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

Nominal Equational Problems Modulo Associativity, Commutativity and Associativity-Commutativity

Washington Luís Ribeiro de Carvalho Segundo

Tese apresentada como requisito parcial para
conclusão do Doutorado em Informática

Orientador
Prof. Dr. Mauricio Ayala-Rincón

Coorientadora
Prof. Dr. Maribel Fernández

Brasília
2019

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

Nominal Equational Problems Modulo Associativity, Commutativity and Associativity-Commutativity

Washington Luís Ribeiro de Carvalho Segundo

Tese apresentada como requisito parcial para
conclusão do Doutorado em Informática

Prof. Dr. Mauricio Ayala-Rincón (Orientador)
Universidade de Brasília

Prof. Dr. Maribel Fernández (Coorientadora)
King's College London

Prof. Dr. Temur Kutsia
Johannes Kepler University Linz

Prof. Dr. Alejandro Díaz-Caro
Universidad Nacional de Quilmes
& ICC (CONICET-UBA)

Prof. Dr. Daniel Lima Ventura
Universidade Federal de Goiás

Prof. Dr. Bruno Luiggi Macchiavello Espinoza
Coordenador do Programa de Pós-graduação em Informática

Brasília, 20 de fevereiro de 2019

Dedicatória

Dedico este trabalho à minha família nuclear: Iraciara Almeida e José Lucas, que foram as pessoas que mais sofreram com minhas ausências e com meu mal humor no momentos em que cumprir com os *deadlines*. Dedico-o também em memória de minha mãe, Denise Gonçales Ribeiro de Carvalho, e de meu tio Ricardo José Gonçales. Minha mãe partiu logo no início desta jornada de 5 anos, e meu tio partiu de forma mais recente. Ambos fazem muita falta nos dias de hoje, e sempre farão.

Agradecimentos

Em primeiro lugar ao meu orientador, Professor Mauricio Ayala-Rincon, à minha coorientadora, Professora Maribel Fernández e à minha coorientadora informal, Professora Daniele Nantes-Sobrinho. Sem o apoio e orientação fornecidos por vocês, este trabalho, de fato, não existiria. Reconheço também o meu orientador de mestrado, Professor Flávio Leonardo de Moura, por ter me reinserido no ambiente acadêmico, após a pausa de três anos. Agradeço aos membros da banca: Professor Temur Kutsia; Professor Alejandro Díaz-Caro; e Professor Daniel Lima-Ventura pelas importantes contribuições e sugestões de correção do texto da tese.

À minha família, cujo apoio e compreensão foi fundamental para execução deste trabalho: Iraciara A. de Souza; José Lucas A. Ribeiro; Washington R. de Carvalho; Denise G. de Carvalho; Derly Gonçales; Manoel Gonçales Neto; M. Cristina Gonçales; Beatriz M. Gonçales; Ricardo J. Gonçales; Maria do Socorro de Almeida; Adilson de Souza; Adilson de Souza Júnior; Kelly Cristina; Iara de Souza, Gilberto Machado; Indiara de Souza; M. Luiza Gomes; Armando Nunes; Antônia Almeida; e Guilherme Almeida.

Aos amigos de trabalho por terem compreendido a importância deste meu trabalho na minha formação e me dado suporte em minhas ausências: Dra. Cecília Leite; Dra. Bianca Amaro; Tainá Assis; Lautaro Mattas; Clediomir Silva; Leonard Richard e toda a equipe de bolsistas que trabalhou comigo nestes anos.

Por fim, agradeço aos amigos: Bruno Mazzo e Juliano Mazzo (que sempre executaram o papel dos irmãos que eu nunca tive); Gabriel F. Silva e Ariane Alves (cujo apoio dado no dia da defesa foi fundamental); Alison R. Panisson (por ter sido um companheiro e amigo na fria Londres); Ana C. Oliveira (por ter compartilhado o trabalho e objetivos com relação à Lógica Nominal); Lucas Silveira; Daniel Saad; Thiago F. Ramos; Bruno Delboni; Andréia B. Avelar; Thaynara de Lima; Mehwish Arshid; e Kaliana D. de Freitas que foram os companheiros de laboratório, como também às amizades conquistadas durante esta jornada.

Com toda a certeza muitos nomes além destes deveriam constar aqui. Pela ausência peço perdão, e gostaria de dizer que todos foram fundamentais para que eu alcançasse o meu objetivo. A todos vocês eu gostaria de dizer:

***U B U N T U* = “Eu sou porque vocês são.”**

Acknowledgements

In the first place to my supervisor, Professor Mauricio Ayala-Rincón, my co-supervisor, Professor Maribel Fernández and my informal co-supervisor, Professor Daniele Nantes-Sobrinho. Without your support and supervision, this work, in fact, would not exist. I acknowledge also my supervisor of the master course, Professor Flávio Leonardo de Moura, for having reintegrated me to the academic environment after the break of three years. I acknowledge the referees: Professor Temur Kutsia; Professor Alejandro Díaz-Caro; and Professor Daniel Lima-Ventura for the important contributions and the suggestions of correction in the thesis text.

To my family, whose support and comprehension was fundamental to the execution of this work: Iraciara A. de Souza; José Lucas A. Ribeiro; Washington R. de Carvalho; Denise G. de Carvalho; Derly Gonçales; Manoel Gonçales Neto; M. Cristina Gonçales; Beatriz M. Gonçales; Ricardo J. Gonçales; Maria do Socorro de Almeida; Adilson de Souza; Adilson de Souza Júnior; Kelly Cristina; Iara de Souza, Gilberto Machado; Indiara de Souza; M. Luiza Gomes; Armando Nunes; Antônia Almeida; and Guilherme Almeida.

To the work friends for having acknowledged the relevance of my work in my formation and having provided me support in my absences: Dra. Cecília Leite; Dra. Bianca Amaro; Tainá Assis; Lautaro Mattas; Clediomir Silva; Leonard Richard and all the research fellows that worked with me in these years.

Finally, I thank the friends: Bruno Mazzo and Juliano Mazzo (that always played the role of the brothers that I never had); Gabriel F. Silva and Ariane Alves (whose support were fundamental in the day of the viva); Alison R. Panisson (for being a partner and a friend in the cold London); Ana C. Oliveira (for sharing her work and goals regarding Nominal Logic); Lucas Silveira; Daniel Saad; Thiago F. Ramos; Bruno Delboni; Adréia B. Avelar; Thaynara de Lima; Mehwish Arshid; and Kaliana D. de Freitas that were the lab partners and the conquered friends during this journey.

I am pretty sure that, beyond these, plenty of names should be here. For these absences, I ask for your pardon, and I would like to say that all of you were fundamental for me to achieve my goal. I would like to say to all of you:

_U B U N T U_ = "I am because you are."

“The world is my country, science is my religion.”
— Christiaan Huygens

Resumo

A sintaxe nominal tem sido utilizada em vários contextos por quase duas décadas. Ela é uma ferramenta poderosa para se lidar com ligação de variáveis de uma forma concreta, que pode ser aplicada a qualquer especificação na qual parâmetros são utilizados para se abstrair variáveis, tal como em predicados e funções. Na sintaxe nominal, objetos que são sintaticamente diferentes podem ter a mesma semântica módulo alfa-conversão, tal como acontece no Cálculo Lambda. O tratamento de igualdades, em especial a alpha-equivalêcia, é algo essencial em linguagens formais e implementações. Este trabalho investiga a alpha-equivalência nominal com símbolos de função associativos (A), comutativos (C) e associativos-comutativos (AC). Verificação de equivalência, casamento e unificação módulo A, C e AC são investigados. Em relação a verificação de igualdade, as alpha-equivalências nominais módulo A, C e AC foram especificadas em Coq e provadas ser corretas. Um algoritmo implementado em OCaml para verificação de igualdade módulo A, C e AC é automaticamente extraído da especificação e experimentos são executados utilizando-se também um algoritmo aperfeiçoado. Limites superiores para o tempo de execução na solução de problemas nominais de verificação equacional são fornecidos. Um algoritmo de unificação módulo C baseado em regras de redução é especificado em Coq e provado ser correto e completo. Por meio do uso de variáveis protegidas, este algoritmo de unificação resolve problemas de casamento nominal módulo C, o que foi também formalizado ser correto e completo. O algoritmo de unificação baseado em regras de redução fornece uma família finita de conjuntos de equações nominais de ponto fixo. Cada uma destas equações pode ter um conjunto infinito de soluções independentes. Portanto, demonstra-se que problemas de unificação nominal módulo C e AC podem gerar um conjunto infinito de soluções independentes. Este fato contrasta com unificação sintática módulo C ou AC, que são conhecidas por estar na classe finitária de problemas. Uma implementação em OCaml do algoritmo de unificação nominal é fornecida e utilizado para se construir exemplos.

Palavras-chave: Lógica nominal, Alpha-equivalência, Unificação de primeira-ordem, Unificação nominal, Unificação módulo teorias equacionais, Equações de ponto fixo.

Abstract

The nominal syntax has been used in many application contexts for almost two decades. It is a powerful tool for dealing with variable binding in a concrete manner that can be applied to any specification in which parameters are used to abstract variables, such as in predicates and functions. In the nominal syntax, syntactically different objects can have the same semantics modulo alpha-conversion, as happens in the lambda calculus. Dealing with equality, and in special with alpha-equivalence, is essential in formal languages and implementations. This work investigates the nominal alpha-equivalence with associative (A), commutative (C) and associative-comutative (AC) function symbols. Equality-checking, matching and unification modulo A, C and AC are investigated. Regarding equality-checking, nominal alpha-equivalence modulo A, C and AC are specified in Coq and proved sound. An algorithm implemented in OCaml for equality-checking modulo A, C and AC is automatically extracted from the specification and experiments are performed using also an improved algorithm. Upper bounds for solving nominal equality-checking problems are given. A rule-based nominal unification modulo C algorithm is specified in Coq and proved sound and complete. By using protected variables, this unification algorithm solves nominal matching problems modulo C, which is formalised to be sound and complete. The rule-based nominal unification algorithm outputs a finite family of sets of fixed point nominal equations. Each of which might have an infinite set of independent solutions. Therefore, nominal unification modulo C or AC are proved to potentially generate infinite independent solutions. This contrasts with syntactic unification modulo C or AC that are known to be in the finitary class. An OCaml implementation of the nominal unification algorithm is provided and used to build examples.

Keywords: Nominal logic, Alpha-equivalence, First-order unification,
Nominal unification, Unification modulo equational theories, Fixed point equations.

Contents

1 Introduction **1**
1.1 Related work . . . 3
1.2 Contribution summary . . . 7
1.3 Organisation . . . 8

2 Background **10**
2.1 First-order unification . . . 10
2.1.1 First-order syntactic unification . . . 13
2.1.2 First-order A, C and AC-unification . . . 16
2.2 Nominal syntax . . . 22
2.2.1 Freshness and the nominal α-equivalence . . . 24
2.2.2 A rule-based nominal unification algorithm . . . 26

3 Specification and Formalisation in Coq: the case of nominal α equality-checking **32**
3.1 Soundness of $\approx_\alpha$ using a *weak* α-equivalence $\sim_\omega$ . . . 36
3.2 Soundness of $\approx_\alpha$ without using $\sim_\omega$ . . . 40
3.3 Comparing the two formalisation approaches . . . 42

4 Nominal α, A, C and AC equality-checking **44**
4.1 Operations over tuples . . . 45
4.2 Extension of the rules for $\approx_\alpha$ . . . 48
4.3 Formalisation of the soundness of $\approx_{\{A,C,AC\}}$ . . . 51
4.4 A naive implementation of the $\approx_{\{A,C,AC\}}$ equality-checking algorithm . . . 55
4.5 Automatic code extraction . . . 58
4.6 Execution tests . . . 62
4.7 Upper bounds . . . 64

5 Nominal C-unification and matching **72**
5.1 Formalisation of nominal C-unification with protected variables . . . 72
5.1.1 Basic formalised notions and results on nominal C-unification . . . 73
5.1.2 Main formalised results for C-unification . . . 78
5.2 Nominal C-matching . . . 93
5.2.1 Basic notions on nominal C-matching and auxiliary formalised properties 93
5.2.2 Main formalised results for nominal C-matching . . . 96

6 Nominal fixed point problems **107**
6.1 Generating combinatorial solutions via *pseudo-cycles* . . . 107
6.2 General solutions for C FP problems. . . . 116
6.2.1 Soundness and completeness of the generator . . . 119
6.2.2 Improvements in the generation of solutions . . . 129

7 Nominal A, C and AC-unification and matching **133**
7.1 Rules for nominal A, C and AC problems . . . 133
7.2 Solutions for nominal AC FP problems . . . 141

8 Conclusion and future work **148**

References **150**

Appendices **154**

A **155**

Chapter 1

Introduction

Equational problems emerge in many application contexts. In fact, operations such as pattern recognition and simplification are completely linked with these kinds of problems. For instance, in optimisation, the partial solutions obtained during the computation steps can be interpreted as simpler *equivalent* versions of the original problem. An example of the use of pattern recognition occurs in searching engines. In this case, a pattern expression is given to be *matched* inside a text. Moreover, in functional programming, the inference of a principal type of an expression uses a general operation with patterns denominated *unification*.

Formally, an equational problem is defined over a set of *terms* and an equivalence relation $\approx$. In *first-order logic*, terms are build, inductively, in the following manner: a constant c is a term; a variable X is a term; an application of a function symbol to a tuple of terms $f(t_0, \ldots, t_n)$ is also a term, where f has arity $n+1$. Then, a substitution is defined as a mapping with a finite domain from variables to terms $\{X_0/t_0, \ldots, X_k/t_k\}$. A substitution σ acts recursively over a term. For instance, if $\sigma = \{X/W, Y/Z\}$ then $f(X, Y)\sigma = f(W, Z)$.

From the previous definitions it is possible to formally establish the main classes of equational problems, *equality-checking*, *matching* and *unification*. Given a pair of terms (s, t) and an equivalence relation $\approx$, these classes are defined, respectively, by the following three questions: i) is s related to t by $\approx$? ii) what is the set $\{\sigma \mid s\sigma \approx t\}$? iii) what is the set $\{\sigma \mid s\sigma \approx t\sigma\}$? In the case of the two sets, the found σ's are denominated *solutions* for the matching (resp. unification) problem $s\sigma \approx_? t$ (resp. $s\sigma \approx_? t\sigma$). It is interesting to know if these sets are *decidable*, what is their *cardinality*, and in which class of *complexity* the problems of finding solutions are.

The relation $\approx$ may contain equational theories over the function symbols of the syntax. For instance, given a commutative (C) binary function symbol f, for any terms s and t, it is true that $f(s, t) \approx f(t, s)$. The same holds for other possible equational properties, such

as associativity (A), nilpotency (N), etc. Observe that function symbols with equational properties have a very frequent use in algebra and logic. For instance, monoids (resp. abelian groups) are defined with a binary A (resp. AC) operator with neutral element (U). In symmetric groups, N is a property of the operator of composition of elements and in the classical logic, the symbols $\vee$ and $\wedge$ are, respectively, interpreted as the AC operators, *disjunction* and *conjunction*, that have as neutral element, respectively, the constants $false$ and $true$.

The set of function symbols available to build terms is called their *signature* and it is represented by the letter Σ. Then, equational problems where Σ contains function symbols with equational theories are denominated *equality-checking (matching or unification) modulo*. According to the functions symbols in Σ, equational problems may be in distinct classes of decidability, cardinality and complexity. Supposing that Σ contains only *syntactic function symbols*, those symbols that do not have equational properties, then any equality-checking, matching or unification problem is decidable, has a unitary set of solutions and is linearly solved. On the other hand, if Σ contains at least one C function symbol, equality-checking is quadratic, general matching and unification are NP-complete (See Rmk. 2.5 of Subsection 2.1.2) and all this kind of equational problems are decidable and have a finite set of solutions, but this set may have more than one element.

Standard first-order terms do not express a relevant aspect in computation denominated *binding*. This can be exemplified as the action of specifying parameters in a definition of a function. For instance, in $f : a \mapsto a^2$ the name a is bound and therefore any renaming of a, say a by b, results in a syntactically different term $f : b \mapsto b^2$ that has the same *semantics* of the first. First-order syntax was generalised to include a new constructor called *abstraction*. This new object is represented by $[a]t$ and expresses the *binding* of a in t. In the last example, the term $f : a \mapsto a^2$ would be represented by $f : [a]a^2$, which is related to the term $f : [b]b^2$ by an equivalence relation $\approx_\alpha$.

This modification was denominated *nominal syntax* and allowed to define in a concrete manner the relation $\approx_\alpha$. It also changes the representation of variables. Variables X are decorated with *permutations* π that are represented by a possibly empty list of *renamings* $(a_0\, b_0) :: \ldots :: (a_n\, b_n) :: nil$. Equational problems in the nominal syntax are decidable, polynomially bounded and have unitary sets of solutions. As expected, such properties may change when Σ has function symbols with equational properties.

The present work explores algorithms for solving equational problems in the nominal syntax where Σ contains A, C and AC function symbols. Also, implementations and a formalisation of the properties of these algorithms are provided.

