IH. Applying rule ($\mathcal{T}_{[\mathbf{a}]}$), the result holds.

- Rule ($\mathcal{T}_{\cap}$): $C(\sigma) = \tau_1 \cap \cdots \cap \tau_k$. It holds that $\Gamma \vdash {}^{\pi}t : \tau_i$. Since $C(\sigma) \neq \omega$ and so $k \neq 0$, by IH, $\Gamma \vdash {}^{\pi}t\theta : \tau_i$ holds for each $i = 1, \ldots, k$. Applying rule ($\mathcal{T}_{\cap}$), $\Gamma \vdash {}^{\pi}t\theta : C(\sigma)$ is obtained.

- Rule ($\mathcal{T}_{\mathsf{f}}$): $l = \mathsf{f}(l_1, \ldots, l_n)$, there is C' such that $C'(\Sigma_{\mathsf{f}}) = \sigma_1 \to \cdots \to \sigma_n \to C(\sigma)$ and $\Gamma \vdash {}^{\pi}l_i : \sigma_i$, for $i = 1, \ldots, n$. Since the environment is not changed in this step, by IH, we have $\Gamma \vdash {}^{\pi}l_i\theta : \sigma_i$. The result is obtained applying rule ($\mathcal{T}_{\mathsf{f}}$).

□

The last lemma is specially important because it ensures that the term which matches the pattern has the same typing modified only by the type operations specified in the solution and some additional type annotations (of fresh atoms and/or new unknowns). Notice that, by Definition 43, such type check is not necessary when verifying if a triplet is a solution of the typed matching problem.

The next lemma states that, for an instance of a term with different unknowns, the term preserves the typing of the instance by adding only the unknowns' type annotations.

Lemma 5.24 (Inversion Substitution Lemma). *Let $\Gamma \vdash t\theta : \gamma$ be a derivable judgement such that $unkn(t) \cap unkn(t\theta) = \varnothing$. Then, there is an environment Γ^* which is Γ updated with annotations of variables in $unkn(t)$ and such that $\Gamma^* \vdash t : \gamma$ is derivable.*

Proof. The proof is by induction on the derivation of $\Gamma \vdash t\theta : \gamma$, but first we consider the case when $t = \pi \cdot X$. In this case, $\Gamma \bowtie X : \gamma \vdash \pi \cdot X : \gamma$ is derivable. Now, assume that t is not a suspended variable.

- Rule $(\mathcal{T}_{\mathsf{a}})$: $t\theta = a$ and, since t is not a variable, $t = a$.
- Rule $(\mathcal{T}_{\mathbf{X}})$: the only way this happens is when t is a suspended variable because the variables are different from the ones of $t\theta$.
- Rule $(\mathcal{T}_{[\mathsf{a}]})$: $t = [a]t'$, $t\theta = [a]t'\theta$, $\gamma = [\sigma]\tau$ and $\Gamma \bowtie a : \sigma \vdash t'\theta : \tau$. By IH, there is Γ^* such that $\Gamma^* \bowtie a : \sigma \vdash t' : \tau$ and, applying rule $(\mathcal{T}_{[\mathsf{a}]})$, one obtains $\Gamma^* \vdash [a]t' : [\sigma]\tau$.
- Rule $(\mathcal{T}_{\cap})$: for $i = 1, \ldots, k$, $\Gamma \vdash t\theta : \tau_i$ and $\gamma = \tau_1 \cap \cdots \cap \tau_k$. By IH, $\Gamma^* \vdash t : \tau_i$ and, applying rule $(\mathcal{T}_{\cap})$, $\Gamma^* \vdash t : \gamma$.
- Rule $(\mathcal{T}_{\mathsf{f}})$: $t = \mathsf{f}(t_1, \ldots, t_n)$, $\Gamma \vdash t_i\theta : \sigma_i$ for $i = 1, \ldots, n$ and there exists C such that $C(\Sigma_{\mathsf{f}}) = \sigma_1 \to \cdots \to \sigma_n \to \gamma$. By IH, there is Γ^* such that $\Gamma^* \vdash t_i : \sigma_i$. Using rule $(\mathtt{f})$, one has $\Gamma^* \vdash \mathsf{f}(t_1, \ldots, t_n) : \gamma$.

$\square$

Definition 44 A pair $\langle \Pi, \rho \rangle$ of an environment and a type is called a **principal pair** for a term t if it is a typing and, for every other typing $\langle \Gamma, \sigma \rangle$, there exist a chain of operations C such that $C(\langle \Pi, \rho \rangle) = \langle \Gamma, \sigma \rangle$.

An important feature of this system is that a term may not have a principal pair, as the next example demonstrate.

Example 5.25. If the declaration of $\prod$ is $\Sigma_{\prod} = [\mathtt{nat}]\mathtt{real} \to \mathtt{nat} \to \mathtt{nat} \to \mathtt{real} \cap [\mathtt{nat}]\mathtt{rat} \to \mathtt{nat} \to \mathtt{nat} \to \mathtt{rat} \cap [\mathtt{nat}]\mathtt{nat} \to \mathtt{nat} \to \mathtt{nat} \to \mathtt{nat}$, then the principal pair of $\prod_{i=Y}^{Z} X$ (sugar of $\prod([i]X, Y, Z)$) does not exist. Notice that $\langle \{X : \mathtt{real}, Y : \mathtt{nat}, Z : \mathtt{nat}\}, \mathtt{real} \rangle$ is a typing of this term and that substitutions and expansions do not change the type declaration, because it has no type variable. Then the least type with respect to $\leq$ that can be derived to $\prod_{i=Y}^{Z} X$ is $\mathtt{real} \cap \mathtt{rat} \cap \mathtt{nat}$. However, there is no type annotation of X that could derive such type and be larger than $\mathtt{real}$ with respect to $\leq$.

On the other hand, if $\Sigma_{\prod} = [\mathtt{nat}]\varphi \to \mathtt{nat} \to \mathtt{nat} \to \varphi$, as in Example 5.1, the principal pair of the term is the typing $\langle \{X : \varphi, Y : \mathtt{nat}, Z : \mathtt{nat}\}, \varphi \rangle$.