1.1 Related work

Reasoning over first-order terms has been investigated since the middle sixties. Robinson [60] proposed his syntactic first-order unification algorithm as an underlying tool for the *resolution principle* in theorem-proving. This unification algorithm was proved sound and complete, and the problem of first-order syntactic unification was proved unitary. Efficient versions of this algorithm appear only in the late seventies, when, independently, Martelli and Montanari [51], and Paterson and Wegman [56] proposed algorithms that were linearly bounded, in time and space. The former is based on smart reasoning strategies, while the latter used a different representation for terms by acyclic digraphs, which was called *term graph* representation.

Later, first-order reasoning was extended to signatures with function symbols that have equational properties such as A, C, distributivity (D), idempotence (I), existence of neutral element (U) among others (see the E-unification survey by Baader and Snyder [15]). For instance, in [64], Siekmann explored first-order unification with C function symbols, proposing a sound and complete algorithm and proving that first-order C-unification is decidable and finitary. First-order C-unification is a NP-complete problem [44].

First-order AC-unification was first investigated by Stickel in [67]. This work presented the translation of an AC-unification problem to the problem of solving a set of Diophantine equations. Proofs of soundness and completeness of this algorithm were given by Fages [38]. Later improved versions (in efficiency) of AC-unification appear in a series of works, for instance in [19, 29]. Additionally, complexities of first-order equational problems were explored by Kapur and Narendran in [43, 44, 45], and by Benanav, Kapur and Narendran in [17]. Eker [36, 37] investigated and implemented AC-matching algorithms via the translation AC-matching problems to hierarchies of bipartite graph matching problems.

The formal treatment of α-equivalence was the main objective of a creation of a new theoretical framework denominated Nominal Logic [42, 57, 58]. This framework is based on two key concepts, *freshness* and *name swapping*, that implement the notion of binding in a concrete manner. A name is *fresh* for a term if it does not occur unbounded, and abstractions $[a]s$ and $[b]t$ are α-equivalent if a is fresh for t and s is α-equivalent to the name swapping of a by b in t (which is denoted by $(a\,b) \cdot t$). In the nominal approach, freshness and α-equivalence take into account an extra parameter named *freshness context*. This object is a set of pairs $a\#X$ with the semantics that a must be fresh in a possible instance of X. This theory was a basis of new developments in the area of equational reasoning, matching and unification.

Nominal unification, that is unification in the nominal syntax, appears only in 2004, in the work of Urban, Pitts and Gabbay [72]. The authors presented a rule-based algorithm that was composed by sets of transformation rules. The simplification rules operate over

two types of constraints, equations $s \approx_? t$ and *freshness constraints* $a \#_? t$. The latter express the question: under which freshness context, can a be fresh for t? One set of rules is used to simplify freshness constraints and the other is applied to equations. The same strategy is used in the unification algorithms that are explored in Chapter 5. In [72], this nominal unification algorithm has been showed sound and complete, and, as in first-order syntactic unification, it provides unitary solutions.

Additionally, Levy and Villaret [48, 49] showed that there exists a quadratic reduction from a nominal to a higher-order patterns unification problem [53, 55, 73]; and Cheney [27] proved the opposite way: a higher-order patterns unification problem can be reduced to a nominal unification problem. Independently, Levy and Villaret [48], and Calvès and Fernández [25] demonstrated that nominal unification is quadratically bounded. The latter extends the Paterson and Wegman syntactic first-order unification algorithm [56], with the definition of a morphism from first-order to the nominal syntax, keeping the Paterson-Wegman term graph representation. Some implementations of nominal unification were used in logic programming languages such as αKanren [22] and αProlog [26].

An algorithm for solving nominal matching problems was first studied by Fernández and Gabbay [41] in which context was called *nominal rewriting*. In this work, the authors defined a nominal matching algorithm as a special case of nominal unification, considering the right-hand sides of equations as ground terms. Then, α-equality-checking and nominal matching were more deeply explored by Càlves and Fernández [24, 23]. To improve efficiency of the proposed algorithms, the representation of nominal terms was changed. A new concept denominated *environment* synthesised in just one expression the atoms that need to be fresh and the permutation that must be applied to the considered term. From this new representation, sets of simplification rules were established to operate over α equality-checking and nominal matching problems, postponing the propagation of operations of checking freshness and the action of permutations. The latter strategy was denominated *lazy permutations*. Upper bounds for the execution were obtained. The execution of α equality-checking is linear for ground and log-linear for non-ground terms, while nominal matching is log-linear for linear problems (where each variable occurs just once) and quadratic for non-linear problems.

Cheney [28] proposed a generalisation of nominal unification and nominal matching that removes the freshness restriction in the application of atom-renamings, which was denominated *equivariant unification* and *equivariant matching*. These generalisations were applied to confluence analysis of nominal rewriting systems [1, 41] and also to the construction of nominal anti-unification algorithms [16]. In addition, Ayala-Rincón et al [8] provided an analysis over closed nominal rewriting systems, that avoids nominal equivariant algorithms and uses only standard nominal matching.

Type systems were also investigated in the nominal syntax. Fernández and Gabbay [40] defined a curry-style polymorphic type system for nominal terms, while Fairweather et al [39] proposed dependent types in the nominal syntax. Moreover, Fairweather and Fernández investigated types on nominal rewriting, and finally Ayala-Rincón et al [12] developed intersection types for nominal terms.

Formalisations of first-order syntactic unification can be found, for example, in Avelar et al. [2], and Brandt, Schlichtkrull and Villadsen [21], which were performed, respectively, in the proof assistants PVS and Isabelle/HOL. Also, the most well-known formalisations of nominal unification were done in the proofs assistants Isabelle/HOL, HOL4, PVS and Coq, respectively, by Urban, Pitts and Gabbay [72, 70], Kumar and Norrish [46], Ayala-Rincón, Fernández and Rocha-Oliveira [11] (also published in [61]), and Ayala-Rincón et al. [3]. These syntactic and nominal formalisations can be grouped in two classes: the functional recursive and the inductive ones. In the first, the unification algorithms are defined recursively, and in the second they are specified through relations that are defined inductively. From a pragmatic point of view, recursive and inductive definitions are equally feasible in proof assistants, but the key difference is that the recursive ones are closer to algorithmic implementations, while the inductive are easier to manage in proofs, due to the induction schemes that are generated automatically by the proof assistants.

In the formalisations of [46] and [70], the proof of soundness of $\approx_\alpha$ uses an auxiliary *weak* α-equivalence, denoted as $\sim_\omega$. Essentially, $\sim_\omega$ is defined as $\approx_\alpha$ with restrictions. One of these restrictions is that it only relates abstractions with the same atom, i.e., $[a]s \sim_\omega [a]t$ if $s \sim_\omega t$, but, if $a \neq b$, $[a]s \sim_\omega [b]t$ is not derivable. In opposition of this approach, the formalisation of the soundness of $\approx_\alpha$ proposed by [11] followed the strategy of [41], and exposed that the formalisation is simplified if one does not use the auxiliary $\sim_\omega$ in the proofs. Chapter 3 provides a detailed comparison between these two formalisation approaches (with and without the use of $\sim_\omega$).

Specially, formalisations of [46] and [11] differ from [72] and [3] in other aspects. The formalisation [46] uses a different approach to build the solutions, that are obtained via construction of a *triangular substitution*. This is a set of singleton bindings for different variables, that are accumulated in the recursive calls, during the execution of the algorithm. In [11], the specification does not include freshness constraints. A simple recursive function generates the minimal freshness context ∇ for a pair $\langle a, t \rangle$, where a is fresh for t under the context ∇. This function is called from the unification algorithm to build a solution for the unification problem.

Beyond unification, following the nominal approach, principles of induction and recursion modulo α-equivalence were formalised in Isabelle/HOL and Coq, and implemented in Agda. These developments were performed, respectively, by Urban [69] (and extended by

Urban and Kaliszyk [71]), Aydemir, Bohannon and Weirich [13], and Copello et al. [31]. In the first and second, specifications diverge from the nominal syntax. The reason is that, in the first, terms are defined using indices to represent bound variables, and, in the second, although the presence of a general treatment of binding scopes, a higher-order argument is used to express bound object-level variables. On the other hand, the Agda implementation is based essentially on nominal swapping and freshness, without using indexes or higher-order expressions.

Formalisations of equational reasoning modulo A, C and AC are available: Nipkow [54] proposed a set of rules that implement rewriting tactics in Isabelle/HOL to reason modulo A, C and AC. This set was used to build equational matching and unification algorithms, but aspects of performance and termination of these algorithms were not explored. Contejean [30] developed a sound and complete A, C and AC-matching algorithm that was defined as a set of rewriting rules that decompose equations until *solved* normal forms are reached. This algorithm was formalised in Coq and implemented in C*i*ME, but efficiency and complexity analysis were not provided. Additionally, Braibrant and Pous [20] designed a plugin for Coq to use the tactic `rewrite` modulo A and AC. The development of tactics `aac_rewrite`, which uses matching modulo A and AC, and `aac_reflexivity` which uses equality-checking modulo A and AC, was based on the `Morphisms` library and an auxiliary OCaml program with implementations of the equality-checking and matching algorithms. Proofs of soundness of the algorithms were presented, but, again, neither complexity analysis nor performance tests were given.

Recently, Durán et al. [35] extended the Maude system implementing A-unification and narrowing algorithms. C, AC, ACU, CU and UI reasoning was already present in previous versions of the system. The pitfall in A-unification is that problems may have an infinite number of solutions (see Subsection 2.1.2). For solving this issue, the authors created a smart detection mechanism of the problematic cases, that generates a warning saying that the provided set of solutions may be incomplete.

In [5] and [6], it is provided a Coq formalisation of an extension of the standard nominal α-equivalence, by adding A, C and AC function symbols to the signature. The soundness of the standard nominal α-equivalence is used in the proof of soundness of the nominal α, A, C and AC-equivalence. Two OCaml implementations of the nominal α, A, C and AC equality-checking algorithm were presented. One implementation was automatically extracted from the specification and the other was coded by hand with an improvement in the A and AC equality-checking cases. Running tests were provided comparing the two implementations, and upper bounds were given. These results are presented in Chapter 4.

Regarding extensions of nominal equational reasoning, nominal narrowing was introduced by Ayala-Rincón, Fernández and Nantes-Sobrinho in [9]. This work adapts Hullot's

seminal work on narrowing, originally developed from the first-order rewriting perspective, to the nominal approach in such a manner that nominal equational unification problems are solvable by narrowing whenever the equational theories can be presented as a class of convergent closed rewriting systems. Another extension of nominal unification was proposed in Schmidt-Schauss et al. [63]. This development proposed an algorithm to solve nominal unification problems with *recursive let* operators. In this algorithm, the solutions of a unification problem are expressed in terms of nominal fixed point equations.

Obtaining solutions for such equations is a recurrent problem; indeed, in [3] it has been showed that nominal C-unification problems are reduced to solving finite families of fixed point equations. This work also proved that nominal C-unification problems may have infinite independent solutions, differing from syntactic C-unification that is well-known to be finitary. Finally, sound and complete nominal C-unification/matching algorithms with protected variables were proposed in [7]. The content of [3] and [7] are described in Chapter 5 and Chapter 6, Section 6.1.

In [4], Ayala-Rincón et al. proposed a sound and complete combinatorial procedure to generate the set of solutions of nominal fixed point problems. This result is described in Chapter 6, Section 6.2. Recently, Ayala-Rincón, Fernández and Nantes-Sobrinho [10] proposed a representation of solutions of nominal C-unification problems based on *fixed-point constraints*. The authors showed that the standard representation of solutions, composed by a pair of a freshness context and a substitution, can be translated conservatively to a pair of fixed-point constraints and a substitution. Following this approach, nominal C-unification problems are finitary, as in the syntactic case.

1.2 Contribution summary

The contributions of the present work can be grouped in three subjects:

1. **Equality-checking**:
    - In Chapter 3, a Coq formalisation of the soundness of nominal α-equivalence is provided with a comparison between two formalisation approaches.
    - In Chapter 4, the nominal α-equivalence is extended, including A, C and AC function symbols in the signature. This extension is proved sound. Also, an OCaml executable code is automatically extracted from the specification and execution tests are performed. The extracted implementation is compared with an improved one. Upper bounds for nominal α, A, C and AC equality-checking are provided;
2. **Unification and matching**:

- In Chapter 5, it is presented a Coq specification of a nominal unification algorithm with C function symbols. This specification takes into account a set of protected variables $\mathcal{X}$ that can not be instantiated. Proofs of termination, soundness and completeness of this algorithm are formalised. Execution examples computed with an OCaml implementation of the nominal C-unification algorithm are also provided.
- Additionally, a nominal C-matching algorithm is obtained setting $\mathcal{X}$ with the variables that occur in right-hand side of the equations in the input problem. The properties of termination, soundness and completeness of this algorithm are available in the Coq formalisation.
- Also, in Section 7.1, an extension of the nominal C-unification/matching algorithm, with A and AC function symbols is considered. This extension adds rules to simplify equations whose terms are headed by A or AC function symbols. The extended algorithm is proved sound. Completeness is reached when either the right-hand side variables are protected or the input problem does not have A function symbols.
- These nominal C, and nominal AC-unification algorithms simplifies the input problem, transforming it into a family of *fixed point problems*.

3. **Fixed point problems**:

- Fixed point problems are proved to have infinite independent solutions, built just using basic terms and the C or the AC function symbols of the signature. Since the nominal C and the nominal AC algorithms were proved sound and they have as output a set of fixed point problems, one concludes that nominal C, and nominal AC-unification problems may have infinite independent solutions. This result contrasts with first-order C and first-order AC-unification that are finitary problems.
- Sound and complete generators of solutions for nominal fixed point problems are also presented. In the C case, a detailed combinatorial analysis over the *cycles* in the permutations of the fixed point equations is explored in Chapter 6. The generation of AC combinatorial solutions is presented in Section 7.2.

1.3 Organisation

Chapter 2 gives the necessary background. Chapter 3 presents the use of Coq as a proof assistant and provides as an example the formalisation of the soundness of the

nominal α-equivalence. Chapter 4 contains a formalisation of the soundness of the nominal α-equivalence modulo A, C and AC, also providing implementations and tests. Chapter 5 explores a formalisation of nominal unification and matching in a signature with C function symbols. Chapter 6 presents the results about generating a sound and complete set of solutions with C function symbols for nominal fixed point equations. Chapter 7 analyses nominal unification and matching, and fixed point problems with A, C and AC function symbols. Finally, Chapter 8 concludes and discusses the future work. The Coq formalisation and the source code of OCaml implementations are available at http://dx.doi.org/10.5281/zenodo.2582109. The hierarchy of the Coq formalisation is presented in Appendix A.

Chapter 2

Background

In this Chapter, a brief explanation about syntactic unification and unification modulo A, C and AC is presented. Also, unification in the nominal syntax is explored.

2.1 First-order unification

A *first-order equational problem* is given through the syntax of *first-order terms*, *substitutions* and their *action* over terms. A *signature* over first-order terms is a set of function symbols with an associated arity, that may have associated *equational properties*. Variables are represented by capital letters $X, Y, Z, \ldots$, and function symbols by lowercase letters $f, g, h, \ldots$.

Definition 2.1 (Signature)**.** *A signature Σ is a countable set of function symbols f with an associated* arity*. Each function symbol may have associated* equational properties*. If a function symbol has arity 0 it is called a* constant*. A function symbol that has no associated equational property is denominated* syntactic*.*

Remark 2.1. *C function symbols are assumed binary, and, exceptionally, AC function symbols are taken with no fixed arity.*

Definition 2.2 (First-order terms $\mathcal{T}(\Sigma, \mathcal{V})$)**.** *Given a signature Σ and a countable set of variables $\mathcal{V}$, a* first-order term *is either a constant $c \in \Sigma$, a variable $X \in \mathcal{V}$, or a function symbol $f \in \Sigma$ with arity $n > 0$ applied to a tuple of n first-order terms. This definition is synthesised in the following expression:*

$$t := c \mid X \mid f(t, ..., t).$$

A first-order term is called syntactic*, if all its function symbols are syntactic.*

The following equational properties are from interest. For simplicity, they are exhibited free from universal quantifiers:

A	(associativity)	:=	$f(f(X,Y),Z) \approx f(X,f(Y,Z))$
C	(commutativity)	:=	$f(X,Y) \approx f(Y,X)$
U	(neutral element)	:=	$f(X,e) \approx X,\ f(e,X) \approx X$
I	(idempotence)	:=	$f(X,X) \approx X$
D	(distributivity)	:=	$f(X,g(Y,Z)) \approx g(f(X,Y),f(X,Z))$

Example 2.1. *Let $\Sigma = \{0,1,+,*\}$ be a signature with constants 0 and 1, that are, respectively, the neutral elements w.r.t. the ACDU binary function symbols $+$ and $*$. Then, in this signature, the following equations, between the following well-formed first-order terms, holds:*

- $*(1,*(+(+(1,X),+(1,Y)),0)) \approx *(*(+(+(1,X),+(1,Y)),0),1)$;
- $*(*(+(+(1,X),+(1,Y)),0),1) \approx *(+(+(1,X),+(1,Y)),0)$;
- $*(+(+(1,Y),+(1,X)),0) \approx *(+(1,Y),+(1,X))$;
- $+(+(1,X),+(*(Y,1),*(Y,X))) \approx +(+(1,X),+(Y,*(Y,X)))$.