Definition 45 A **typed rewrite rule** $\Phi \Vdash \nabla \vdash l \to r : \rho$ is a rule such that $\langle \Phi, \rho \rangle$ is a principal pair for l and it is a typing for r, in such a way that the variable

environments of each unknown are equal in the derivations of $\Phi \vdash l : \rho$ and of $\Phi \vdash r : \rho$.

The condition of having equal variable enviroments throughout derivations is called diamond property in [Fairweather, 2014] and linearity property in [Fairweather and Fernández, 2016]. There, only derivations with this property are valid. Here, it is required only for rewrite rules in order to obtain the Subject Reduction Lemma, as proved in Subsection 5.3.1.

Example 5.26. The rules in the system of Table 5.3 are typed rewrite rules. Take, for instance, the fifth rule. The type derivation for the left-hand side is expressed following.

$$\dfrac{\dfrac{\dfrac{\dfrac{X : \varphi, a : \omega, b : \omega \vdash X : \varphi \ (\mathcal{T}_{\mathbf{X}})}{X : \varphi, a : \omega \vdash [b]X : [\omega]\varphi}\ (\mathcal{T}_{[\mathbf{a}]})}{X : \varphi, a : \omega \vdash \mathsf{Lam}[b]X : \omega \to \varphi}\ (\mathcal{T}_{\mathsf{f}})}{X : \varphi \vdash [a]\mathsf{Lam}[b]X : [\omega](\omega \to \varphi)}\ (\mathcal{T}_{[\mathbf{a}]}) \qquad X : \varphi \vdash Z : \omega \ (\mathcal{T}_{\cap})}{X : \varphi \vdash \mathsf{Sub}([a]\mathsf{Lam}[b]X, Z) : \omega \to \varphi}\ (\mathcal{T}_{\mathsf{f}})$$

Notice that this typing is a principal pair of $\mathsf{Sub}([a]\mathsf{Lam}[b]X, Z)$ and so is $\langle\{X : \varphi\}, \varphi' \to \varphi\rangle$ because each of them can be obtained by the other and they generate any other typing for this term. Applying the lifting $L_{\langle\langle\{X:\varphi\},\omega\to\varphi\rangle,\langle\{X:\varphi\},\varphi'\to\varphi\rangle\rangle}$ to $\langle\{X : \varphi\}, \omega \to \varphi\rangle$ gives us $\langle\{X : \varphi\}, \varphi' \to \varphi\rangle$ and this second one is obtained applying the substitution $(\varphi' \mapsto \omega)$ to the former one. The variable environment of X and Z in this derivation with $\{b\#Z\}$ is $\{X : \varphi\}$. For the term in the right-hand side, we have the same variable environment, as we can see in the type derivation below.

$$\dfrac{\dfrac{\dfrac{\dfrac{X : \varphi, b : \omega, a : \omega \vdash X : \varphi \ (\mathcal{T}_{\mathbf{X}})}{X : \varphi, b : \omega \vdash [a]X : [\omega]\varphi}\ (\mathcal{T}_{[\mathbf{a}]}) \qquad X : \varphi, b : \omega \vdash Z : \omega \ (\mathcal{T}_{\cap})}{X : \varphi, b : \omega \vdash \mathsf{Sub}([a]X, Z) : \varphi}\ (\mathcal{T}_{\mathsf{f}})}{X : \varphi \vdash [b]\mathsf{Sub}([a]X, Z) : [\omega]\varphi}\ (\mathcal{T}_{[\mathbf{a}]})}{X : \varphi \vdash \mathsf{Lam}[b]\mathsf{Sub}([a]X, Z) : \omega \to \varphi}\ (\mathcal{T}_{\mathsf{f}})$$

Definition 46 Let $R = \Phi \Vdash \nabla \vdash l \to r : \rho$ be a typed rewrite rule. A **nominal typed rewrite step** $\Gamma \Vdash \Delta \vdash s \to_{\tau}^{R} t$ holds whenever $(\Phi \Vdash \nabla \vdash l : \rho)\,{}^{?}{\approx_\alpha}\, (\Gamma \Vdash \Delta \vdash s|_p) = (C, \theta, \pi)$, for some $p \in Pos(s)$, and $\Delta \vdash t \approx_\alpha s[p \leftarrow {}^{\pi}r\theta]$.

Table 5.3: Λ_x

Symbols	Type Declarations	Arity
App :	$(\varphi_1 \to \varphi_2) \to \varphi_1 \to \varphi_2$	2
Lam :	$([\varphi_1]\varphi_2) \to \varphi_1 \to \varphi_2$	1
Sub :	$([\varphi_1]\varphi_2) \to \varphi_1 \to \varphi_2$	2

Rules :

$$\begin{array}{rrcl}
X : \varphi \Vdash \varnothing \vdash & \mathsf{App}(\mathsf{Lam}\,[a]X, Y) & \longrightarrow & \mathsf{Sub}([a]X, Y) : \varphi \\
X : \varphi \Vdash a\#X \vdash & \mathsf{Sub}([a]X, Y) & \longrightarrow & X : \varphi \\
Y : \varphi \Vdash \varnothing \vdash & \mathsf{Sub}([a]a, Y) & \longrightarrow & Y : \varphi \\
X : \varphi_1 \to \varphi_2, Y : \varphi_1 \Vdash \varnothing \vdash & \mathsf{Sub}([a]\mathsf{App}(X, Y), Z) & \longrightarrow & \mathsf{App}(\mathsf{Sub}([a]X, Z), \mathsf{Sub}([a]Y, Z)) : \varphi_2 \\
X : \varphi \Vdash b\#Z \vdash & \mathsf{Sub}([a]\mathsf{Lam}\,[b]X, Z) & \longrightarrow & \mathsf{Lam}\,[b]\mathsf{Sub}([a]X, Z) : \omega \to \varphi
\end{array}$$

Example 5.27. Take the first rule of the system in Table 5.3, which is typed and represents the β-reduction in the context of explicit substitutions. The typed matching problem

$$(X : \varphi \Vdash \varnothing \vdash \mathsf{App}(\mathsf{Lam}[a]X, Y))\ {}^{?}{\approx}_\alpha\ (\varnothing \Vdash \varnothing \vdash \mathsf{App}(\mathsf{Lam}[b]\mathsf{App}(b, b), \mathsf{Lam}[b]b))$$

has a solution $S = ((\varphi \mapsto (\varphi' \to \varphi')), (a\,b), [X \mapsto \mathsf{App}(b, b)] \circ [Y \mapsto \mathsf{Lam}[b]b])$. Look at the type derivation of the left-hand side of the rule.

$$\dfrac{\dfrac{\dfrac{X : \varphi, a : \omega \vdash X : \varphi\ (\mathcal{T}_{\mathbf{X}})}{X : \varphi \vdash [a]X : [\omega]\varphi}\,(\mathcal{T}_{[\mathbf{a}]})}{X : \varphi \vdash \mathsf{Lam}[a]X : \omega \to \varphi}\,(\mathcal{T}_{\mathsf{f}}) \qquad X : \varphi \vdash Y : \omega\ (\mathcal{T}_{\cap})}{X : \varphi \vdash \mathsf{App}(\mathsf{Lam}[a]X, Y) : \varphi}\,(\mathcal{T}_{\mathsf{f}})$$

To verify the solution S, it is necessary to check that the permutation and the term substitution satisfy the matching for the terms and freshness contexts, and to see if typability in the instantiated variable environments continues working. The variable environment of Y is trivial because the type is ω. For X, one has that $\varnothing \vdash \mathsf{Lam}[a]a : \varphi' \to \varphi'$.

By Lemma 5.23, $\varnothing \vdash \mathsf{App}(\mathsf{Lam}[b]\mathsf{App}(b, b), \mathsf{Lam}[b]b) : \varphi' \to \varphi'$ is derivable. Indeed, we have the derivations below of the "self-application", of the "identity" and of the term

considered above. Consider $\sigma = \varphi' \to \varphi'$.

$$\mathcal{D}' = \quad (\mathcal{T}_{\mathsf{f}})\,\dfrac{(\mathcal{T}_{[\mathbf{a}]})\,\dfrac{(\mathcal{T}_{\mathsf{f}})\,\dfrac{b : (\sigma \to \sigma) \cap \sigma \vdash b : \sigma \to \sigma \ (\mathcal{T}_{\mathbf{a}}) \qquad b : (\sigma \to \sigma) \cap \sigma \vdash b : \sigma \ (\mathcal{T}_{\mathbf{a}})}{b : (\sigma \to \sigma) \cap \sigma \vdash \mathsf{App}(b, b) : \sigma}}{\varnothing \vdash [b]\mathsf{App}(b, b) : [(\sigma \to \sigma) \cap \sigma]\sigma}}{\varnothing \vdash \mathsf{Lam}[b]\mathsf{App}(b, b) : ((\sigma \to \sigma) \cap \sigma) \to \sigma}$$

$$\mathcal{D}'' = \quad (\mathcal{T}_{\cap})\,\dfrac{(\mathcal{T}_{\mathsf{f}})\,\dfrac{(\mathcal{T}_{[\mathbf{a}]})\,\dfrac{b : \sigma \vdash b : \sigma \ (\mathcal{T}_{\mathbf{a}})}{\varnothing \vdash [b]b : [\sigma]\sigma}}{\varnothing \vdash \mathsf{Lam}[b]b : \sigma \to \sigma} \qquad \dfrac{\dfrac{b : \varphi \vdash b : \varphi \ (\mathcal{T}_{\mathbf{a}})}{\varnothing \vdash [b]b : [\varphi]\varphi}\,(\mathcal{T}_{[\mathbf{a}]})}{\varnothing \vdash \mathsf{Lam}[b]b : \sigma}\,(\mathcal{T}_{\mathsf{f}})}{\varnothing \vdash \mathsf{Lam}[b]b : (\sigma \to \sigma) \cap \sigma}$$

$$(\mathcal{T}_{\mathsf{f}})\,\dfrac{\dfrac{\mathcal{D}'}{\varnothing \vdash \mathsf{Lam}[b]\mathsf{App}(b, b) : ((\sigma \to \sigma) \cap \sigma) \to \sigma} \qquad \dfrac{\mathcal{D}''}{\varnothing \vdash \mathsf{Lam}[b]b : (\sigma \to \sigma) \cap \sigma}}{\varnothing \vdash \mathsf{App}(\mathsf{Lam}[b]\mathsf{App}(b, b), \mathsf{Lam}[b]b) : \varphi' \to \varphi'}$$

5.3.1 Subject Reduction

Now the main theorem of Subject Reduction is presented, i.e., the result that proves that typed rewrite steps using typed rewrite rules preserve types under the condition of uniformity of such rules. Notice that uniformity (Definition 30) is crucial to have preservation of typings as we can see in the following example. This condition was inspired by [Fairweather and Fernández, 2016].

Example 5.28. The rule $R = (X : \varphi \Vdash \varnothing \vdash X \longrightarrow (a\,b) \cdot X : \varphi)$ is typed but it does not preserves typings. Observe that the only variable environment for the left-hand and right-hand side is $\{X : \varphi\}$ and that the matching problem

$$(X : \varphi \Vdash \varnothing \vdash X : \varphi) {}^{?}{\approx}_{\alpha} \ (a : \varphi \Vdash \varnothing \vdash a : \varphi)$$

has a solution $(Id, id, [X \mapsto a])$, but $\{a : \varphi\}$ it is not enough to type b, the resulting term of the typed rewrite step. This occurs because non-uniform rules can introduce atoms that are fresh in the left-hand side into the right-hand side.

Theorem 5.29 (Subject Reduction)**.** *Given a uniform typable rewrite rule $R = \Phi, \nabla \vdash l \to r : \sigma$, if $\Gamma \vdash s : \gamma$ and $\Gamma \Vdash \Delta \vdash s \to_{\tau}^{R} t$, then $\Gamma \vdash t : \gamma$.*

Proof. By the Subterm Lemma, there is a judgement $\Gamma' \vdash s|_p : \gamma'$ whose derivation is part of the derivation of $\Gamma \vdash s : \gamma$ and p is the position where the redex occurs. Indeed, this

lemma is considered when $s|_p$ is not inside some s' with type ω in such derivation. In this case, we can always assign the type ω for the corresponding subterm t' of t.