Definition 2.3 (Size of a first-order term)**.** *The* size *of a first-order term t, denoted as $|t|$, is recursively defined as:* $|X| := 1, \quad |f(t_0,...,t_n)| := 1 + \sum_{0\le i\le n} |t_i|$.

Notation 2.1. *As usual, the notation $|_|$ will be also used to denote the cardinality of sets.*

Definition 2.4 (First-order problem)**.** *A* first-order problem *P over $\mathcal{T}(\Sigma,\mathcal{V})$ is a finite set of equations between first-order terms $\in \mathcal{T}(\Sigma,\mathcal{V})$, of the form $\{s_0 \approx_? t_0,...,s_n \approx_? t_n\}$. If s_i, t_i for $i = 0..n$ are syntactic first-order terms, P is called a* syntactic first-order problem*.*

Example 2.2.

$$P = \left\{ \begin{array}{l} +(+(1,X),+(Y,*(Y,X))) \approx_? +(+(1,W),+(0,*(0,Y))), \\ *(Z,W) \approx_? *(+(+(1,X),+(1,Y)),0) \end{array} \right\}$$

is a first-order problem over the signature $\Sigma = \{0,1,+,*\}$ from Ex. 2.1.

Definition 2.5 (Sets of variables)**.**

1. *The set of variables occurring in a term t will be denoted by $Var(t)$;*

2. *This notation extends to a set S of terms in a natural way: $Var(S) := \bigcup_{t\in S} Var(t)$;*

3. *Let P be a first-order problem, $Var(P)$ denotes the set $\bigcup_{s\approx_? t\in P} Var(s) \cup Var(t)$.*

Definition 2.6 (Substitution)**.**

1. *A* substitution *σ is a mapping from variables to terms such that its* domain*, defined by $dom(\sigma) := \{X \mid X \neq X\sigma\}$, is finite;*

2. *For $X \in dom(\sigma)$, $X\sigma$ is called the* image *of X by σ;*

3. *Define the* image *of σ as $im(\sigma) := \{X\sigma \mid X \in dom(\sigma)\}$.*

4. *Let $dom(\sigma) = \{X_0, \cdots, X_n\}$, then σ can be represented as a set of the form $\{X_0/t_0, \cdots, X_n/t_n\}$, where $X_i\sigma = t_i$, for $0 \leq i \leq n$.*

Definition 2.7 (Composition of substitutions)**.**
The composition of substitutions

$$\sigma = \{X_0/t_0, \cdots, X_n/t_n\} \text{ and } \delta = \{X_{n+1}/t_{n+1}, \cdots, X_{n+k+1}/t_{n+k+1}\},$$

denoted by $\sigma\delta$, is defined by $\{X_0/t_0\delta, \cdots, X_n/t_n\delta,\ X_{n+1}/t_{n+1}, \cdots, X_{n+k+1}/t_{n+k+1}\}$.

Definition 2.8 (Action of a substitution over a first-order term)**.** *A substitution σ* acts *over a first-order term recursively:*

$$c\sigma := c \ \mid \ f(t_0, ..., t_n)\sigma := f(t_0\sigma, ..., t_n\sigma).$$

Example 2.3.

a) $X\{X/a, Z/b\} = a$;

b) $f(X, Y, c)\{X/a, Y/b, Z/d\} = f(a, b, c)$;

c) $g(f(X, Y), f(Z, W))\{X/a, Y/b, Z/c, W/d\} = g(f(a, b), f(c, d))$.

Definition 2.9 (Action of a substitution over a fist-order problem)**.** *The action of a substitution σ over a first-order problem P is defined by $P\sigma := \bigcup_{s\approx_? t\in P}\{s\sigma \approx_? t\sigma\}$.*

Definition 2.10 (First-order equality-checking)**.** *Let P be a first-order problem over $\mathcal{T}(\Sigma, \mathcal{V})$. The* equality-checking *of P returns* success *($\top$) if for each $s \approx_? t \in P$ it is true that $s \approx t$. Otherwise it returns* fail *($\bot$).*

Definition 2.11 (First-order solutions). *Let P be a first-order problem over $\mathcal{T}(\Sigma, \mathcal{V})$. The substitution σ is called a* first-order unification (resp. matching) solution *for the problem P, if for each $s \approx_? t \in P$ it is true that $s\sigma \approx t\sigma$ (resp. $s\sigma \approx t$). The set of first-order unification (resp. matching) solutions of P is denoted by $\mathcal{U}(P)$ (resp. $\mathcal{M}(P)$).*

Definition 2.12 (Equivalence between substitutions). *Let $\approx$ be an equivalence relation, and σ and δ substitutions. σ is said equivalent to δ modulo $\approx$, denoted by $\sigma \approx \delta$, if for every X in $dom(\sigma) \cup dom(\delta)$, $X\sigma \approx X\delta$.*

Definition 2.13 (More general solutions and complete set of solutions). *Let P be a first-order problem and $\sigma, \delta \in \mathcal{U}(P)$ (resp. $\mathcal{M}(P)$). Then, σ is* more general than *δ, denoted by $\sigma \preccurlyeq \delta$, if there exists λ such that $\sigma\lambda \approx \delta$. A set $\mathcal{S} \subseteq \mathcal{U}(P)$ (resp. $\mathcal{M}(P)$) is* complete *if for every $\gamma \in \mathcal{U}(P)$ (resp. $\mathcal{M}(P)$), there exists $\sigma \in \mathcal{S}$, such that $\sigma \preccurlyeq \gamma$.*

Definition 2.14 (Independent solutions). *Let P be a first-order problem and $\sigma, \delta \in \mathcal{U}(P)$ (resp. $\mathcal{M}(P)$). σ and δ are called* independent solutions, *if neither $\sigma \preccurlyeq \delta$, nor $\delta \preccurlyeq \sigma$.*

Definition 2.15 (Minimal-complete set of solutions). *A complete set $\mathcal{S} \subseteq \mathcal{U}(P)$ (resp. $\mathcal{M}(P)$) is* minimal *if for any $\sigma, \delta \in \mathcal{S}$, were σ is not equivalent to δ, σ and δ are independent.*

Remark 2.2. *Let P be a first-order unification problem over $\mathcal{T}(\Sigma, \mathcal{V})$, Subsec. 2.1.2 shows that the chosen signature Σ has influence in the expected cardinality of a minimal-complete set $\mathcal{S} \subseteq \mathcal{U}(P)$ (resp. $\mathcal{M}(P)$). The* type *associated with a signature Σ is* unitary, *(resp.* finitary *and* infinitary*), denoted by 1 (resp. ω or ∞), if the expected cardinality of a minimal-complete set is at most unitary (finite or countable). If it is impossible to build such a minimal-complete set, the associated type is* zero *(0).*

2.1.1 First-order syntactic unification

In this Subsec., the seminal unification algorithm proposed by Robinson [60] is shortly described. This algorithm can be expressed by the set of simplification rules of Fig. 2.1 over pairs of substitutions and syntactic first-order problems. In these simplification rules, the symbol $\uplus$ represents the operator of disjoint union. In the following, capital calligraphy letters $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{R}$, ..., will denote pairs of substitutions and problems. For a unification problem P, the input of the algorithm is a pair $\mathcal{P} = \langle id, P \rangle$ composed by the identity substitution and the problem itself.

Notation 2.2. *Reductions using system of Fig. 2.1 are denoted by $\Rightarrow$, thus $\mathcal{P} \Rightarrow \mathcal{Q}$ means that the second pair is obtained from the first by an application of a simplification rule.*

$$\frac{\langle \sigma, P \uplus \{s \approx_? s\}\rangle}{\langle \sigma, P\rangle} \ \begin{array}{l} s \text{ is a constant} \\ \text{or a variable}\end{array} \ (\textbf{Trivial}) \qquad \frac{\langle \sigma, P \uplus \{t \approx_? X\}\rangle}{\langle \sigma, P \uplus \{X \approx_? t\}\rangle} \ \begin{array}{l} t \text{ is not a} \\ \text{variable}\end{array} \ (\textbf{Orient})$$

$$\frac{\langle \sigma, P \uplus \{f(s_0, ..., s_m) \approx_? g(t_0, ..., t_n)\}\rangle}{\bot} \ f \neq g \ (\textbf{Clash})$$

$$\frac{\langle \sigma, P \uplus \{f(s_0, ..., s_n) \approx_? f(t_0, ..., t_n)\}\rangle}{\langle \sigma, P \cup \{s_0 \approx_? t_0, ..., s_n \approx_? t_n\}\rangle} \ (\textbf{Decomp})$$

$$\frac{\langle \sigma, P \uplus \{X \approx_? t\}\rangle}{\bot} \ \begin{array}{l} X \in Var(t) \text{ and} \\ t \text{ is not a variable}\end{array} \ (\textbf{Occurs}) \qquad \frac{\langle \sigma, P \uplus \{X \approx_? t\}\rangle}{\langle \sigma\{X/t\}, P\{X/t\}\rangle} \ X \notin Var(t) \ (\textbf{Elim})$$

Figure 2.1: Simplification rules for the Robinson unification algorithm

Definition 2.16 (Set of variables of a pair)**.** *The* set of variables of a pair $\mathcal{P} = \langle \sigma, P\rangle$, *denoted by* $Var(\mathcal{P})$*, is defined as being equal to* $Var(P)$*.*

Lemma 2.1 (Termination of $\Rightarrow$)**.** *The relation $\Rightarrow$ is terminating.*

Proof. The proof is by case analysis on the derivation rules, using the decreasing lexicographic measure over pairs, such that for $\mathcal{P} = \langle \sigma, P\rangle$ it is given by the triple:

$$\langle |Var(P)|, \sum_{s \approx_? t \in P} |s| + |t|, \begin{array}{l} \text{number of equations} \\ \text{in } P \text{ in the form } t \approx_? X \end{array} \rangle$$

For rules (**Clash**) and (**Occurs**) it is assumed that the measure of $\bot$ is equal to $\langle 0, 0, 0\rangle$. Notice that in reductions by rules (**Trivial**), (**Decomp**) the second coordinate of the measure decreases, while the first either decreases or keeps the same. A reduction by rule (**Orient**) reduces the third coordinate of the measure and keeps the values of the first two coordinates. Finally, in reductions by (**Elim**) the first coordinate of the measure decreases. □

Notation 2.3 (Standard rewriting notation)**.** *The standard rewriting nomenclature will be used, e.g.,* $\mathcal{P}$ *will be called a* normal form *or* irreducible *by* $\Rightarrow$*, denoted by* $\Rightarrow$-nf*, whenever there is no* $\mathcal{Q}$ *such that* $\mathcal{P} \Rightarrow \mathcal{Q}$*;* $\Rightarrow^*$ *and* $\Rightarrow^+$ *denote respectively derivations in zero or more and one or more applications of the rules.*

Example 2.4. *Considering the problem*

$$P = \left\{ \begin{array}{l} +(+(1, X), +(Y, *(Y, X))) \approx_? +(+(1, W), +(0, *(0, Y))), \\ *(Z, W) \approx_? *(+(+(1, X), +(1, Y)), 0) \end{array} \right\}$$

of Ex. 2.2, over $\Sigma = \{0, 1, +, *\}$, *where* $+$ *and* $*$ *are considered syntactic. The system of Fig. 2.1 derives* $\mathcal{P} = \langle id, P\rangle$ *to* $\langle\{X/0,\ Y/0,\ W/0, Z/+(+(1,0), +(1,0))\}, \emptyset\rangle$, *that is a* $\Rightarrow$*-nf. The simplification steps are given by the following. Highlighted equations show where the rules are applied.*

$$
\begin{aligned}
\mathcal{P} =\ & \langle id, \left\{ \begin{array}{l} \boxed{+(+(1,X), +(Y, *(Y,X))) \approx_? +(+(1,W), +(0, *(0,Y)))}, \\ *(Z,W) \approx_? *(+(+(1,X), +(1,Y)), 0) \end{array} \right\}\rangle \\
\Rightarrow_{(\mathbf{Decomp})}\ & \langle id, \left\{ \begin{array}{l} \boxed{+(1,X) \approx_? +(1,W)}, \\ +(Y, *(Y,X)) \approx_? +(0, *(0,Y)), \\ *(Z,W) \approx_? *(+(+(1,X), +(1,Y)), 0) \end{array} \right\}\rangle \\
\Rightarrow_{(\mathbf{Decomp})}\ & \langle id, \left\{ \begin{array}{l} \boxed{1 \approx_? 1},\ X \approx_? W, \\ +(Y, *(Y,X)) \approx_? +(0, *(0,Y)), \\ *(Z,W) \approx_? *(+(+(1,X), +(1,Y)), 0) \end{array} \right\}\rangle \\
\Rightarrow_{(\mathbf{trivial})}\ & \langle id, \left\{ \begin{array}{l} \boxed{X \approx_? W}, \\ +(Y, *(Y,X)) \approx_? +(0, *(0,Y)), \\ *(Z,W) \approx_? *(+(+(1,X), +(1,Y)), 0) \end{array} \right\}\rangle \\
\Rightarrow_{(\mathbf{elim})}\ & \langle \{X/W\}, \left\{ \begin{array}{l} \boxed{+(Y, *(Y,W)) \approx_? +(0, *(0,Y))}, \\ *(Z,W) \approx_? *(+(+(1,W), +(1,Y)), 0) \end{array} \right\}\rangle \\
\Rightarrow_{(\mathbf{decomp})}\ & \langle \{X/W\}, \left\{ \begin{array}{l} \boxed{Y \approx_? 0},\ *(Y,W) \approx_? *(0,Y), \\ *(Z,W) \approx_? *(+(+(1,W), +(1,Y)), 0) \end{array} \right\}\rangle \\
\Rightarrow_{(\mathbf{elim})}\ & \langle \{X/W,\ Y/0\}, \left\{ \begin{array}{l} \boxed{*(0,W) \approx_? *(0,0)}, \\ *(Z,W) \approx_? *(+(+(1,W), +(1,0)), 0) \end{array} \right\}\rangle \\
\Rightarrow_{(\mathbf{decomp})}\ & \langle \{X/W,\ Y/0\}, \left\{ \begin{array}{l} \boxed{0 \approx_? 0},\ W \approx_? 0, \\ *(Z,W) \approx_? *(+(+(1,W), +(1,0)), 0) \end{array} \right\}\rangle \\
\Rightarrow_{(\mathbf{trivial})}\ & \langle \{X/W,\ Y/0\}, \left\{ \begin{array}{l} \boxed{W \approx_? 0}, \\ *(Z,W) \approx_? *(+(+(1,W), +(1,0)), 0) \end{array} \right\}\rangle \\
\Rightarrow_{(\mathbf{elim})}\ & \langle \{X/0,\ Y/0,\ W/0,\}, \left\{ \boxed{*(Z,0) \approx_? *(+(+(1,0), +(1,0)), 0)} \right\}\rangle \\
\Rightarrow_{(\mathbf{decomp})}\ & \langle \{X/0,\ Y/0,\ W/0,\}, \{ \boxed{Z \approx_? +(+(1,0), +(1,0))},\ 0 \approx_? 0\}\rangle \\
\Rightarrow_{(\mathbf{elim})}\ & \langle \{X/0,\ Y/0,\ W/0, Z/+(+(1,0), +(1,0))\}, \{ \boxed{0 \approx_? 0} \}\rangle \\
\Rightarrow_{(\mathbf{trivial})}\ & \langle \{X/0,\ Y/0,\ W/0, Z/+(+(1,0), +(1,0))\}, \emptyset\rangle
\end{aligned}
$$

Observe that $\sigma = \{X/0,\ Y/0,\ W/0, Z/+(+(1,0),+(1,0))\} \in \mathcal{U}(P)$, *since*

$$\begin{array}{rll} +(+(1,X),+(Y,*(Y,X)))\sigma & \approx +(+(1,0),+(0,*(0,0))) \approx & +(+(1,W),+(0,*(0,Y)))\sigma \text{ and} \\ *(Z,W)\sigma & \approx *(+(+(1,0),+(1,0)),0) \approx & *(+(+(1,X),+(1,Y)),0)\sigma \end{array}.$$

Example 2.5. *Let* $P = \{f(f(a,X),c) \approx_? f(g(a,b),c)\}$ *and* $Q = \{f(f(a,X),c) \approx_? f(f(a,h(X)),c)\}$ *syntactic unification problems. Then, the simplifications of a)* $\mathcal{P} = \langle id, P\rangle$ *and b)* $\mathcal{Q} = \langle id, Q\rangle$ *using rules of Fig. 2.1 result both in* $\perp$, *as show the following derivations:*

a)

$$\begin{array}{rl} \mathcal{P} = & \langle id, \{ f(f(a,X),c) \approx_? f(g(a,b),c) \}\rangle \\ \Rightarrow_{(\textbf{Decomp})} & \langle id, \{ f(a,X) \approx_? g(a,b), c \approx_? c\}\rangle \\ \Rightarrow_{(\textbf{Clash})} & \perp \end{array}$$

b)

$$\begin{array}{rl} \mathcal{Q} = & \langle id, \{ f(f(a,X),c) \approx_? f(f(a,h(X)),c) \}\rangle \\ \Rightarrow_{(\textbf{Decomp})} & \langle id, \{ f(a,X) \approx_? f(a,h(X)), c \approx_? c\}\rangle \\ \Rightarrow_{(\textbf{Decomp})} & \langle id, \{ a \approx_? a, X \approx_? h(X), c \approx_? c\}\rangle \\ \Rightarrow_{(\textbf{Trivial})} & \langle id, \{ X \approx_? h(X), c \approx_? c\}\rangle \\ \Rightarrow_{(\textbf{Occurs})} & \perp. \end{array}$$

Notice that the set $\mathcal{S} = \{\sigma \mid \langle id, P\rangle \Rightarrow^* \langle \sigma, \emptyset\rangle\}$ is at most unitary. Although rules of Fig. 2.1 are applied non-deterministically, each equation has a unique way of simplification. Thus derivations has no branches. Moreover, the algorithm defined by $\Rightarrow$ has been showed sound and complete ([52]). The proofs of these properties are based on analysis of the derivation rules. Efficient versions of the algorithm of Fig. 2.1 were proposed independently by [51] and [56]. Both first-order syntactic unification algorithms have been showed linearly bounded in time and space.