Since $\langle \Phi, \sigma \rangle$ is a typing of l, by Lemma 5.23, $\Gamma' \vdash {}^{\pi}l\theta : C(\sigma)$, for a solution (C, θ, π) for the matching problem $(\Phi \Vdash \nabla \vdash l : \sigma)\,{}^{?}{\approx_\alpha}\,(\Gamma' \Vdash \Delta \vdash s|_p)$. Insofar as $\Delta \vdash {}^{\pi}l\theta \approx_\alpha s|_p$, it holds also that $\Gamma' \vdash {}^{\pi}l\theta : \gamma'$. By Lemma 5.24, there exists Γ^* that differs from Γ' only in the unknowns which are in l and such that $\Gamma^* \vdash {}^{\pi}l : \gamma'$.

By the principality of $\langle \Phi, \sigma \rangle$ for l, $\langle {}^{\pi}\Phi, \sigma \rangle$ is principal for ${}^{\pi}l$, by Lemma 5.14 (meta-level equivariance). So, there exists C' such that $\langle \Gamma^*, \gamma' \rangle = C'(\langle {}^{\pi}\Phi, \sigma \rangle)$ and (C', θ, π) is also a solution for the typed matching problem $(\Phi \Vdash \nabla \vdash l : \sigma)\,{}^{?}{\approx_\alpha}\,(\Gamma^* \Vdash \Delta \vdash s|_p)$. To conclude the proof, we need to show that $\Gamma' \vdash {}^{\pi}r\theta : \gamma'$, what can be achieved by using the Matching Lemma (Lemma 5.23) if we are able to prove that (C', π, θ) is a solution of the typed matching problem

$$(\Phi \Vdash \nabla \vdash r : \sigma)\,{}^{?}{\approx_\alpha}\,(\Gamma^* \Vdash \Delta \vdash {}^{\pi}r\theta)$$

because the additional unknowns that are annotated in Γ^* do not occur in ${}^{\pi}r\theta$.

In fact, one must demostrate that, for all $X \in \mathit{unkn}(r)$, ${}^{\pi'^{-1}}\Gamma^* \bowtie C'(\Phi') \vdash X\theta : C'(\Phi)_X$ is derivable, as described in Definition 43, for π' in ${}^{\pi}r$. The necessary type annotations in ${}^{\pi'^{-1}}\Gamma^* \bowtie C'(\Phi')$ to type the term $X\theta$ are of the unknowns and free atoms introduced by θ, unless they are typed with ω. The different permutations do not change the type annotations of unknowns. So we will concentrate on the free atoms of $X\theta$.

Take a a free atom in $X\theta$. Consider $\pi_1, \ldots, \pi_k$ the permutations that accompany X in ${}^{\pi}l$. Since ${}^{\pi_i^{-1}}\Gamma^* \bowtie C'(\Phi') \vdash X\theta : C'(\Phi)_X$ is derivable for all $i = 1, \ldots, k$, we know that $a : \rho \in C'(\Phi')$ or $a : \rho \in {}^{\pi_i^{-1}}\Gamma^*$ for some $\rho \neq \omega$. In the first case, it is all right because $a : \rho$ would be also in ${}^{\pi'^{-1}}\Gamma^* \bowtie C'(\Phi')$. In the second case, $\pi_i(a) : \rho \in \Gamma^*$ for all $i = 1, \ldots, k$. If $\pi'(a) = \pi_i(a)$ for some i, then it is done, because $a : \rho$ would be in ${}^{\pi'^{-1}}\Gamma^*$. Otherwise, by the uniformity of the rule, $\pi'(a)$ would be abstracted over $\pi' \cdot X$. However, in this case, $a : \rho$ should be initially in $C'(\Phi')$, as already considered. $\square$

Chapter 6

Conclusions and Future Work

In this thesis, we presented results on three main aspects with respect to the nominal setting. First, a formalisation of important properties involving a nominal unification algorithm was developed, what provided an initial background about nominal setting in the proof assistant PVS. Secondly, we explored some criteria that guarantee confluence and local confluence in the context of nominal rewriting. Last, we extended the study on types for nominal terms developing an intersection type system with the subject reduction property.

Regarding the *theory* `nominal unification` developed in PVS, a nominal unification algorithm that only takes terms as parameters was presented. Unlike other approaches, which use transformation rules and take the corresponding freshness problems as part of the unification problem, here we have specified a function that can compute freshness problems separately. Our nominal unification algorithm is more straightforward and closer to the ones that implement first-order unification.

Additionally, we formalised transitivity for $\approx_\alpha$ in a direct manner without using a weak intermediate relation as in [Urban, 2010]. Here, the proof was based on elementary lemmas about permutations, freshness and α-equivalence; such lemmas are well-known in the context of nominal unification. In [Urban, 2010], the same auxiliary lemmas to demonstrate transitivity were proved, including some extra lemmas to deal with this weak-equivalence. We believe that the current formalisation of transitivity of $\approx_\alpha$ is simpler in the sense that it only uses the essential notions and results. Symmetry of $\approx_\alpha$ is also formalised independently from transitivity, diverging from [Urban et al., 2004, Urban, 2010].

The style of proofs formalised here could have been formalised in any higher-order proof assistant; PVS was chosen with the goal of enriching the libraries for term rewriting systems. Important features of PVS such as dependent types can be replaced by other mechanisms in Isabelle/HOL, for instance. For example, the substitution generated in the computation of *unify*($\mathtt{t}, \mathtt{s}$) must be of type `Subs_unif(t,s)` (this is the PVS specification

for the type $Subs((\mathtt{t}, \mathtt{s}))$ in Definition 17) in order to prove termination. In Isabelle/HOL, this is treated by defining substitutions in a slightly different way. PVS also allows to use type variables when defining a theory; those variables can be parameterised when such theory is imported by another one. In Isabelle/HOL, parameterising theories is not straightforward (from our point of view), but functions can be defined polymorphically, which provides different feasible solutions for the same kind of formalisation. A formalisation in Isabelle/HOL would bring out the possibility of a direct comparison regarding the previous formalisations of unification in [Urban, 2004], but it should be emphasised that the advantages of the current formalisation arise from the differences in the theoretical proofs (Section 3.2).

With respect to the study of (local) confluence for nominal rewriting, we have presented easy-to-check criteria in rewrite theories, improving previous criteria [Fernández and Gabbay, 2007, Suzuki et al., 2015]. The theorem of confluence of orthogonal uniform NRSs and the Critical Pair Criterion were first shown in [Fernández and Gabbay, 2007], but the notion of orthogonality prohibited ambiguity between permuted variants of the same rule and the second result required joinability of the CP generated by such overlaps. [Suzuki et al., 2015] relaxed the conditions over the first theorem by allowing permuted versions of rules, but a new condition called "α-stability" must be provided in turn. The authors also presented a criterion for α-stability called ASP.