2.1.2 First-order A, C and AC-unification

This section presents unification algorithms to solve first-order unification problems in a signature Σ that may contain, beyond the syntactic, A, C and AC function symbols.

First-order A-unification

First-order A-unification is translated to the problem of solving equations in free semigroups, and this problem has been showed to have an infinite set of solutions in [50]. The classic example of an A-unification problem that have an infinite set of solutions is given by $P = \{f(X, c) \approx_? f(c, X)\}$ where f, c are, respectively, a binary associative function symbol and a constant. The minimal-complete set of first-order solutions of P contains the following infinite set:

$$\{\{X/c\},\ \{X/f(c, c)\},\ \{X/f(c, f(c, c))\},\ \{X/f(f(c, c), f(c, c))\},\ \ldots\}.$$

First-order C-unification

A first-order C-unification algorithm can be obtained by a simple extension of the Robinson algorithm (Fig. 2.1), adding rule (**C**) of Fig. 2.2, and imposing a restriction at rule $\Rightarrow_{(\mathbf{Decomp})}$, that is applied only to non-commutative function symbols.

$$\frac{\langle \sigma, P \uplus \{f(s_0, s_1) \approx_? f(t_0, t_1)\}\rangle}{\langle \sigma, P \cup \{s_0 \approx_? t_i,\ s_1 \approx_? t_{1-i}\}\rangle}\ f \text{ is C and } i = 0, 1\ (\mathbf{C})$$

Figure 2.2: Commutative rule

Remark 2.3. *From now, the relation $\Rightarrow$ will represent reduction steps of the first-order C-unification algorithm defined by Figs. 2.1 and 2.2.*

Lemma 2.2 (Termination of $\Rightarrow$)**.** *The relation $\Rightarrow$ is terminating.*

Proof. The proof is based on case analysis with well-founded induction on the derivation rules of $\Rightarrow$. It is used the same lexicographic measure of proof of Lem. 2.1, and all the cases, except that of rule (**C**), were already explored. For rule (**C**), one just observes that $|\{f(s_0, s_1) \approx_? f(t_0, t_1)\}| = |\{s_0 \approx_? t_i,\ s_1 \approx_? t_{1-i}\}| + 2 > |\{s_0 \approx_? t_i,\ s_1 \approx_? t_{1-i}\}|$, for $i = 0, 1$, and then the second coordinate of the lexicographic measure is greater before application of the rule, while the first coordinate keeps the same value. □

Example 2.6. *Let $P = \langle id,\ \{f_0^C(f_1^C(X, Y), f_1^C(a, b)) \approx_? f_0^C(f_1^C(c, d), f_1^C(Z, W))\}\rangle$ be a first-order C-unification problem and f_0^C, f_1^C commutative function symbols. Reducing $\langle id, P\rangle$ by $\Rightarrow$, one obtains the following set of $\Rightarrow$-nf's:*

$$\langle\{X/c,\ Y/d,\ Z/a,\ W/b\},\ \emptyset\rangle,$$
$$\langle\{X/c,\ Y/d,\ W/a,\ Z/b\},\ \emptyset\rangle,$$
$$\langle\{X/d,\ Y/c,\ Z/a,\ W/b\},\ \emptyset\rangle,$$
$$\langle\{X/d,\ Y/c,\ W/a,\ Z/b\},\ \emptyset\rangle \text{ and } \bot.$$

This reduction can be represented by the labelled tree of Fig 2.3, where the label of the root is $\langle id, P\rangle$ and labels of the other nodes are obtained after application of rules of the C-unification algorithm to their parents. This representation is called derivation tree *of $\langle id, P\rangle$. Observe that, rule* (**C**) *is the rule that generates branches in the derivation tree. The right-hand side (*rhs*) derivations of Fig 2.3 are omitted because their eight generated leaves have labels equal to $\bot$, as result of applications of rule* (**Clash**)*. Nevertheless, the left-hand side (*lhs*) branch of the tree contains four leaves whose substitutions in the labels are solutions for the input problem.*

Given a first-order nominal C-problem P, notice that the set $\{\sigma \mid \langle id, P\rangle \Rightarrow^* \langle \sigma, \emptyset\rangle\}$ is finitary. Equations have finite occurrences of C function symbols, then the rule (**C**) generates only a finite number of branches. Termination of the algorithm was already showed in Lem. 2.2, and the proofs of soundness and completeness are also performed by case analysis on the application of the simplification rules. One concludes that first-order C-unification has type finitary. These last results can be found in [64].

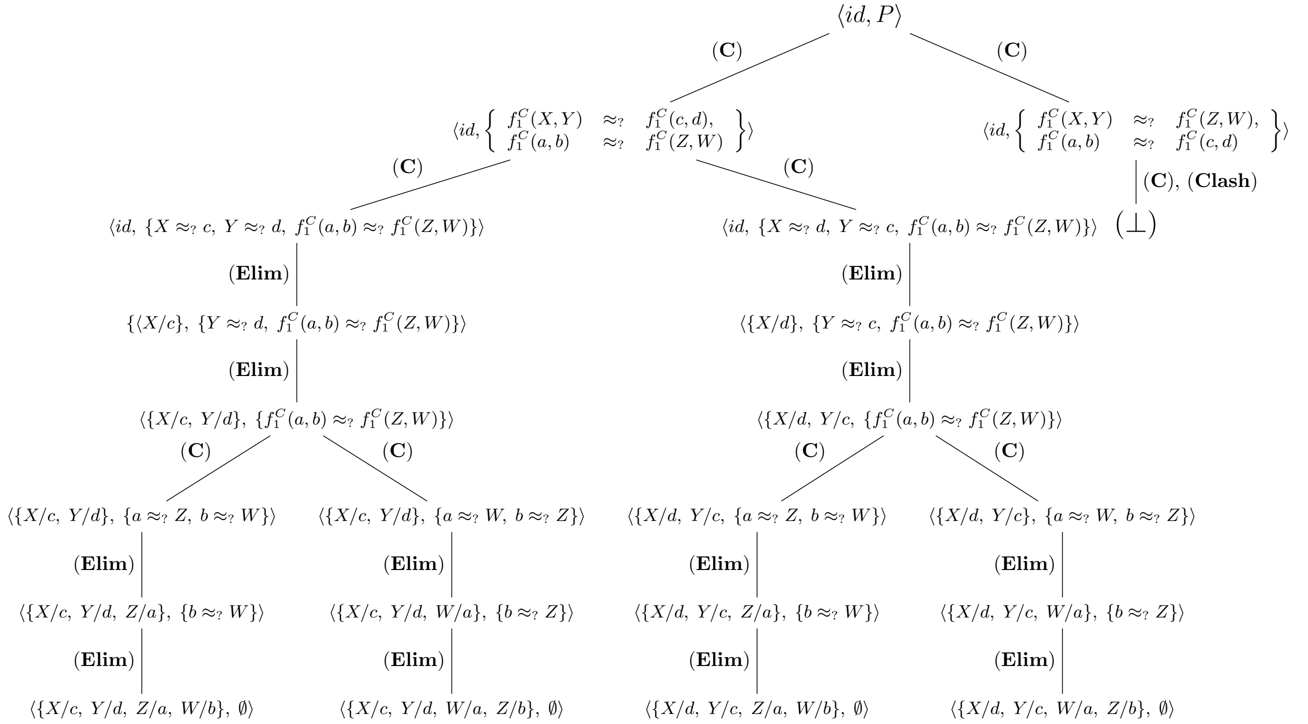

Figure 2.3: Derivation tree of Ex. 2.6.

Remark 2.4. *The presented first-order C-unification algorithm may generate a set of solutions that is not minimal. For instance, given the problem $P = \{f\langle X, a\rangle \approx_? f\langle a, Y\rangle\}$, where f is a C function symbol, the algorithm outputs the solutions $\{X/Y\}$ and $\{X/a, Y/a\}$, but clearly $\{X/Y\} \preccurlyeq \{X/a, Y/a\}$. To obtain such a minimal complete set one can*

execute the presented algorithm, deciding for each σ and δ in the obtained solutions set if these substitutions are or not independent, keeping more general solutions and removing redundancies afterwards.

Remark 2.5 (First-order C-unification NP-completeness).
Deciding if a first-order C-unification problem is solvable is an NP-complete problem. To prove that the problem is NP, a non-deterministic decision procedure using the reduction rules of $\Rightarrow$ is built. In this procedure, whenever rule (**C**) *applies, only one of the two possible branches is guessed. In this manner, if the derivation tree has a solution as a leaf, this procedure guesses a path to the solution, answering positively to the decision problem. According to the measure used in the proof of termination (Lem. 2.2), the reduction $\Rightarrow$ is polynomially bounded, which implies that this non-deterministic procedure is polynomially bounded.*

To prove NP-completeness, one can polynomially reduce the well-known NP-complete positive 1-in-3-SAT problem into first-order C-unification, as done in [14]. An instance of the positive 1-in-3-SAT problem consists of a set of clauses $\mathcal{C} = \{\mathcal{C}_i | 1 \leq i \leq n\}$, where each $\mathcal{C}_i$ is a disjunction of three propositional variables, say $\mathcal{C}_i = p_i \vee q_i \vee r_i$. A solution of $\mathcal{C}$ is a valuation with exactly one variable true in each clause.

The proposed reduction of $\mathcal{C}$ into a first-order C-unification problem would require just a commutative function symbol, say f, two constants, say c_0 and c_1, a variable for each clause $\mathcal{C}_i$, say Y_i, and a variable for each propositional variable p in $\mathcal{C}$, say X_p. Instantiating X_p as c_0 or c_1, would be interpreted as evaluating p as true or false, respectively. Each clause $\mathcal{C}_i = p_i \vee q_i \vee r_i$ in $\mathcal{C}$ is translated into an equation E_i of the form

$$f(f(f(X_{p_i}, X_{q_i}), X_{r_i}), Y_i) \approx_? f(f(f(c_0, c_1), c_0), f(f(c_1, c_0), c_1)).$$

The first-order C-unification problem for $\mathcal{C}$ is given by $\mathcal{P}_{\mathcal{C}} = \{E_i | 1 \leq i \leq n\}$. Thus, to conclude it is only necessary to check that σ is a solution for $\mathcal{P}_{\mathcal{C}}$ if and only if σ instantiates exactly one of the variables X_{p_i}, X_{q_i} and X_{r_i} in each equation with c_0 and the other two with c_1, which means that $\mathcal{C}$ has a solution.

First-order AC-unification

Standard first-order AC-unification algorithms use algebraic approaches, such as translating the unification problem to solving a system of Diophantine equations. Such approach is illustrated in Ex. 2.7.

Example 2.7 (Adapted from [67]). *Let $P = \{f(X, X, Y, a, b, c) \approx_? f(b, b, b, c, Z)\}$ be a nominal AC-unification problem and f an AC function symbol.*

To find a sound and complete set of solutions, one first execute a "pre-cooking" process, where common elements are eliminated in the lhs *and* rhs *of the equations.* P *is transformed into* $P' = \{f(X, X, Y, a) \approx_? f(b, b, Z)\}$.

In the second step, the algorithm transforms P'*, replacing constants and subterms that are headed by different function symbols by the new variables* W_0 *and* W_1*, The result is* $P'' = \{f(X, X, Y, W_0) \approx_? f(W_1, W_1, Z)\}$.

Then, observing the multiplicity of occurrences of each variable, the equations of the problem are translated into Diophantine equations. $f(X, X, Y, W_0) \approx_? f(W_1, W_1, Z)$ *is translated into* $2X + Y + W_0 = 2W_1 + Z$. *Each solution of this equation must consists of non-negative integers, which is represented by a basis of new variables* $Z_0, ..., Z_6$*, as shows Table 2.1.*

Table 2.1: Solutions for the equation $2X + Y + W_0 = 2W_1 + Z$

X	Y	W_0	W_1	Z	$2X+Y+W_0$	$2W_1+Z$	
0	0	1	0	1	1	1	Z_0
0	1	0	0	1	1	1	Z_1
0	0	2	1	0	2	2	Z_2
0	1	1	1	0	2	2	Z_3
0	2	0	1	0	2	2	Z_4
1	0	0	0	2	2	2	Z_5
1	0	0	1	0	2	2	Z_6

Observing the columns of Table 2.1, one obtains the following equations:

$$\begin{aligned} X &= Z_5 + Z_6 \\ Y &= Z_1 + Z_3 + 2Z_4 \\ W_0 &= Z_0 + 2Z_2 + Z_3 \\ W_1 &= Z_2 + Z_3 + Z_4 + Z_6 \\ Z &= Z_0 + Z_1 + 2Z_5 \end{aligned}$$

There exist $2^7 = 128$ *possibilities of including or not variables* $Z_0, ..., Z_6$ *in the solutions. This set is restricted to* 69 *cases where variables* X, Y, Z, W_0 *and* W_1 *are simultaneously* $\neq 0$. *Also, cases where* W_0 *and* W_1*, that are originally associated to constants, and now are designated to more than one variable (example* $W_0 = Z_0 + 2Z_2 + Z_3$*) are eliminated. After this elimination step, there are only six remaining cases:*

$$
\begin{array}{lllllll}
\sigma_0 & = & \{X/Z_5, & Y/Z_3, & W_0/Z_3, & W_1/Z_3, & Z/f(Z_5, Z_5)\} \\
\sigma_1 & = & \{X/Z_5, & Y/f(Z_1, Z_3), & W_0/Z_3, & W_1/Z_3, & Z/f(Z_1, Z_5, Z_5)\} \\
\sigma_2 & = & \{X/Z_5, & Y/f(Z_4, Z_4), & W_0/Z_0, & W_1/Z_4, & Z/f(Z_0, Z_5, Z_5)\} \\
\sigma_3 & = & \{X/Z_5, & Y/f(Z_1, Z_4, Z_4), & W_0/Z_0, & W_1/Z_4, & Z/f(Z_0, Z_1, Z_5, Z_5)\} \\
\sigma_4 & = & \{X/Z_6, & Y/Z_1, & W_0/Z_0, & W_1/Z_6, & Z/f(Z_0, Z_1)\} \\
\sigma_5 & = & \{X/f(Z_5, Z_6), & Y/Z_1, & W_0/Z_0, & W_2/Z_6, & Z/f(Z_0, Z_1, Z_5, Z_5)\}
\end{array}
$$

Observe that the variables in the images of W_0 and W_1 must be mapped, respectively, to distinct constants a and b. This step eliminates σ_0 and σ_1 because in these cases W_0 and W_1 are mapped to the same variable. Then, the set of solution candidates for the original problem $P = \{f(X, X, Y, a, b, c) \approx_? f(b, b, b, c, Z)\}$ is given by:

$$
\begin{array}{lllll}
\sigma'_2 & = & \{X/Z_5, & Y/f(b, b), & Z/f(a, Z_5, Z_5)\} \\
\sigma'_3 & = & \{X/Z_5, & Y/f(Z_1, b, b), & Z/f(a, Z_1, Z_5, Z_5)\} \\
\sigma'_4 & = & \{X/b, & Y/Z_1, & Z/f(a, Z_1)\} \\
\sigma'_5 & = & \{X/f(Z_5, b), & Y/Z_1, & Z/f(a, Z_1, Z_5, Z_5)\}
\end{array}
$$

Finally, after normalising $\sigma'_2, \sigma'_3, \sigma'_4$ and σ'_5, by replacing variables in the images that already occur in the domain of the substitutions, the set of general solutions for the initial problem is given by:

$$
\mathcal{U}(P) = \left\{ \begin{array}{c} \{Y/f(b, b),\ Z/f(a, X, X)\}, \\ \{Y/f(Z_1, b, b),\ Z/f(a, Z_1, X, X)\}, \\ \{X/b,\ Z/f(a, Y)\}, \\ \{X/f(Z_5, b),\ Z/f(a, Y, Z_5, Z_5)\} \end{array} \right\}
$$

The following list summarises the steps executed in the previous example by the first-order AC-unification algorithm:

1. Common terms in the *lhs* and *rhs* are eliminated;
2. A mapping from constants and subterms that are headed by different function symbols to a set of new variables is generated;
3. A corresponding system of Diophantine equations is generated observing the multiplicity of the variables in each equation of the problem;

4. The positive solutions of the system of Diophantine equations are generated;
5. A new variable is associated with each solution;
6. Solution candidates are built for each group of variables that does not designates zero to the original variables of the problem;
7. In the solution candidates, the images of the variables of step 2 are replaced by their respective designated terms;
8. The feasible cases are normalised observing the variables that occur in the image and the domain of the substitutions and composing the binds.

The algorithm described above was proved terminating, sound and complete by Stickel [67] and Fages [38].

The following Table 2.2 summarises the type of unification and complexity of first-order problems under some equational theories. Also complexities and types of equality-checking and matching problems are considered in the table. The column "Related work" presents some references.

Table 2.2: Types and complexities of first-order equational problems

		Complexities			
Equational theory	Unification type	Equality-checking	Matching	Unification	Related work
Syntactic	1	$O(n)$	$O(n)$	$O(n)$	[60, 51, 56]
C	ω	$O(n^2)$	NP-complete	NP-complete	[17, 44]
A	∞	$O(n)$	NP-complete	decidable	[50, 17]
AU	∞	$O(n)$	NP-complete	decidable	[50, 44]
AI	0	$O(n)$	NP-hard	NP-hard	[43]
AC	ω	$O(n^3)$	NP-complete	NP-complete	[17, 44, 45]
ACU	ω	$O(n^3)$	NP-complete	NP-complete	[45]
D	ω	-	NP-hard	NP-hard	[68]

2.2 Nominal syntax

The *nominal syntax* ([57]) is defined using a new class of objects called *atoms*, represented by $\mathcal{A}$. Atoms are the simplest structure, just object-level variables, and they only differ in their names, so for $a, b \in \mathcal{A}$ the expression $a \neq b$ is redundant. Atoms are denoted by lower-case roman characters: a, b, c, etc. Together with this new class, the set of *atom permutations* Π, i.e., atom bijections over $\mathcal{A}$ with a finite domain, is also used. A *swapping* is a particular permutation that has just two atoms a, b in his domain, and it is denoted

by a pair $(a\,b)$, where a is mapped to b and b to a simultaneously. A permutation π can be expressed by a finite list of swappings of the form $(a_1\,b_1) :: \ldots :: (a_n\,b_n) :: nil$, where nil denotes the identity permutation.

When no confusion arises, the identity (nil) and the symbol "::" will be omitted from lists of swappings. The symbols $\oplus$ and π^{-1} stand, respectively, for the list concatenation operation and the reverse list of π.

A *nominal term* is defined as being either: an *empty tuple* (or *unity*) $\langle\rangle$; an *atom as term object* $\overline{a}$; an *abstraction* $[a]t$; an *application* of a function symbol to a term $f\,t$; a *pair* $\langle u, v\rangle$; or a *suspension* $\pi.X$. In the last, the action of π is suspended until X is instantiated. When composed, pairs generate tuples $\langle t_0, ..., t_n\rangle$ with arbitrary bracketing and number of elements, but notice that the construction of unary tuples is not allowed. Also, observe that function symbols have no fixed arity. Thus, the application of a function symbol to the ($\langle\rangle$) may be interpreted as the neutral element in the given signature. For instance, $\wedge\langle\rangle$, $\vee\langle\rangle$, $+\langle\rangle$ and $\times\langle\rangle$ might be interpreted as "false", "true", 0 and 1, respectively.

Definition 2.17 (Nominal terms)**.** *The set of* nominal terms *is generated by the following grammar:*

$$s, t \;::=\; \langle\rangle \;\mid\; \overline{a} \;\mid\; [a]t \;\mid\; \langle s, t\rangle \;\mid\; f\,t \;\mid\; \pi.X$$

with $a \in \mathcal{A}$, $\pi \in \Pi$ and $f \in \Sigma$.

From now, if no confusion arises, nominal terms will be called just terms.

Definition 2.18. *The* size of a nominal term t*, denoted as $|t|$, is recursively defined as:*

$$|\langle\rangle| \;:=\; 1, \;\; |\overline{a}| \;:=\; 1, \;\; |\pi.X| \;:=\; 1,$$
$$|[a]t| \;:=\; |t| + 1, \;\; |\langle u, v\rangle| \;:=\; |u| + |v| + 1, \;\; |f\,s| \;:=\; |s| + 1.$$

Definition 2.19 (Action of permutations over atoms)**.** *The action of a permutation over atoms is recursively defined as:*

$$((a\,b) :: \pi)\cdot a := \pi\cdot b, \;\; ((a\,b) :: \pi)\cdot b := \pi\cdot a, \;\; ((a\,b) :: \pi)\cdot c := \pi\cdot c, \;\; nil\cdot a := a.$$

Definition 2.20 (Action of permutations over terms)**.** *The action of a permutation over terms is specified as the homeomorphic extension of the action of permutations over atoms:*

$$\begin{array}{lll} \pi\cdot\langle\rangle \;:=\; \langle\rangle & \pi\cdot\langle u, v\rangle \;:=\; \langle\pi\cdot u, \pi\cdot v\rangle & \pi\cdot f\,t \;:=\; f\,(\pi\cdot t) \\ \pi\cdot\overline{a} \;:=\; \overline{\pi\cdot a} & \pi\cdot([a]t) \;:=\; [\pi\cdot a](\pi\cdot t) & \pi\cdot(\pi'\,.\,X) \;:=\; (\pi'\oplus\pi)\,.\,X \end{array}$$

Example 2.8. *The permutation $(a\;e)(b\;e)(c\;e)(d\;e)$ acting over the term*

$$f\,[a]\langle\langle\overline{a}, \overline{c}\rangle, \langle\langle g\,\overline{d}, \overline{b}\rangle, \langle\overline{e}, X\rangle\rangle\rangle$$

has as result

$$f[b]\langle\langle\overline{b},\overline{d}\rangle,\langle\langle g\,\overline{e},\overline{c}\rangle,\langle\overline{a},(a\ e)(b\ e)(c\ e)(d\ e).X\rangle\rangle\rangle.$$

2.2.1 Freshness and the nominal α-equivalence

The nominal α-equivalence is defined through the inference rules of Fig. 2.4, where judgements of the form $\nabla \vdash s \approx_\alpha t$ are derived. In this sentence, ∇ is called a *freshness context* that is defined as a set of pairs in $\mathcal{A} \times \mathcal{V}$. The interesting rules for α-equivalence are those for abstractions and suspensions. For instance, there exist two possible rules (($\approx_\alpha$ **[aa]**) and ($\approx_\alpha$ **[ab]**)) for abstractions. The former and the latter checks, respectively, whether abstracted terms are α-equivalent under the same, or different abstracted atoms. For the latter rule, one needs to check whether renaming one of the abstracted terms t by swapping these different atoms, say $(a\,b)$, the α-equivalence with the other abstracted term s holds (the *equality judgement* $\nabla \vdash s \approx_\alpha (a\,b)\cdot t$). In addition, the atom a used in this renaming has to be fresh in the abstracted term t that is renamed. In this way, rule ($\approx_\alpha$ **[ab]**) has as premisse the *freshness judgment* $\nabla \vdash a \# t$ that is derived via inference rules of Fig. 2.5. This establishes the link between rules of Figs 2.4 and 2.5.

Definition 2.21 (Difference set)**.** *$ds(\pi,\pi') = \{a \mid \pi\cdot a \neq \pi'\cdot a\}$ is the set of atoms where π and π' differ (the* difference set*).*

Notation 2.4. *$ds(\pi,\pi')\#X$ is an abbreviation of $\{a \# X \mid a \in ds(\pi,\pi')\}$.*

$$\frac{}{\nabla \vdash \langle\rangle \approx_\alpha \langle\rangle}\,(\approx_\alpha \langle\rangle) \qquad \frac{}{\nabla \vdash \overline{a} \approx_\alpha \overline{a}}\,(\approx_\alpha \mathbf{atom}) \qquad \frac{\nabla \vdash s \approx_\alpha t}{\nabla \vdash f\,s \approx_\alpha f\,t}\,(\approx_\alpha \mathbf{app})$$

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

$$\frac{ds(\pi,\pi')\#X \subseteq \nabla}{\nabla \vdash \pi.X \approx_\alpha \pi'.X}\,(\approx_\alpha \mathbf{var}) \qquad \frac{\nabla \vdash s_0 \approx_\alpha t_0 \quad \nabla \vdash s_1 \approx_\alpha t_1}{\nabla \vdash \langle s_0, s_1\rangle \approx_\alpha \langle t_0, t_1\rangle}\,(\approx_\alpha \mathbf{pair})$$

Figure 2.4: Rules for the relation $\approx_\alpha$

Notation 2.5. *$dom(\Delta|_X)$, $dom(\Delta|_X)\#Y$ and $\nabla \vdash dom(\Delta|_X) \# s$ denote, respectively, the sets $\{a \mid a\#X \in \Delta\}$, $\{a\#Y \mid a \in dom(\Delta|_X)\}$ and $\{\nabla \vdash a \# s \mid a \in dom(\Delta|_X)\}$.*

$$\frac{}{\nabla \vdash a \,\#\, \langle\rangle}\,(\#\langle\rangle) \quad \frac{}{\nabla \vdash a \,\#\, \overline{b}}\,(\#\,\mathbf{atom}) \quad \frac{\nabla \vdash a \,\#\, t}{\nabla \vdash a \,\#\, f\, t}\,(\#\,\mathbf{app}) \quad \frac{}{\nabla \vdash a \,\#\, [a]t}\,(\#\,\mathbf{a[a]})$$

$$\frac{\nabla \vdash a \,\#\, t}{\nabla \vdash a \,\#\, [b]t}\,(\#\,\mathbf{a[b]}) \quad \frac{(\pi^{-1} \cdot a \,\#\, X) \in \nabla}{\nabla \vdash a \,\#\, \pi.X}\,(\#\,\mathbf{var}) \quad \frac{\nabla \vdash a \,\#\, s \quad \nabla \vdash a \,\#\, t}{\nabla \vdash a \,\#\, \langle s, t\rangle}\,(\#\,\mathbf{pair})$$

Figure 2.5: Rules for the freshness relation

Example 2.9. *The judgement* $\nabla \vdash a \,\#\, \langle [a]\langle \overline{a}, \overline{b}\rangle, \pi.X\rangle$ *and* $\nabla \vdash a \,\#\, [b]\langle f\, \pi.X,\ \overline{b}\rangle$ *can be derived only if the pair* $(\pi^{-1} \cdot a)\#X$ *is in the context* ∇*. The derivation trees that justifies these assertions are given below:*

$$\frac{\dfrac{}{\nabla \vdash a \,\#\, [a]\langle \overline{a}, \overline{b}\rangle}\,(\#\,\mathbf{a[a]}) \qquad \dfrac{(\pi^{-1} \cdot a)\#X \in \nabla}{\nabla \vdash a \,\#\, \pi.X}\,(\#\,\mathbf{var})}{\nabla \vdash a \,\#\, \langle [a]\langle \overline{a}, \overline{b}\rangle, \pi.X\rangle}\,(\#\,\mathbf{pair})$$

$$\frac{\dfrac{\dfrac{\dfrac{(\pi^{-1} \cdot a)\#X \in \nabla}{\nabla \vdash a \,\#\, \pi.X}\,(\#\,\mathbf{var})}{\nabla \vdash a \,\#\, f\, \pi.X}\,(\#\,\mathbf{app}) \qquad \dfrac{}{\nabla \vdash a \,\#\, \overline{b}}\,(\#\,\mathbf{atom})}{\nabla \vdash a \,\#\, \langle f\, \pi.X,\ \overline{b}\rangle}\,(\#\,\mathbf{pair})}{\nabla \vdash a \,\#\, [b]\langle f\, \pi.X,\ \overline{b}\rangle}\,(\#\,\mathbf{a[b]})$$

Example 2.10. *The judgement* $\nabla \vdash [a]\langle [a]f\,\overline{a}, \pi.X\rangle \approx_\alpha [b]\langle [b]f\,\overline{b}, \pi'.X\rangle$ *is derived if* $ds(\pi, \pi' \oplus (a\, b))\#X$ *is subset of* ∇ *and* $(\pi'^{-1} \cdot a)\#X \in \nabla$*. The derivation tree of this judgement is given below:*

$$\frac{\dfrac{\dfrac{\dfrac{\dfrac{}{\nabla \vdash \overline{a} \approx_\alpha \overline{a}}\,(\approx_\alpha \mathbf{atom})}{\nabla \vdash f\,\overline{a} \approx_\alpha f\,\overline{a}}\,(\approx_\alpha \mathbf{app})}{\nabla \vdash [a]f\,\overline{a} \approx_\alpha [a]f\,\overline{a}}\,(\approx_\alpha [\mathbf{aa}]) \qquad \dfrac{ds(\pi, \pi' \oplus (a\, b))\#X \subseteq \nabla}{\nabla \vdash \pi.X \approx_\alpha (\pi' \oplus (a\, b)).X}\,(\approx_\alpha \mathbf{var})}{\nabla \vdash \langle [a]f\,\overline{a}, \pi.X\rangle \approx_\alpha \langle [a]f\,\overline{a}, (\pi' \oplus (a\, b)).X\rangle}\,(\approx_\alpha \mathbf{pair}) \qquad \bigtriangledown_1}{\nabla \vdash [a]\langle [a]f\,\overline{a}, \pi.X\rangle \approx_\alpha [b]\langle [b]f\,\overline{b}, \pi'.X\rangle}\,(\approx_\alpha \mathbf{a[b]})$$

where:

$$\bigtriangledown_1 = \frac{\dfrac{\dfrac{\dfrac{}{\nabla \vdash a \,\#\, \overline{b}}\,(\#\,\mathbf{atom})}{\nabla \vdash a \,\#\, f\,\overline{b}}\,(\#\,\mathbf{app})}{\nabla \vdash a \,\#\, [b]f\,\overline{b}}\,(\#\,\mathbf{a[b]}) \qquad \dfrac{(\pi'^{-1} \cdot a)\#X \in \nabla}{\nabla \vdash a \,\#\, \pi'.X}\,(\#\,\mathbf{var})}{\nabla \vdash a \,\#\, \langle [b]f\,\overline{b}, \pi'.X\rangle}\,(\#\,\mathbf{pair})$$

2.2.2 A rule-based nominal unification algorithm

This subsection presents a version of the rule-based nominal unification algorithm proposed by Urban, Pitts and Gabbay [72].

Definition 2.22 (Nominal constraints)**.** *The set* nominal constraints *is composed by equations $s \approx_? t$ and* freshness constraints *of the form $a \#_? t$, where $a \in \mathcal{A}$ and s and t are terms.*

Definition 2.23 (Nominal triple)**.** *A* nominal triple *is an element of the form $\langle \nabla, \sigma, P \rangle$, where ∇ is a freshness context, σ is a substitution and P is finite set of nominal constraints.*

The nominal unification algorithm is defined through the two set of rules of Figs. 2.6 and 2.7 that simplify, respectively, equations and freshness constraints. From now, calligraphic uppercase letters (e.g., $\mathcal{P}, \mathcal{Q}, \mathcal{R}$, etc) will denote nominal triples.

Definition 2.24 (Substitution action over nominal terms)**.** *The* action of a substitution *σ over a nominal term t is defined recursively as:*

$$\begin{array}{lllllllll} \langle\rangle\sigma & := & \langle\rangle & \quad \overline{a}\sigma & := & \overline{a} & \quad (f\,t)\sigma & := & f\,t\sigma \\ \langle s,t\rangle\sigma & := & \langle s\sigma, t\sigma\rangle & \quad ([a]t)\sigma & := & [a]t\sigma & \quad (\pi.X)\sigma & := & \pi \cdot X\sigma \end{array}$$

Along the document, IH stands for induction hypothesis.

Lemma 2.3 (Substitutions and permutations commute)**.** *The actions of permutations and substitutions commute: $(\pi \cdot t)\sigma = \pi \cdot (t\sigma)$*

Proof. The proof is easily done by induction in the structure of t.