Inspired by [Suzuki et al., 2015], we relaxed the CP Criterion by avoiding the check of joinability for CPs of permuted versions of a rule, also adding the condition of α-stability. In parallel, this result was explored in [Suzuki et al., 2016] too. However, our results include an additional criterion of α-stability, namely closedness of rules, which is simple to check with a nominal matching problem, the non-equivariant problem, that can be solved in linear time [Calvès and Fernández, 2010].

Our contributions embrace the extension of such theorems on confluence of NRSs to closed rewriting, an efficient notion of rewriting, even for non-closed rules. This extension was specially important not only for the sake of efficiency, but also theoretically: our results do not require α-stability or uniformity, unlike the versions presented for the standard nominal rewriting. Those results can also be found in [Ayala-Rincón et al., 2016a], written in collaboration with M. Ayala-Rincón, M. Fernández and M. Gabbay.

Concerning intersection types, we have chosen to present an adapted Essential System, presented first in [van Bakel, 1993] in the context of λ-calculus. Here, the syntax of types is extended with user defined type constructors and an abstraction type constructor. These extra types required modifications on the construction of the type ordering as well as of the type operations of lifting, substitution and expansion. An specialised type

inference system for nominal terms was also built. The notion of typability differs from [Fairweather, 2014] because there is no restriction over the permutations of a suspension and no condition over type derivations such as "diamond property" or "linearity".

Expected results over this type system could be achieved in this thesis. For instance, we have compatibility of type substitution and expansion with the type ordering, the equivariance of typings under meta and object-level action of permutations, and preservation of typings for α-equivalent terms. With respect to rewriting, the notion of matching needed to be replaced by a typed matching as well as the notion of rewriting. The constraints that must be imposed are due to the possibly capturing nature of our substitutions. With the adaptations on the definitions and the condition of uniformity on the rewrite rules, we were able to prove preservation of typing under typed rewrite steps, i.e., the subject reduction property.

Future work: Although nominal approaches have several advantages in the treatment of bound variables, there is still work to be done regarding the study of relevant computational properties. At a first glance, a subsequent study to be done is applying nominal unification for the construction of a nominal completion algorithm *à la* Knuth-Bendix as part of a PVS development for nominal rewriting. A completion algorithm for closed nominal rewriting systems is provided in [Fernández and Rubio, 2012].

Besides, the results on confluence of this thesis may extend the PVS *theory* of `nominal unification`, since they have not been formalised yet in the context of nominal setting. To do so, it would be necessary to provide the proper definitions of nominal rewriting using equivariant matching and closed nominal rewriting, with nominal equivariance-free matching. In the context of TRSs, confluence of orthogonal systems and the Critical Pair Criterion are formalised in PVS, as shown in [Galdino and Ayala-Rincón, 2010] and [Rocha-Oliveira et al., 2016].

Another possible application of this formalisation of the nominal unification algorithm is in the verification of nominal resolution approaches as done, for instance, in the propositional case in [Constable and Moczydlowski, 2009].

In our intersection type system, it is desirable to reach some normalisation results with respect to nominal rewriting. Unlike the λ-calculus or explicit substitutions calculi, our rules are not known, what can make such results on normalisation very challenging. A similar work was done for TRSs in [van Bakel and Fernández, 1997]. The notion of typed rewriting can also be extended to the closed rewriting relation, as done for the polymorphic system in [Fairweather, 2014]. There, it is shown that, although the typed rewriting is more expressive, the typed closed rewriting is more efficient and most of the systems of interest can be modelled by this last approach.

Additionally, criteria to guarantee the principal pair property have to be investigated. We do believe that such feature can be achieved for some restricted version of our type system, since the one presented in [van Bakel et al., 1996], which combines TRSs with the λ-calculus, has the principal pair property for an intersection type system. That system is close to ours in the sense that it possesses function symbols with type declarations and λ-abstractions.

Index

α-equivalence, 25, 46, 64
judgement, 13
transitivity, 25, 27
α-stability, 45–47

abstraction, 12, 28, 33
arity, 11, 12, 53, 71
atom, 11
set, 14

closed, 41
rewriting, 42, 47, 48
rule, 41, 46, 47
term in context, 41, 42
confluence, 15, 16, 40, 43, 47, 51
local, 40, 49
critical pair, 43–45

depth, 31, 32
derivable, 13
derivable judgement, 59
difference set, 13

equivariance, 39, 40, 43, 52, 63, 64

formula
antecedent, 16
consequent, 16
fresh critical pair, 49
fresh overlap, 49
proper, 49
root permutative, 49
fresh quasi-orthogonality, 49, 51
freshened variant, 41, 42, 49, 50
freshness, 24, 30, 32
constraint, 13
context, 13, 24, 29, 39, 41
judgement, 13
function
application, 12
symbol, 11, 71

identity, 13
induction scheme, 19
intersection type, 54

joinable, 16, 43, 49

last type variable, 57, 58
left-linear, 45

matching, 40–42, 47
meta-variables, 12
mgu, 35, 43, 49

name, 11
nominal
rewriting, 39
nominal rewriting, 36
nominal term, 12, 21, 54

orthogonality, 45
overlap, 43
permutative, 44
proper, 44
root permutative, 44, 45
trivial, 43

pair, 12, 32

permutation, 11, 22, 39
  action, 11, 12, 23
  meta-action, 39, 54
permuted variant, 40, 43, 44
position, 14, 39
principal pair, 69, 70
proof assistant, 15, 17
proof rule, 16, 17
PVS, 12, 15
  command, 16, 17
  datatypes, 19
  measure, 20, 22, 32
  ordering, 20
  TCC, 15
  theory, 15, 21, 22

quasi-orthogonality, 45

replacement, 14
rewrite
  judgement, 38
  peak, 40, 45, 50
  reflexive transitive closure, 39, 42
  rule, 38, 39
  step, 39
  theory, 38, 39, 45

sequent, 16
sequent calculus, 15
signature, 11
strict type, 53
substitution, 12, 13, 23, 33, 34, 39, 41, 46, 68, 69
  action, 13
  nuclear, 13, 23
subterm, 14
suspension, 12, 22
swapping, 11, 22

term in context, 41, 43
term-former, 11
termination, 22, 33, 40
TRS, 15
tuple, 12
  empty, 12
type, 53
  abstraction, 54, 59
  arrow, 54, 59
  constructor, 53
  declaration, 54, 71
  environment, 54
  expansion, 55, 58–61
  inference system, 60
  judgement, 59
  lifting, 55, 56, 60
  matching, 67, 68, 72
  operation, 55, 60
  ordering, 55, 57, 59
  rewrite rule, 70
  rewriting, 71, 73
  strengthening, 62
  subject reduction, 73
  substitution, 55–57, 60, 61
  update, 54
  variable, 53, 58
  weakening, 62

unification, 31, 42, 43, 49
  completeness, 35
  function, 32
  soundness, 34
uniformity, 44, 45
unknown, 11
  set, 14

variable environment, 66, 71

weak equivalence, 28

References

[Aoto and Kikuchi, 2016] Aoto, T. and Kikuchi, K. (2016). A Rule-Based Procedure for Equivariant Nominal Unification. Work presented in HOR 2016, http://www.nue.ie.niigata-u.ac.jp/ aoto/research/papers/report/equnif.pdf. 3

[Avelar et al., 2014] Avelar, A. B., Galdino, A. L., de Moura, F. L. C., and Ayala-Rincón, M. (2014). First-order Unification in the PVS Proof Assistant. *Logic Journal of the IGPL*, 22(5):758–789. 4, 17, 23

[Ayala-Rincón and Kamareddine, 2001] Ayala-Rincón, M. and Kamareddine, F. (2001). Unification via the λs_e-Style of Explicit Substitution. *Logic Journal of the IGPL*, 9(4):489–523. 2

[Ayala-Rincón et al., 2016a] Ayala-Rincón, M., Fernández, M., Gabbay, M. J., and Rocha-Oliveira, A. C. (2016a). Checking Overlaps of Nominal Rewriting Rules. *Electronic Notes in Theoretical Computer Science*, 323:39 – 56. Proceedings of the Tenth Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2015). 3, 8, 9, 79

[Ayala-Rincón et al., 2016b] Ayala-Rincón, M., Fernández, M., and Rocha-Oliveira, A. C. (2016b). Completeness in PVS of a Nominal Unification Algorithm. *Electronic Notes in Theoretical Computer Science*, 323:57 – 74. Proceedings of the Tenth Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2015). 4, 5

[Aydemir et al., 2007] Aydemir, B., Bohannon, A., and Weihrich, S. (2007). Nominal Reasoning Techniques in Coq (Extended Abstract). In *Proc. of the 1st Int. Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP)*, ENTCS, pages 69–77. 7

[Baader and Nipkow, 1998] Baader, F. and Nipkow, T. (1998). *Term rewriting and all that*. Cambridge UP. 2, 45, 53

[Barendregt et al., 1983] Barendregt, H., Coppo, M., and Dezani-Ciancaglini, M. (1983). A Filter Lambda Model and the Completeness of Type Assignment. *J. Symb. Log.*, 48(4):931–940. 6

[Calvès, 2010] Calvès, C. (2010). *Complexity and Implementation of Nominal Algorithms*. PhD thesis, King's College London. 4, 23, 42

[Calvès, 2013] Calvès, C. (2013). Unifying Nominal Unification. In *24th International Conference on Rewriting Techniques and Applications, RTA 2013*, pages 143–157. 5

[Calvès and Fernández, 2010] Calvès, C. and Fernández, M. (2010). Matching and α-equivalence check for nominal terms. *J. Comput. Syst. Sci.*, 76(5):283–301. 3, 42, 79

[Calvès and Fernández, 2010] Calvès, C. and Fernández, M. (2010). The First-Order Nominal Link. In *Logic-Based Program Synthesis and Transformation - 20th International Symposium, LOPSTR 2010, Revised Selected Papers*, pages 234–248. 5

[Cheney, 2004] Cheney, J. (2004). The Complexity of Equivariant Unification. In *Automata, Languages and Programming: 31st Int. Colloquium, ICALP 2004*, pages 332–344. 42

[Cheney, 2005] Cheney, J. (2005). Relating Nominal and Higher-Order Pattern Unification. In *Proceedings of the 19th International Workshop on Unification (UNIF 2005)*, pages 104–119. 4

[Cheney, 2009] Cheney, J. (2009). A Simple Nominal Type Theory. *Electr. Notes Theor. Comput. Sci.*, 228:37–52. 10

[Cheney, 2012] Cheney, J. (2012). A Dependent Nominal Type Theory. *Logical Methods in Computer Science*, 8(1). 10

[Cheney and Urban, 2004] Cheney, J. and Urban, C. (2004). α-Prolog: A Logic Programming Language with Names, Binding and α-Equivalence. In *Logic Programming, 20th International Conference, ICLP 2004, Saint-Malo, France, September 6-10, 2004, Proceedings*, pages 269–283. 8

[Clouston, 2007] Clouston, R. A. (2007). Closed terms (unpublished notes). Available from `http://cs.au.dk/~ranald/closedterms.pdf`. 43

[Constable and Moczydlowski, 2009] Constable, R. and Moczydlowski, W. (2009). Extracting the resolution algorithm from a completeness proof for the propositional calculus. *Annals of Pure and Applied Logic*, 161(3):337–348. 80

[Copello et al., 2016] Copello, E., Tasistro, A., Szasz, N., Bove, A., and Fernández, M. (2016). Alpha-Structural Induction and Recursion for the Lambda Calculus in Constructive Type Theory. *Electr. Notes Theor. Comput. Sci.*, 323:109–124. 7

[Coppo and Dezani-Ciancaglini, 1978] Coppo, M. and Dezani-Ciancaglini, M. (1978). A New Type Assignment for λ-terms. *Archiv für mathematische Logik und Grundlagenforschung*, 19(1):139–156. 6

[Coppo et al., 1981] Coppo, M., Dezani-Ciancaglini, M., and Venneri, B. (1981). Functional Characters of Solvable Terms. *Mathematical Logic Quarterly*, 27(26):45–58. 6