- $(\pi \cdot \langle\rangle)\sigma = \langle\rangle = \pi \cdot (\langle\rangle\sigma)$
- $(\pi \cdot \overline{a})\sigma = \overline{b}\sigma = \overline{b} = \pi \cdot \overline{a} = \pi \cdot (\overline{a}\sigma)$
- $(\pi \cdot [a]t)\sigma = [\pi \cdot a]((\pi \cdot t)\sigma) =_{IH} [\pi \cdot a](\pi \cdot (t\sigma)) = \pi \cdot (([a]t)\sigma)$
- $(\pi \cdot \langle s,t\rangle)\sigma = \langle(\pi \cdot s)\sigma, (\pi \cdot t)\sigma\rangle =_{IH} \langle \pi \cdot (s\sigma), \pi \cdot (t\sigma)\rangle = \pi \cdot (\langle s,t\rangle\sigma)$
- $(\pi \cdot (\pi'.X))\sigma = \pi' \oplus \pi \cdot (X\sigma) = \pi \cdot (\pi' \cdot (X\sigma)) = \pi \cdot ((\pi'.X)\sigma)$

□

Notation 2.6. *Let ∇ and ∇' be freshness contexts and σ and σ' be substitutions.*

- *$\nabla' \vdash \nabla\sigma$ denotes that $\nabla' \vdash a \# X\sigma$ holds for each $(a\#X) \in \nabla$, and*
- *$\nabla \vdash \sigma \approx \sigma'$ that $\nabla \vdash X\sigma \approx_\alpha X\sigma'$ for all X in $dom(\sigma) \cup dom(\sigma')$.*

Definition 2.25 (Solution for a nominal triple)**.** *A* solution for a nominal triple $\mathcal{P} = \langle \Delta, \delta, P \rangle$ *is a pair of the form* $\langle \nabla, \sigma \rangle$*, where the following conditions are satisfied:*

i) $\nabla \vdash \Delta\sigma$*;*

ii) if $a \#_? t \in P$ *then* $\nabla \vdash a \# t\sigma$*;*

iii) if $s \approx_? t \in P$ *then* $\nabla \vdash s\sigma \approx_\alpha t\sigma$*;*

iv) there exists λ *such that* $\nabla \vdash \delta\lambda \approx \sigma$*.*

$\mathcal{U}(\mathcal{P})$ *denotes the* solution set of a triple $\mathcal{P}$*. The solution set of a finite set of nominal constraints* P *is defined as* $\mathcal{U}(\langle \emptyset, id, P \rangle)$*.*

Definition 2.26 (More general solution and complete set of solutions)**.** *For* $\langle \nabla, \sigma \rangle$ *and* $\langle \nabla', \sigma' \rangle$ *in* $\mathcal{U}(\mathcal{P})$*,* $\langle \nabla, \sigma \rangle$ *is called* more general *than* $\langle \nabla', \sigma' \rangle$*, denoted* $\langle \nabla, \sigma \rangle \preccurlyeq \langle \nabla', \sigma' \rangle$*, if there exists a substitution* λ *satisfying* $\nabla' \vdash \sigma\lambda \approx \sigma'$ *and* $\nabla' \vdash \nabla\lambda$*. A subset* $\mathcal{S}$ *of* $\mathcal{U}(\mathcal{P})$ *is said to be a* complete set of solutions *of* $\mathcal{P}$ *if for all* $\langle \nabla', \sigma' \rangle \in \mathcal{U}(\mathcal{P})$*, there exists* $\langle \nabla, \sigma \rangle$ *in* $\mathcal{S}$ *such that* $\langle \nabla, \sigma \rangle \preccurlyeq \langle \nabla', \sigma' \rangle$*.*

Definition 2.27 (Set of variables of a set of nominal constraints)**.** *The set of variables occurring in a set of nominal constraints* P *is denoted by* $Var(P)$ *and defined as* $(\bigcup_{s \approx_? t \in P} Var(s) \cup Var(t)) \cup \bigcup_{a \#_? u \in P} Var(u)$*.*

Definition 2.28 (Variables of a nominal triple)**.** *The set of variables of* $\mathcal{P} = \langle \nabla, \sigma, P \rangle$ *is defined as* $Var(P)$ *and denoted by* $Var(\mathcal{P})$*.*

Definition 2.29 (Size of a set of nominal constraints)**.** *The* size of a set of nominal constraints *is denoted by* $|P|$ *and defined as* $(\sum_{s \approx_? t \in P} |s| + |t|) + \sum_{a \#_? u \in P} |u|$*.*

In the following rules of systems of Figs. 2.6 and 2.7, the symbols $\uplus$ and $\cup$ stand, respectively, for the disjoint and the standard union operation.

Remark 2.6. *Notice that, in rule* ($\approx_?$ **inst**) *of Fig. 2.6, the substitution* $\sigma\{X/\pi^{-1} \cdot t\}$ *is the result of the composition of* σ *with* $\{X/\pi^{-1} \cdot t\}$ *(cf. Def. 2.7).*

Notation 2.7.

- $\mathcal{P} \Rightarrow_\approx \mathcal{Q}$ *(resp.* $\mathcal{P} \Rightarrow_\# \mathcal{Q}$*) denotes that* $\mathcal{P}$ *is related to* $\mathcal{Q}$ *by the system of Fig. 2.6 (resp. Fig. 2.7);*

- *Let* P *be a set of nominal constraints,* $P_\approx$ *and* $P_\#$ *will, respectively, denote the sets of equations and freshness constraints in the set* P*.*

$$\frac{\langle \nabla, \sigma, P \uplus \{s \approx_? s\}\rangle}{\langle \nabla, \sigma, P\rangle} \ (\approx_? \textbf{refl}) \qquad \frac{\langle \nabla, \sigma, P \uplus \{\langle s_0, t_0\rangle \approx_? \langle s_1, t_1\rangle\}\rangle}{\langle \nabla, \sigma, P \cup \{s_0 \approx_? s_1, t_0 \approx_? t_1\}\rangle} \ (\approx_? \textbf{pair})$$

$$\frac{\langle \nabla, \sigma, P \uplus \{\pi.X \approx_? \pi'.X\}\rangle}{\langle ds(\pi, \pi')\#X \cup \nabla, \sigma, P\}\rangle} \ (\approx_? \textbf{var}) \qquad \frac{\langle \nabla, \sigma, P \uplus \{f\, s \approx_? f\, t\}\rangle}{\langle \nabla, \sigma, P \cup \{s \approx_? t\}\rangle} \ (\approx_? \textbf{app})$$

$$\frac{\langle \nabla, \sigma, P \uplus \{[a]s \approx_? [a]t\}\rangle}{\langle \nabla, \sigma, P \cup \{s \approx_? t\}\rangle} \ (\approx_? [\textbf{aa}]) \qquad \frac{\langle \nabla, \sigma, P \uplus \{[a]s \approx_? [b]t\}\rangle}{\langle \nabla, \sigma, P \cup \{s \approx_? (a\, b)\cdot t, a \#_? t\}\rangle} \ (\approx_? [\textbf{ab}])$$

$$\frac{\langle \nabla, \sigma, P \uplus \{\pi.X \approx_? t\} \text{ or } \{t \approx_? \pi.X\}\rangle}{\langle \nabla, \sigma\{X/\pi^{-1}\cdot t\}, P\{X/\pi^{-1}\cdot t\}\rangle} \ X \notin \mathtt{var}(t)\ (\approx_? \textbf{inst})$$

Figure 2.6: Reduction rules for equations

$$\frac{\langle \nabla, \sigma, P \uplus \{a \#_? \langle\rangle\}\rangle}{\langle \nabla, \sigma, P\rangle} \ (\#_? \langle\rangle) \qquad \frac{\langle \nabla, \sigma, P \uplus \{a \#_? \bar{b}\}\rangle}{\langle \nabla, \sigma, P\rangle} \ (\#_? \mathbf{a\bar{b}})$$

$$\frac{\langle\langle \nabla, \sigma, P \uplus \{a \#_? f\, t\}\rangle}{\langle \nabla, \sigma, P \cup \{a \#_? t\}\rangle} \ (\#_? \textbf{app}) \qquad \frac{\langle \nabla, \sigma, P \uplus \{a \#_? [a]t\}\rangle}{\langle \nabla, \sigma, P\rangle} \ (\#_? \mathbf{a}[\mathbf{a}])$$

$$\frac{\langle \nabla, \sigma, P \uplus \{a \#_? [b]t\}\rangle}{\langle \nabla, \sigma, P \cup \{a \#_? t\}\rangle} \ (\#_? \mathbf{a}[\mathbf{b}]) \qquad \frac{\langle \nabla, \sigma, P \uplus \{a \#_? \pi.X\}\rangle}{\langle\{(\pi^{-1}\cdot a)\#X\} \cup \nabla, \sigma, P\rangle} \ (\#_? \textbf{var})$$

$$\frac{\langle \nabla, \sigma, P \uplus \{a \#_? \langle s, t\rangle\}\rangle}{\langle \nabla, \sigma, P \cup \{a \#_? s, a \#_? t\}\rangle} \ (\#_? \textbf{pair})$$

Figure 2.7: Reduction rules for freshness constraints

Lemma 2.4 (Termination of $\Rightarrow_\approx$ and $\Rightarrow_\#$)**.** *Both relations $\Rightarrow_\approx$ and $\Rightarrow_\#$ are terminating.*

Proof. The proof is by well-founded induction on the lexicographic measure $\langle |Var(P_\approx)|, |P_\approx|, |P_\#|\rangle$ over triples $\langle \nabla, \sigma, P\rangle$.

- If $P \Rightarrow_\approx Q$ by rule ($\approx_?$ **inst**), then $|Var(P_\approx)| > |Var(Q_\approx)|$. In all other cases of $P \Rightarrow_\approx Q$, one has $|Var(P_\approx)| \geq |Var(Q_\approx)|$ and $|P_\approx| > |Q_\approx|$;

- If $P \Rightarrow_\# Q$, $|Var(P_\approx)| = |Var(Q_\approx)|$, $|P_\approx| = |Q_\approx|$ and $|P_\#| > |Q_\#|$.

$\square$

Notation 2.8 (Standard rewriting notation)**.** *The standard rewriting nomenclature of Not. 2.3 is extended to nominal triples (Def. 2.23):*

- $\mathcal{P}$ *is called a* normal form *or* irreducible *by* $\Rightarrow_{\approx}$ *(resp.* $\Rightarrow_{\#}$*), denoted by* $\Rightarrow_{\approx}$-nf *(resp.* $\Rightarrow_{\#}$*-nf), whenever there is no* $\mathcal{Q}$ *such that* $\mathcal{P} \Rightarrow_{\approx} \mathcal{Q}$ *(*$\mathcal{P} \Rightarrow_{\#} \mathcal{Q}$*);* $\Rightarrow^{*}_{\approx}$ *and* $\Rightarrow^{+}_{\approx}$ *(resp.* $\Rightarrow^{*}_{\#}$ *and* $\Rightarrow^{+}_{\#}$*) denote respectively derivations in zero or more, and one or more applications of the rules.*

Definition 2.30 (Reduction strategy for $\Rightarrow_{\approx}$ and $\Rightarrow_{\#}$)**.** *For a unification problem P the algorithm defined using rules of Figs. 2.6 and 2.7 proceeds by the following* reduction strategy*:*

1. *The input problem P is converted into the nominal triple* $\mathcal{P} = \langle \emptyset, id, P \rangle$*;*
2. $\langle \emptyset, id, P \rangle$ *is reduced by* $\Rightarrow_{\approx}$ *until reaching a normal form* $\mathcal{Q} = \langle \emptyset, \sigma, Q \rangle$*.*
   If $Q_{\approx} \neq \emptyset$*, the algorithm outputs* $\perp$ *(fail);*
3. *If* $Q_{\approx} = \emptyset$*, then* $\mathcal{Q} \Rightarrow^{*}_{\#} \mathcal{R}$*, where* $\mathcal{R} = \langle \nabla, \sigma, R \rangle$ *is a* $\Rightarrow_{\#}$*-nf.*
   If $R \neq \emptyset$*, the algorithm also outputs* $\perp$ *(fail), otherwise the pair* $\langle \nabla, \sigma \rangle$ *is the output of the algorithm.*

Exs. 2.11 to 2.13 present the simplification steps applying the strategy of Def. 2.30 to a problem.

Example 2.11. *The algorithm applied to* $P = \{[a]\langle X, f\,\overline{a}\rangle \approx_? [b]\langle\langle Z, Z\rangle, Z\rangle\}$ *outputs* $\langle \emptyset, \{X/\langle f\,\overline{a}, f\,\overline{a}\rangle, Z/f\,\overline{b}\}\rangle$*:*

$$\begin{array}{rl}
\mathcal{P} = & \langle \emptyset, id, \{\, [a]\langle X, f\,\overline{a}\rangle \approx_? [b]\langle\langle Z, Z\rangle, Z\rangle \,\}\rangle \\
\Rightarrow_{(\approx_?\mathbf{[ab]})} & \langle \emptyset, id, \{\, \langle X, f\,\overline{a}\rangle \approx_? \langle\langle (a\,b).Z, (a\,b).Z\rangle, (a\,b).Z\rangle\, , a \#_? \langle\langle Z, Z\rangle, Z\rangle\}\rangle \\
\Rightarrow_{(\approx_?\mathbf{pair})} & \langle \emptyset, id, \{\, X \approx_? \langle (a\,b).Z, (a\,b).Z\rangle\, , f\,\overline{a} \approx_? (a\,b).Z, a \#_? \langle\langle Z, Z\rangle, Z\rangle\}\rangle \\
\Rightarrow_{(\approx_?\mathbf{inst})} & \langle \emptyset, \{X/\langle (a\,b).Z, (a\,b).Z\rangle\}, \{\, f\,\overline{a} \approx_? (a\,b).Z\, , a \#_? \langle\langle Z, Z\rangle, Z\rangle\}\rangle \\
\Rightarrow_{(\approx_?\mathbf{inst})} & \langle \emptyset, \{X/\langle f\,\overline{a}, f\,\overline{a}\rangle, Z/f\,\overline{b}\}, \{\, a \#_? \langle\langle f\,\overline{b}, f\,\overline{b}\rangle, f\,\overline{b}\rangle \,\}\rangle \\
\Rightarrow_{(\#_?\,\mathbf{pair})_{(2\times)}} & \langle \emptyset, \{X/\langle f\,\overline{a}, f\,\overline{a}\rangle, Z/f\,\overline{b}\}, \{\, a \#_? f\,\overline{b} \,\}\rangle \\
\Rightarrow_{(\#_?\,\mathbf{app})} & \langle \{\emptyset\}, \{X/\langle f\,\overline{a}, f\,\overline{a}\rangle, Z/f\,\overline{b}\}, \{\, a \#_? \overline{b} \,\}\rangle \\
\Rightarrow_{(\#_?\,\mathbf{a\overline{b}})} & \langle \{\emptyset\}, \{X/\langle f\,\overline{a}, f\,\overline{a}\rangle, Z/f\,\overline{b}\}, \emptyset\rangle
\end{array}$$

Notice that $\langle\emptyset, \{X/\langle f\,\overline{a}, f\,\overline{a}\rangle, Z/f\,\overline{b}\}\rangle$ *is a solution for* P *since* $\emptyset \vdash [a]\langle\langle f\,\overline{a}, f\,\overline{a}\rangle, f\,\overline{a}\rangle \approx_\alpha [b]\langle\langle f\,\overline{b}, f\,\overline{b}\rangle, f\,\overline{b}\rangle$.

Example 2.12. *The algorithm applied to* $P = \{f\,[a]\langle X, \langle\overline{a}, Y\rangle\rangle \approx_? f\,[b]\langle\overline{b}, \langle\overline{b}, Z\rangle\rangle\}$ *outputs* $\langle a\#Z, \{X/\overline{a}, Y/(a\,b).Z\}, \emptyset\rangle$*:*

$$\begin{array}{rl}
\mathcal{P} = & \langle\emptyset, id, \{\, f\,[a]\langle X, \langle\overline{a}, Y\rangle\rangle \approx_? f\,[b]\langle\overline{b}, \langle\overline{b}, Z\rangle\rangle \,\}\rangle \\
\Rightarrow_{(\approx_?\mathbf{app})} & \langle\emptyset, id, \{\, [a]\langle X, \langle\overline{a}, Y\rangle\rangle \approx_? [b]\langle\overline{b}, \langle\overline{b}, Z\rangle\rangle \,\}\rangle \\
\Rightarrow_{(\approx_?[\mathbf{ab}])} & \langle\emptyset, id, \{\, \langle X, \langle\overline{a}, Y\rangle\rangle \approx_? \langle\overline{a}, \langle\overline{a}, (a\,b).Z\rangle\rangle,\ a \#_? \langle\overline{b}, \langle\overline{b}, Z\rangle\rangle\}\rangle \\
\Rightarrow_{(\approx_?\mathbf{pair})} & \langle\emptyset, id, \{\, X \approx_? \overline{a},\ \langle\overline{a}, Y\rangle \approx_? \langle\overline{a}, (a\,b).Z\rangle,\ a \#_? \langle\overline{b}, \langle\overline{b}, Z\rangle\rangle\}\rangle \\
\Rightarrow_{(\approx_?\mathbf{inst})} & \langle\emptyset, \{X/\overline{a}\}, \{\, \langle\overline{a}, Y\rangle \approx_? \langle\overline{a}, (a\,b).Z\rangle,\ a \#_? \langle\overline{b}, \langle\overline{b}, Z\rangle\rangle\}\rangle \\
\Rightarrow_{(\approx_?\mathbf{pair})} & \langle\emptyset, \{X/\overline{a}\}, \{\, \overline{a} \approx_? \overline{a},\ Y \approx_? (a\,b).Z,\ a \#_? \langle\overline{b}, \langle\overline{b}, Z\rangle\rangle\}\rangle \\
\Rightarrow_{(\approx_?\mathbf{refl})} & \langle\emptyset, \{X/\overline{a}\}, \{\, Y \approx_? (a\,b).Z,\ a \#_? \langle\overline{b}, \langle\overline{b}, Z\rangle\rangle\}\rangle \\
\Rightarrow_{(\approx_?\mathbf{inst})} & \langle\emptyset, \{X/\overline{a}, Y/(a\,b).Z\}, \{\, a \#_? \langle\overline{b}, \langle\overline{b}, Z\rangle\rangle \,\}\rangle \\
\Rightarrow_{(\#_?\,\mathbf{pair})} & \langle\emptyset, \{X/\overline{a}, Y/(a\,b).Z\}, \{\, a \#_? \overline{b},\ a \#_? Z\}\rangle \\
\Rightarrow_{(\#_?\,\mathbf{a\overline{b}})} & \langle\emptyset, \{X/\overline{a}, Y/(a\,b).Z\}, \{\, a \#_? Z \,\}\rangle \\
\Rightarrow_{(\#_?\,\mathbf{var})} & \langle\{a\#Z\}, \{X/\overline{a}, Y/(a\,b).Z\}, \emptyset\rangle
\end{array}$$

Notice that $\langle\{a\#Z\}, \{X/\overline{a}, Y/(a\,b).Z\}, \emptyset\rangle$ *is a solution for* P *since* $\{a\#Z\} \vdash f\,[a]\langle\overline{a}, \langle\overline{a}, (a\,b).Z\rangle\rangle \approx_\alpha f\,[b]\langle\overline{b}, \langle\overline{b}, Z\rangle\rangle\}$.