[Domínguez and Fernández, 2014] Domínguez, J. and Fernández, M. (2014). Relating nominal and higher-order rewriting. In *Mathematical Foundations of Computer Science 2014 - 39th Int. Symposium, MFCS 2014. Proc., Part I*, volume 8634 of *LNCS*, pages 244–255. Springer. 42, 43, 54

[Dowek et al., 2000] Dowek, G., Hardin, T., and Kirchner, C. (2000). Higher-order Unification via Explicit Substitutions. *Information and Computation*, 157(1/2):183–235. 2

[Fairweather, 2014] Fairweather, E. (2014). *Type Systems for Nominal Terms*. PhD thesis, King's College London. 9, 10, 11, 71, 74, 80

[Fairweather and Fernández, 2016] Fairweather, E. and Fernández, M. (2016). Typed Nominal Rewriting. Submitted. Available from http://www.inf.kcl.ac.uk/staff/maribel/papers.html. 7, 10, 74, 76

[Fairweather et al., 2011] Fairweather, E., Fernández, M., and Gabbay, M. J. (2011). Principal Types for Nominal Theories. In *FCT*, pages 160–172. 10

[Fairweather et al., 2015] Fairweather, E., Fernández, M., Szasz, N., and Tasistro, A. (2015). Dependent Types for Nominal Terms with Atom Substitutions. In *13th International Conference on Typed Lambda Calculi and Applications, TLCA 2015, July 1-3, 2015, Warsaw, Poland*, pages 180–195. 10

[Fernández and Gabbay, 2010] Fernández, M. and Gabbay, M. (2010). Closed Nominal Rewriting and Efficiently Computable Nominal Algebra Equality. In *Proc. 5th Int. Workshop on Logical Frameworks and Meta-languages: Theory and Practice, LFMTP 2010*, pages 37–51. 2, 40, 41, 44, 48, 49

[Fernández and Gabbay, 2006] Fernández, M. and Gabbay, M. J. (2006). Curry-Style Types for Nominal Terms. In *Types for Proofs and Programs, International Workshop, TYPES 2006, Nottingham, UK, April 18-21, 2006, Revised Selected Papers*, pages 125–139. 9

[Fernández and Gabbay, 2007] Fernández, M. and Gabbay, M. J. (2007). Nominal rewriting. *Information and Computation*, 205(6):917–965. 1, 3, 5, 8, 13, 28, 29, 30, 38, 41, 42, 43, 44, 45, 46, 47, 48, 52, 53, 54, 70, 79

[Fernández and Rubio, 2012] Fernández, M. and Rubio, A. (2012). Nominal Completion for Rewrite Systems with Binders. In *Automata, Languages, and Programming*, volume 7392 of *LNCS*, pages 201–213. Springer. 80

[Gabbay and Pitts, 1999] Gabbay, M. and Pitts, A. M. (1999). A New Approach to Abstract Syntax Involving Binders. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 214–224. 1

[Galdino and Ayala-Rincón, 2009] Galdino, A. L. and Ayala-Rincón, M. (2009). A PVS Theory for Term Rewriting Systems. *Electr. Notes Theor. Comput. Sci.*, 247:67–83. 17, 23

[Galdino and Ayala-Rincón, 2010] Galdino, A. L. and Ayala-Rincón, M. (2010). A Formalization of the Knuth-Bendix(-Huet) Critical Pair Theorem. *J. Autom. Reasoning*, 45(3):301–325. 23, 80

[Hindley, 2002] Hindley, J. R. (2002). *Basic Simple Type Theory*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press. 6

[Huet, 1975] Huet, G. P. (1975). A Unification Algorithm for Typed $\vec{\lambda}$-Calculus. *Theoretical Computer Science*, 1:27–57. 2

[Huet, 1980] Huet, G. P. (1980). Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems: Abstract Properties and Applications to Term Rewriting Systems. *J. of the ACM*, 27(4):797–821. 2

[Kamareddine and Nour, 2007] Kamareddine, F. and Nour, K. (2007). A Completeness Result for a Realisability Semantics for an Intersection Type System. *Ann. Pure Appl. Logic*, 146(2-3):180–198. 57, 58

[Klop et al., 1993] Klop, J.-W., van Oostrom, V., and van Raamsdonk, F. (1993). Combinatory reduction systems, introduction and survey. *Theoretical Computer Science*, 121:279–308. 1

[Knuth and Bendix, 1970] Knuth, D. and Bendix, P. (1970). Simple word problems in universal algebras. In *Computational Problems in Abstract Algebra*. Pergamon Press, Oxford. 2

[Kumar and Norrish, 2010] Kumar, R. and Norrish, M. (2010). (Nominal) Unification by Recursive Descent with Triangular Substitutions. In *Interactive Theorem Proving, First International Conference, ITP 2010*, volume 6172 of *Lecture Notes in Computer Science*, pages 51–66. Springer. 8, 15

[Lengrand et al., 2004] Lengrand, S., Lescanne, P., Dougherty, D. J., Dezani-Ciancaglini, M., and van Bakel, S. (2004). Intersection Types for Explicit Substitutions. *Inf. Comput.*, 189(1):17–42. 9

[Levy and Villaret, 2010] Levy, J. and Villaret, M. (2010). An Efficient Nominal Unification Algorithm. In *Proceedings of the 21st International Conference on Rewriting Techniques and Applications, RTA 2010*, pages 209–226. 4, 5

[Levy and Villaret, 2012] Levy, J. and Villaret, M. (2012). Nominal unification from a higher-order perspective. *ACM Transactions on Computational Logic*, 13(2):10. 4

[Mayr and Nipkow, 1998] Mayr, R. and Nipkow, T. (1998). Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192:3–29. 1

[Newman, 1942] Newman, M. H. A. (1942). On Theories with a Combinatorial Definition of “Equivalence”. *The Annals of Mathematics*, 43(2):pp. 223–243. 2

[Owre and Shankar, 1997] Owre, S. and Shankar, N. (1997). Abstract Datatypes in PVS. Technical report, SRI International. http://pvs.csl.sri.com/papers/csl-93-9/csl-93-9.pdf. 21

[Owre and Shankar, 1999] Owre, S. and Shankar, N. (1999). The Formal Semantics of PVS. Technical report, SRI International. http://shemesh.larc.nasa.gov/fm/papers/Owre-CR-1999-209321-Semantics-PVS.pdf. 16, 19