Example 2.13. *The algorithm applied to either a)* $P = \{[a]\langle X, Y\rangle \approx_? [b]\langle\overline{a}, \overline{b}\rangle\}$ *or b)* $P = \{[a]\langle X, \overline{b}\rangle \approx_? [b]\langle\overline{b}, \overline{c}\rangle\}$ *outputs* $\perp$*. Observe that* P *has no solutions in both cases:*

a)

$$\begin{array}{rl}
\mathcal{P} = & \langle\emptyset, id, \{\, [a]\langle X, Y\rangle \approx_? [b]\langle\overline{a}, \overline{b}\rangle \,\}\rangle \\
\Rightarrow_{(\approx_?[\mathbf{ab}])} & \langle\emptyset, id, \{\, \langle X, Y\rangle \approx_? \langle\overline{b}, \overline{a}\rangle,\ a \#_? \langle\overline{a}, \overline{b}\rangle\}\rangle \\
\Rightarrow_{(\approx_?\mathbf{pair})} & \langle\emptyset, id, \{\, X \approx_? \overline{b},\ Y \approx_? \overline{a},\ a \#_? \langle\overline{a}, \overline{b}\rangle\}\rangle \\
\Rightarrow_{(\approx_?\mathbf{inst})} & \langle\emptyset, \{X/\overline{b}\}, \{\, Y \approx_? \overline{a},\ a \#_? \langle\overline{a}, \overline{b}\rangle\}\rangle \\
\Rightarrow_{(\approx_?\mathbf{inst})} & \langle\emptyset, \{X/\overline{b}, Y/\overline{a}\}, \{\, a \#_? \langle\overline{a}, \overline{b}\rangle \,\}\rangle \\
\Rightarrow_{(\#_?\,\mathbf{pair})} & \langle\emptyset, \{X/\overline{b}, Y/\overline{a}\}, \{a \#_? \overline{a},\ a \#_? \overline{b}\,\}\rangle \\
\Rightarrow_{(\#_?\,\mathbf{a\overline{b}})} & \langle\emptyset, \{X/\overline{b}, Y/\overline{a}\}, \{a \#_? \overline{a}\}\rangle.
\end{array}$$

Since this is a normal form and the set of nominal constraints is non-empty the algorithm outputs $\perp$*.*

b)

$$\begin{array}{rl} \mathcal{P} = & \langle \emptyset, id, \{ \boxed{[a]\langle X, \overline{b}\rangle \approx_? [b]\langle \overline{b}, \overline{c}\rangle} \}\rangle \\ \Rightarrow_{(\approx_?[\mathbf{ab}])} & \langle \emptyset, id, \{ \boxed{\langle X, \overline{b}\rangle \approx_? \langle \overline{a}, \overline{c}\rangle},\ a \#_? \langle \overline{b}, \overline{c}\rangle\}\rangle \\ \Rightarrow_{(\approx_?\mathbf{pair})} & \langle \emptyset, id, \{ \boxed{X \approx_? \overline{a}},\ b \approx_? \overline{c},\ a \#_? \langle \overline{b}, \overline{c}\rangle\}\rangle \\ \Rightarrow_{(\approx_?\mathbf{inst})} & \langle \emptyset, \{X/\overline{a}\}, \{\overline{b} \approx_? \overline{c},\ a \#_? \langle \overline{b}, \overline{c}\rangle\}\rangle. \end{array}$$

Since this is a $\Rightarrow_{\#}$-normal form and the set of equations is non empty the algorithm outputs $\perp$.

Nominal unification has type 1 and it is quadratically bounded

[72] presented a proof that the rules of Figs. 2.6 and 2.7 are sound and complete, and the set of generated solutions is at most unitary. Thus nominal unification has type 1. These proofs of soundness and completeness of unification algorithm, given by the strategy, are contained in the proofs of Chap. 5 for nominal unification modulo C. Then these proofs are omitted.

As explained in the related work section (Sec. 1.1), the works of [25] and [48] demonstrated that nominal unification is quadratically bounded. It is a open question if it is possible to build a nominal unification algorithm that is executed in subquadratic time (and space).

Chapter 3

Specification and Formalisation in Coq: the case of nominal α equality-checking

In this Chapter, a short introduction of the use of Coq is provided. The Coq proof assistant [32] is a system designed for specification and formal verification of programs and mathematical proofs. It is based on the so-called *Calculus of Inductive Constructions*, the λ-calculus with a rich type system, and uses a specification language named *Gallina*. Based on the Curry-Howard isomorphism, Coq establishes a correspondence between logical statements and type checking judgements. Terms of Gallina allow representing logical statements, programs, properties of these programs, and proofs of the logical statements and properties. The proofs are constructed through the use of an interactive system with the application of *tactics*. Tactics are sets of instructions that are pre-specified in the system using another language denoted by $\mathtt{L_{tac}}$.

As a use case of Coq, it is presented the formalisation of soundness of $\approx_\alpha$ using two different approaches. The first approach (presented in Sec. 3.1) was originally formalised in Isabelle/HOL by Urban [70] as an improvement of a previous Isabelle/HOL formalisation given by [72]. This improvement introduced the use of an auxiliary weak-equivalence relation $\sim_\omega$. The second approach (presented in Sec. 3.2) is part of the development in PVS of nominal unification given in [11] (also in Chap. 3 of [61]). The latter is straightforward and more elegant, which is checked in Sec. 3.3 by a comparison between the two formalisation approaches. The following Lems. 3.1 to 3.3 are used in both approaches.

In the Coq specification, file `Terms.v`, the grammar is specified as in Fig. 3.1. Atoms and variables are inductively defined through a mapping to Coq naturals, and permutations (resp. freshness contexts) are defined as lists of pairs of atoms (resp. pairs of

atoms and variables). Constructors `Ut, At, Ab, Pr, Fc` and `Su` specify the empty tuple, atoms as term objects, abstractions, pairs, function applications and suspended variables, respectively. For the `Fc` constructor, the first and second `nat` arguments correspond to super and subscripts of the applied function symbol. These scripts are, respectively, used to distinguish the equational properties of the function symbol and its indexation between the class of operators with the same equational properties. In the formalisation, the function symbols f_j^A, f_k^{AC} and f_n^C are given respectively by `Fc 0 j`, `Fc 1 k` and `Fc 2 n`, all having type `term` → `term`. Indeed, superscripts 0, 1 and 2 are used to specify A, AC and C function symbols, respectively, and all other superscripts are representing syntactic function symbols. Function scripts will be omitted when no confusion arises. In the specification of terms, `Notation` declarations are used to provide friendly representations for the term constructors.

```
Inductive Atom : Set := atom : nat → Atom.
Inductive Var : Set := var : nat → Var.
Definition Perm := list (Atom × Atom).
Definition Context := list (Atom × Var).
```

```
Inductive term : Set :=
  | Ut : term                                  Notation «» := (Ut).
  | At : Atom → term                           Notation %a := (At a).
  | Ab : Atom → term → term                    Notation [a]^t := (Ab a t).
  | Pr : term → term → term                    Notation <|t1,t2|> := (Pr t1 t2).
  | Fc : nat → nat → term → term               Notation pi|.X := (Su pi X).
  | Su : Perm → Var → term
```

Figure 3.1: Coq specification of the grammar of nominal terms

Although in nominal syntax different atoms a and b are assumed to be different, this is not automatically true in computational specifications. Indeed, since the given approach uses variables ranging over naturals to represent atoms, different variables might represent the same atom. Then, rules ($\approx_\alpha$ [**ab**]) and (# **a**[**b**]), respectively, from Figs. 2.4 and 2.5, were specified with the extra condition $a \neq b$.

In the Coq specification, the definition of action of permutations and its properties are formalised in file `Perm.v` as Defs. 2.19 and 2.20. The action of a permutation over an atomic term object $\overline{a}$, e.g., $\pi \cdot \overline{a}$, gives as result a term $\overline{\pi \cdot a}$.

The sets of derivable freshness and α-equivalence judgements (by rules of Figs. 2.4 and 2.5) are specified as the inductive definitions **fresh** and **alpha_equiv** respectively of Figs. 3.2 and 3.3. These definitions are included, respectively, in the files `Fresh.v` and `Alpha_Equiv.v`.

In the **fresh** definition, the interesting rule is fresh_Su. Notice that, in this rule, the judgement **fresh** C a (Su $p\,X$), which means that a is fresh for the term Su $p\,X$ under the

context C, is derivable only if the pair `((!`p `$` a`),` X) is in the freshness context C, where `!`p stands for the reverse list of permutation p. For the definition **alpha_equiv**, the interesting rules are alpha_equiv_Ab_2 and alpha_equiv_Su. In the former, the judgement **alpha_equiv** C (`[`a`]^`t) (`[`a'`]^`t') is derivable if tree conditions are satisfied:

i) $a \neq a$'. Atoms a and a' are different;

ii) **alpha_equiv** C t (`|[`(a,a')`]|` `@` t'), which means that t is α-equivalent to `|[`(a,a')`]|` `@` t'. The last is the result of the action of the swapping `|[`(a,a')`]|` over the term t'; and

iii) $C \vdash a$ `#` t', that is a notation for the freshness judgement **fresh** C a t'.

In the rule alpha_equiv_Su, **alpha_equiv** C (p`|.`X) (p'`|.`X) is derivable only if for every atom a that is in the difference set of p and p' (expressed by (In_ds p p' a)) the pair (a, X) is in the freshness context C. The last sentence corresponds to the specification of the condition $ds(\pi, \pi')\# \subseteq \nabla$, in rule ($\approx_\alpha$ **var**) of Fig. 2.4. The specification of the predicate In_ds and the formalisation of its properties are in file `Disagr.v`.

`Inductive` **fresh** : Context → **Atom** →**term** → `Prop` :=

| fresh_Ut : ∀ C a, **fresh** C a Ut

| fresh_Pr : ∀ C a t_1 t_2, (**fresh** C a t_1) → (**fresh** C a t_2) →
(**fresh** C a (`<|`t_1`,`t_2`|>`))

| fresh_Fc : ∀ C a E n t, (**fresh** C a t) → (**fresh** C a (Fc E n t))

| fresh_Ab_1 : ∀ C a t, **fresh** C a (Ab a t)

| fresh_Ab_2 : ∀ C a a' t, $a \neq a$' →
(**fresh** C a t) → (**fresh** C a (Ab a' t))

| fresh_At : ∀ C a a', $a \neq a$' → (**fresh** C a (At a'))

| fresh_Su : ∀ C p a X, set_In (`((!`p `$` a`),` X)) C →
fresh C a (Su p X) .

Figure 3.2: Specification of **fresh** (freshness judgment derivation)

Lemma 3.1 (Freshness preservation of $\approx_\alpha$)**.** *If* $\nabla \vdash a \# t$ *and* $\nabla \vdash t \approx_\alpha t'$ *then* $\nabla \vdash a \# t'$*.*

Proof. This result was formalised in file `Alpha_Equiv.v`, Lem. `alpha_equiv_fresh`. Its proof is obtained by induction on $\approx_\alpha$. The interesting case is the analysis of the ($\approx_\alpha$ [**ab**]) rule, whose hypotheses are $a_0 \neq b_0$, $\nabla \vdash t \approx_\alpha (a_0\, b_0)\, t'$, $\nabla \vdash a_0 \# t'$, and $\nabla \vdash a \# [a_0]t$. By IH $\nabla \vdash a \# t \Rightarrow \nabla \vdash a \# (a_0\, b_0)\, t'$. It should be concluded that $\nabla \vdash a \# [b_0]t'$. For doing that, it is necessary to evaluate three cases: $a = a_0$, $b_0 = a \neq a_0$ and $b_0 \neq a \neq a_0$. The difficult case is the last one, that is solved by application of the ($\#$ **a**[**b**]) rule with the use of a technical lemma about the freshness relation. □

```
Inductive alpha_equiv : Context →term →term → Prop :=
 | alpha_equiv_Ut : ∀ C, alpha_equiv C («») («»)
 | alpha_equiv_At : ∀ C a, alpha_equiv C (%a) (%a)
 | alpha_equiv_Pr : ∀ C t1 t2 t1' t2',
   (alpha_equiv C t1 t1') → (alpha_equiv C t2 t2') →
   alpha_equiv C (<|t1,t2|>) (<|t1',t2'|>)
 | alpha_equiv_Fc : ∀ m n t t' C,
   (alpha_equiv C t t') → alpha_equiv C (Fc m n t) (Fc m n t')
 | alpha_equiv_Ab_1 : ∀ C a t t',
   (alpha_equiv C t t') → alpha_equiv C ([a]^t) ([a]^t')
 | alpha_equiv_Ab_2 : ∀ C a a' t t', a ≠ a' →
   (alpha_equiv C t (|[(a,a')]| @ t')) → C ⊢ a # t' →
   alpha_equiv C ([a]^t) ([a']^t')
 | alpha_equiv_Su : ∀ (C: Context) p p' (X: Var),
   (∀ a, (In_ds p p' a) → set_In ((a,X)) C) →
   alpha_equiv C (p|.X) (p'|.X) .
```

Figure 3.3: Specification of **alpha_equiv** (α-equivalence judgment derivation)

Lemma 3.2 (Invariance of terms under $\approx_\alpha$ and the action of permutations)**.** $(\forall a \in ds(\pi, \pi'), \nabla \vdash a \# t)$ *iff* $\nabla \vdash \pi \cdot t \approx_\alpha \pi' \cdot t$.

Proof. This result is formalised in file `Alpha_Equiv.v`, Lem. `alpha_equiv_pi`, by induction on term t under arbitrary permutations, being the case of abstraction the interesting one. Suppose $t = [a]t'$. The IH is given by:

$$\forall \pi'_0, (\forall a_0 \in ds(\pi, \pi'_0), \nabla \vdash a_0 \# t') \Leftrightarrow \nabla \vdash \pi \cdot t' \approx_\alpha \pi'_0 \cdot t'$$

It should be proved that:

$$(\forall a_0 \in ds(\pi, \pi'), \nabla \vdash a_0 \# [a]t') \Leftrightarrow \nabla \vdash \pi \cdot [a]t' \approx_\alpha \pi' \cdot [a]t'$$

Applying the definition of the permutation action, the *rhs* rewrites to $\nabla \vdash [\pi \cdot a](\pi \cdot t') \approx_\alpha [\pi' \cdot a](\pi' \cdot t')$. The proof relies in analysing four cases according to whether $\pi \cdot a$ and $\pi' \cdot a$ as well as a_0 and a are or not equal. The case $\pi \cdot a \neq \pi' \cdot a$ and $a_0 \neq a$, that is the most difficult one, is explained below.

(⇒) As main premise one has $\forall a_0 \in ds(\pi, \pi'), \nabla \vdash a_0 \# [a]t'$ and should conclude that $\nabla \vdash [\pi \cdot a](\pi \cdot t') \approx_\alpha [\pi' \cdot a](\pi' \cdot t')$. Applying ($\approx_\alpha$ [**ab**]), one should prove two subgoals: $\nabla \vdash \pi \cdot t' \approx_\alpha ((\pi \cdot a)\ (\pi' \cdot a))\,(\pi' \cdot t')$ and $\nabla \vdash (\pi \cdot a) \# (\pi' \cdot t')$. For the

first subgoal one applies IH and it remains to prove that $\nabla \vdash a_0 \# t'$ supposing $a_0 \in ds(\pi, \pi' \oplus [((\pi \cdot a)\ (\pi' \cdot a))])$. Using a previous result about difference sets this rewrites to $a_0 \in ds(\pi, \pi')$. Thus, from the main premise one has $\nabla \vdash a_0 \# [a]t'$ and then, since $a_0 \neq a$, $\nabla \vdash a_0 \# t'$. The second subgoal is proved instantiating a_0 in the main premise with $(\pi')^{-1} \cdot (\pi \cdot a)$, thus obtaining $\nabla \vdash ((\pi')^{-1} \cdot (\pi \cdot a)) \# [a]t'$. By a lemma on freshness, it results in $\nabla \vdash (\pi \cdot a) \# [\pi' \cdot a](\pi' \cdot t')$. Finally, since $\pi \cdot a \neq \pi' \cdot a$, it can be concluded that $\nabla \vdash (\pi \cdot a) \# (\pi' \cdot t')$.