[Paulson, 1985] Paulson, L. C. (1985). Verifying the Unification Algorithm in LCF. *Science of Computer Programming*, 5(2):143–169. 23

[Pitts, 2003] Pitts, A. M. (2003). Nominal Logic, a First Order Theory of Names and Binding. *Inf. Comput.*, 186(2):165–193. 10

[Pitts et al., 2015] Pitts, A. M., Matthiesen, J., and Derikx, J. (2015). A Dependent Type Theory with Abstractable Names. *Electr. Notes Theor. Comput. Sci.*, 312:19–50. 10

[Pottier, 2006] Pottier, F. (2006). An Overview of Cαml. *Electr. Notes Theor. Comput. Sci.*, 148(2):27–52. 8

[Rocha-Oliveira and Ayala-Rincón, 2012] Rocha-Oliveira, A. C. and Ayala-Rincón, M. (2012). Formalizing the confluence of orthogonal rewriting systems. In *Proc. 7th Workshop on Logical and Semantic Frameworks, with Applications, LSFA 2012*, pages 145–152. 53

[Rocha-Oliveira et al., 2016] Rocha-Oliveira, A. C., Galdino, A. L., and Ayala-Rincón, M. (2016). Confluence of Orthogonal Term Rewriting Systems in the Prototype Verification System. *Journal of Automated Reasoning*, pages 1–21. 2, 17, 80

[Rosen, 1973] Rosen, B. K. (1973). Tree-Manipulating Systems and Church-Rosser Theorems. *J. of the ACM*, 20(1):160–187. 2

[Russell, 1908] Russell, B. (1908). Mathematical Logic as Based on the Theory of Types. *American Journal of Mathematics*, 30(3):222–262. 5

[Schmidt-Schauss et al., 2016] Schmidt-Schauss, M., Kutsia, T., Levy, J., and Villaret, M. (2016). Nominal Unification of Higher Order Expressions with Recursive Let. In *Pre-proceedings of the 26th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2016)*. 4

[Shankar et al., 2001] Shankar, N., Owre, S., Rushby, J. M., and Stringer-Calvert, D. W. J. (2001). PVS Prover Guide. Technical report, SRI International. http://pvs.csl.sri.com/doc/pvs-prover-guide.pdf. 16, 23

[Shinwell et al., 2003] Shinwell, M. R., Pitts, A. M., and Gabbay, M. J. (2003). Freshml: programming with binders made simple. *SIGPLAN Notices*, 38(9):263–274. 8

[Stehr, 2000] Stehr, M.-O. (2000). `CINNI` - A Generic Calculus of Explicit Substitutions and its Application to λ- ς- and π-Calculi. *Electronic Notes in Theoretical Computer Science*, 36:70–92. The 3rd Int. Workshop on Rewriting Logic and its Applications. 1

[Suzuki et al., 2015] Suzuki, T., Kikuchi, K., Aoto, T., and Toyama, Y. (2015). Confluence of Orthogonal Nominal Rewriting Systems Revisited. In *26th Int. Conf. on Rewriting Techniques and Applications (RTA 2015)*, volume 36 of *LIPIcs*, pages 301–317. 3, 8, 46, 47, 48, 54, 79

[Suzuki et al., 2016] Suzuki, T., Kikuchi, K., Aoto, T., and Toyama, Y. (2016). Critical Pair Analysis in Nominal Rewriting. In *7th International Symposium on Symbolic Computation in Software Science, SCSS 2016, Tokyo, Japan, March 28-31, 2016*, pages 156–168. 3, 8, 9, 79

[Thiemann, 2013] Thiemann, R. (2013). Formalizing bounded increase. In *Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings*, pages 245–260. 2

[Urban, 2004] Urban, C. (2004). Nominal Unification. http://www.inf.kcl.ac.uk/staff/urbanc/Unification. 5, 8, 10, 27, 29, 34, 79

[Urban, 2008] Urban, C. (2008). Nominal Techniques in Isabelle/HOL. *J. Autom. Reasoning*, 40(4):327–356. 7

[Urban, 2010] Urban, C. (2010). Nominal Unification Revisited. In *Proceedings 24th International Workshop on Unification, UNIF 2010*, volume 42 of *Electronic Proceedings in Theoretical Computer Science*, pages 1–11. 5, 8, 10, 27, 29, 30, 31, 78

[Urban et al., 2004] Urban, C., Pitts, A. M., and Gabbay, M. (2004). Nominal Unification. *Theoretical Computer Science*, 323(1-3):473–497. 3, 4, 5, 8, 10, 13, 29, 30, 31, 32, 42, 78

[van Bakel, 1992] van Bakel, S. (1992). Complete restrictions of the intersection type discipline. *Theor. Comput. Sci.*, 102(1):135–163. 6, 56

[van Bakel, 1993] van Bakel, S. (1993). Essential intersection type assignment. In *Foundations of Software Technology and Theoretical Computer Science, 13th Conference, Bombay, India, December 15-17, 1993, Proceedings*, pages 13–23. 79

[van Bakel, 1995] van Bakel, S. (1995). Intersection type assignment systems. *Theor. Comput. Sci.*, 151(2):385–435. 6, 9, 10, 56

[van Bakel, 2011] van Bakel, S. (2011). Strict intersection types for the lambda calculus. *ACM Comput. Surv.*, 43(3):20. 60

[van Bakel et al., 1996] van Bakel, S., Barbanera, F., and Fernández, M. (1996). Rewrite Systems with Abstraction and beta-Rule: Types, Approximants and Normalization. In *Programming Languages and Systems - ESOP'96, 6th European Symposium on Programming, Linköping, Sweden, April 22-24, 1996, Proceedings*, pages 387–403. 81

[van Bakel and Fernández, 1997] van Bakel, S. and Fernández, M. (1997). Normalization results for typeable rewrite systems. *Inf. Comput.*, 133(2):73–116. 9, 10, 56, 57, 58, 62, 65, 80

[Ventura et al., 2015] Ventura, D. L., Kamareddine, F., and Ayala-Rincón, M. (2015). Explicit Substitution Calculi with de Bruijn Indices and Intersection Type Systems. *Logic Journal of the IGPL*, 23(2):295–340. 9