($\Leftarrow$) The main premise is $\nabla \vdash [\pi \cdot a](\pi \cdot t)' \approx_\alpha [\pi' \cdot a](\pi' \cdot t')$ and one has to prove $\forall a_0 \in ds(\pi, \pi'), \nabla \vdash a_0 \# [a]t'$. Inverting the main premise according to rule ($\approx_\alpha$ **[ab]**) one has as hypotheses $\nabla \vdash \pi \cdot t' \approx_\alpha ((\pi \cdot a)\ (\pi' \cdot a))\,(\pi' \cdot t')$ and $\nabla \vdash (\pi \cdot a) \# (\pi' \cdot t')$. Applying ($\approx_\alpha$ **[ab]**) to the goal it remains to show that $\nabla \vdash a_0 \# t'$, and then two cases should be evaluated: $\pi \cdot a = \pi' \cdot a_0$ and $\pi \cdot a \neq \pi' \cdot a_0$. For the first subgoal, one uses the second hypothesis with a technical freshness property to conclude that $\nabla \vdash (\pi' \cdot a_0) \# (\pi' \cdot t') \Rightarrow \nabla \vdash a_0 \# t'$. For the second subgoal, using the first hypothesis and applying IH with the instantiation $\pi'_0 = \pi' \oplus [((\pi \cdot a)\ (\pi' \cdot a))]$ it just remains to prove that $a_0 \in ds(\pi, \pi' \oplus [((\pi \cdot a)\ (\pi' \cdot a))])$ assuming $a_0 \in ds(\pi, \pi')$. But because $\pi \cdot a \neq \pi' \cdot a_0$, it is a trivial application of a previous result about difference sets.

□

Remark 3.1. *Lem. 3.2 is applied in Sec. 3.1 in the proof of Lem. 3.9 and in both Secs. 3.1 and 3.2 in proofs related with symmetry, as preservation of α-equivalence under swappings, e.g.,* $\nabla \vdash (a\,b) \cdot t \approx_\alpha (c\,d) \cdot t$ *with* $\nabla \vdash a, b, c, d \# t$.

Lemma 3.3 (Reflexivity of $\approx_\alpha$)**.** $\nabla \vdash t \approx_\alpha t$

Proof. Reflexivity of $\approx_\alpha$ is formalised in file `Alpha_Equiv.v`, Lem. `alpha_equiv_refl`, by induction on the structure of t. All the cases are trivial, and are solved by simple applications of the inference rules of $\approx_\alpha$. □

3.1 Soundness of $\approx_\alpha$ using a *weak* α-equivalence $\sim_\omega$

This section describes a formalisation of the soundness of $\approx_\alpha$, using the so-called "weak" α-equivalence relation $\sim_\omega$ defined and specified, respectively, in Figs. 3.4 and 3.5. First, $\sim_\omega$ is proved be an equivalence relation, which is straightforward and gives an intermediate transitivity result for $\approx_\alpha$: $\nabla \vdash t_1 \approx_\alpha t_2$ and $t_2 \sim_\omega t_3$ implies $\nabla \vdash t_1 \approx_\alpha t_3$. Second, this result is used in conjunction with some auxiliary lemmas to prove first the transitivity, and then the symmetry of $\approx_\alpha$.

$$\frac{}{\langle\rangle \sim_\omega \langle\rangle}\ (\sim_\omega \langle\rangle) \qquad \frac{}{\overline{a} \sim_\omega \overline{a}}\ (\sim_\omega \mathbf{atom}) \qquad \frac{\nabla \vdash s \approx_\alpha t}{f\, s \sim_\omega f\, t}\ (\sim_\omega \mathbf{app})$$

$$\frac{s_0 \sim_\omega t_0 \quad s_1 \sim_\omega t_1}{\langle s_0, s_1\rangle \sim_\omega \langle t_0, t_1\rangle}\ (\sim_\omega \mathbf{pair}) \qquad \frac{s \sim_\omega t}{[a]s \sim_\omega [a]t}\ (\sim_\omega [\mathbf{aa}]) \qquad \frac{ds(\pi, \pi') = \emptyset}{\pi.X \sim_\omega \pi'.X}\ (\sim_\omega \mathbf{var})$$

Figure 3.4: Rules for weak α-equivalence $\sim_\omega$

Lems. 3.4, 3.5 and 3.6 are properties of $\sim_\omega$ that are formalised in file `w_Equiv.v` of the specification. These lemmas are used in Lem. 3.7, which relates $\approx_\alpha$ with $\sim_\omega$, in proofs of Lem. 3.9, and Lems. 3.10 and 3.11 on the transitivity and the symmetry of $\approx_\alpha$. All the proofs related strictly to $\sim_\omega$ are straightforward and their details can be found in the formalisation.

Notice that in rule w_equiv_Su of Fig 3.5, the expression (∀ *a*, ¬ (In_ds *p p' a*)) denotes the condition $ds(\pi, \pi') = \emptyset$ of rule ($\sim_\omega$ **var**).

Lemma 3.4 (Equivalence of $\sim_\omega$)**.** *$\sim_\omega$ is an equivalence relation.*

Proof. Reflexivity is proved by induction on the structure of terms; symmetry and transitivity are proved by induction in the derivation rules of $\sim_\omega$. □

Lemma 3.5 (Equivariance of $\sim_\omega$)**.** *If $t \sim_\omega t'$ then $\pi \cdot t \sim_\omega \pi \cdot t'$.*

Proof. The proof is by induction on the derivation rules of $\sim_\omega$ in the sentence $t \sim_\omega t'$. □

Lemma 3.6 (Freshness preservation of $\sim_\omega$)**.** *If $\nabla \vdash a \# t$ and $t \sim_\omega t'$ then $\nabla \vdash a \# t'$.*

Proof. The proof is by induction on the derivation rules of $\sim_\omega$ in the sentence $t \sim_\omega t'$. □

Lemma 3.7 (Intermediate transitivity for $\approx_\alpha$ with $\sim_\omega$)**.** *If $\nabla \vdash t_1 \approx_\alpha t_2$ and $t_2 \sim_\omega t_3$ then $\nabla \vdash t_1 \approx_\alpha t_3$.*

Proof. Formalisation is by induction on the derivation rules of $\approx_\alpha$, and it is given in the file `Alpha_Equiv_old.v`, Lem. `alpha_equiv_w_equiv_trans`. It uses properties of terms and expansions of the inference rules in the definition of $\approx_\alpha$, except for the rule ($\approx_\alpha$ [**ab**]) whose analysis requires the application of the equivariance property for $\sim_\omega$ (Lem. 3.5) and preservation of freshness under $\sim_\omega$ (Lem. 3.6). Namely, in the inductive step of this case one has as premises $\nabla \vdash t_1 \approx_\alpha (a\, b) \cdot t_2$, $\nabla \vdash [a]t_1 \approx_\alpha [b]t_2$, $[b]t_2 \sim_\omega t_3$, $\nabla \vdash a \# t_2$ and as IH: for all t_0, $(a\, b) \cdot t_2 \sim_\omega t_0$ implies $\nabla \vdash t_1 \approx_\alpha t_0$; and one needs to conclude that $\nabla \vdash [a]t_1 \approx_\alpha t_3$ (some non relevant premises are omitted). By the definition

```
Inductive w_equiv :term →term → Prop :=
 | w_equiv_Ut : w_equiv («») («»)
 | w_equiv_At : ∀ a, w_equiv (%a) (%a)
 | w_equiv_Fc : ∀ m n t t', (w_equiv t t') →
   w_equiv (Fc m n t) (Fc m n t')
 | w_equiv_Pr : ∀ t1 t2 t1' t2',
   (w_equiv t1 t1') → (w_equiv t2 t2') →
   w_equiv (<|t1,t2|>) (<|t1',t2'|>)
 | w_equiv_Ab : ∀ a t t', (w_equiv t t') →
   w_equiv ([a]^t) ([a]^t')

 | w_equiv_Su : ∀ p p' X, (∀ a, ¬ (In_ds p p' a)) →
   w_equiv (p|.X) (p'|.X) .
```

Figure 3.5: Specification of **w_equiv**

of $\sim_\omega$, $[b]t_2 \sim_\omega t_3$, t_3 should be of the form $[b]t_3'$. Thus, one needs to conclude that $\nabla \vdash [a]t_1 \approx_\alpha [b]t_3'$. Additionally, one has $t_2 \sim_\omega t_3'$ and, by equivariance of $\sim_\omega$ (Lem. 3.5), it follows that $(a\,b)t_2 \sim_\omega (a\,b) \cdot t_3'$. Also, by preservation of freshness under $\sim_\omega$ (Lem. 3.6) one has $\nabla \vdash a \# t_3'$. Instantiating the IH with $(a\,b) \cdot t_3'$, one has that $\nabla \vdash t_1 \approx_\alpha (a\,b) \cdot t_3'$. From this, applying rule ($\approx_\alpha$ [**ab**]), one finally concludes that $\nabla \vdash [a]t_1 \approx_\alpha [b]t_3'$. □

Lemma 3.8 (Equivariance of $\approx_\alpha$). *If* $\nabla \vdash s \approx_\alpha t$ *then* $\nabla \vdash \pi \cdot s \approx_\alpha \pi \cdot t$.

Proof. The formalisation of this result is in file `Alpha_Equiv_old.v`, Lem. `alpha_equiv_equivariance`, and it was done by induction on the inference rules of $\approx_\alpha$. The tricky case is when one has as hypotheses $a \neq b$, $\nabla \vdash t \approx_\alpha (a\,b) \cdot t'$ and $\nabla \vdash a \# t'$. The IH is $\nabla \vdash \pi \cdot t \approx_\alpha \pi \cdot ((a\,b) \cdot t')$. It should be proved that $\nabla \vdash \pi \cdot ([a]t) \approx_\alpha \pi \cdot ([b]t')$. Applying the definition of the permutation action and the ($\approx_\alpha$ [**ab**]) rule, three subgoals have to be proved: $\pi \cdot a \neq \pi \cdot b$, $\nabla \vdash (\pi \cdot t) \approx_\alpha ((\pi \cdot a)\,(\pi \cdot b))\,(\pi \cdot t')$ and $\nabla \vdash (\pi \cdot a) \# (\pi \cdot t')$. The first and the last are trivially solved by technical lemmas. For the second sub goal, Lem. 3.7 instantiating t_2 with $\pi \cdot ((a\,b) \cdot t')$ is applied. Then one of the new subgoals is the IH and the other one is $(\pi \cdot ((a\,b) \cdot t)) \sim_\omega ((\pi \cdot a)\,(\pi \cdot b))\,(\pi \cdot t')$. The latter is an instance of a technical lemma about the distribution of a permutation over swappings. □

Lemma 3.9 (Second intermediate transitivity lemma). *If* $\nabla \vdash t_1 \approx_\alpha t_2$ *and* $\nabla \vdash t_2 \approx_\alpha \pi \cdot t_2$ *then* $\nabla \vdash t_1 \approx_\alpha \pi \cdot t_2$.

Proof. The proof is by induction on $\approx_\alpha$ in the hypothesis $\nabla \vdash t_1 \approx_\alpha t_2$ and it is formalised in file `Alpha_Equiv_old.v`, Lem. `pi_alpha_equiv`. The interesting case is for abstractions,

that is $t_1 = [a]t'_1$ and $t_2 = [b]t'_2$. Several cases are considered according to whether a and b, a and $\pi \cdot a$, b and $\pi \cdot b$ as well as a and $\pi \cdot b$ are (or not) equal. The interesting case happens when $a \neq b$, $b \neq \pi \cdot b$ and $a \neq \pi \cdot b$ and is presented below.

After applying rule ($\approx_\alpha$ **[ab]**) to the hypothesis $\nabla \vdash [a]t'_1 \approx_\alpha [b]t'_2$, one obtains $\nabla \vdash t'_1 \approx_\alpha (a\,b) \cdot t'_2$ with $\nabla \vdash a \# t'_2$. And the IH becomes $\forall \pi, \nabla \vdash (a\,b)\,t'_2 \approx_\alpha \pi \cdot ((a\,b) \cdot t'_2) \Rightarrow \nabla \vdash t'_1 \approx_\alpha \pi \cdot ((a\,b) \cdot t'_2)$. Applying rule ($\approx_\alpha$ **[ab]**) after the action of permutation π in the second hypothesis $\nabla \vdash [b]t'_2 \approx_\alpha \pi \cdot [b]t'_2$, one obtains $\nabla \vdash t'_2 \approx_\alpha (b\ (\pi \cdot b))\,(\pi \cdot t'_2)$ with $\nabla \vdash b \# (\pi \cdot t'_2)$.

It should be proved that $\nabla \vdash [a]t'_1 \approx_\alpha \pi \cdot ([b]t'_2)$. Again, applying the action of permutation π and rule ($\approx_\alpha$ **[ab]**), two subgoals are obtained: $\nabla \vdash a \# (\pi \cdot t'_2)$ and $\nabla \vdash t'_1 \approx_\alpha (a\ (\pi \cdot b))\,(\pi \cdot t'_2)$. The former is proved by Lem. 3.1 with some manipulations over permutations and swappings. And the latter is proved by first applying Lem. 3.7 which establishes as intermediate steps: $\nabla \vdash t'_1 \approx_\alpha (a\ (\pi \cdot b)) \cdot (\pi \cdot ((a\,b)(a\,b)t'_2))$ and $(a\ (\pi \cdot b)) \cdot (\pi \cdot ((a\,b)(a\,b)t'_2)) \sim_\omega (a\ (\pi \cdot b))(\pi \cdot t'_2)$. The last is proved using equivariance and equivalence of $\sim_\omega$ (Lems. 3.5 and 3.4 respectively) while the former is obtained by IH instantiated with permutation $\pi' = (a\,b) :: \pi \oplus (a\ (\pi \cdot b))$ when its premisse becomes $\nabla \vdash (a\,b)\,t'_2 \approx_\alpha ((a\,b) :: \pi \oplus (a\ (\pi \cdot b))) \cdot ((a\,b) \cdot t'_2)$. Thus by application of Lem. 3.2 one concludes. □

Lemma 3.10 (Transitivity of $\approx_\alpha$ through $\sim_\omega$)**.** *If $\nabla \vdash t_1 \approx_\alpha t_2$ and $\nabla \vdash t_2 \approx_\alpha t_3$ then $\nabla \vdash t_1 \approx_\alpha t_3$.*

Proof. The formalisation is by induction in $\nabla \vdash t_1 \approx_\alpha t_2$ with generalisation of t_3 and it is in file `Alpha_Equiv_old.v`, Lem. `alpha_equiv_trans`. The difficult case occurs when $t_1 = [a]t'_1$, $t_2 = [b]t'_2$ and $t_3 = [c]t'_3$, with $a \neq b \neq c \neq a$. The IH is given as $\forall t_0, \nabla \vdash t'_2 \approx_\alpha t_0 \Rightarrow \nabla \vdash t'_1 \approx_\alpha t_0$, and the other hypotheses are: $\nabla \vdash t'_1 \approx_\alpha (a\,b) \cdot t'_2$, $\nabla \vdash a \# t'_2$, $\nabla \vdash t'_2 \approx_\alpha (b\,c)\,t'_3$ and $\nabla \vdash b \# t'_3$. It should be concluded that $\nabla \vdash [a]t'_1 \approx_\alpha [c]t'_3$.

Applying the rule ($\approx_\alpha$ **[ab]**) to the goal one obtains the subgoals $\nabla \vdash a \# t'_3$ and $\nabla \vdash t'_1 \approx_\alpha (a\,c)\,t'_3$. The former one is proved using the Lem. 3.1. Applying IH over the latter subgoal, it remains to prove $\nabla \vdash (a\,b) \cdot t'_2 \approx_\alpha (a\,c)\,t'_3$. So, it is needed to prove the intermediate statement $\nabla \vdash [(b\,c)(a\,b)] \cdot t'_3 \approx_\alpha [(b\,c)(a\,b)(b\,c)] \cdot t'_3$, that is possible by application of the Lem. 3.2. Manipulating swappings and using the Lem. 3.9 it is possible to infer $\nabla \vdash (a\,b) \cdot t'_2 \approx_\alpha [(b\,c)(a\,b)(b\,c)] \cdot t'_3$.

Finally, applying the Lem. 3.7 with $t_2 := [(b\,c)(a\,b)(b\,c)] \cdot t'_3$ only remains to prove that $[(b\,c)(a\,b)(b\,c)] \cdot t'_3 \sim_\omega (a\,c)\,t'_3$., that can easily be reached using properties of $\sim_\omega$ such as its equivalence and equivariance (Lems. 3.5 and 3.4 respectively). □

Lemma 3.11 (Symmetry of $\approx_\alpha$ through $\sim_\omega$)**.** *If $\nabla \vdash t \approx_\alpha t'$ then $\nabla \vdash t' \approx_\alpha t$.*

Proof. The proof is by induction on $\approx_\alpha$ over $\nabla \vdash t \approx_\alpha t'$ and it is formalised in file `Alpha_Equiv_old.v`, Lem. `alpha_equiv_sym`. The non-trivial case is when $a \neq b$ and the hypotheses are $\nabla \vdash t_0 \approx_\alpha (a\,b) \cdot t'_0$, $\nabla \vdash a \# t'_0$ and $\nabla \vdash (a\,b) \cdot t'_0 \approx_\alpha t_0$, with the subgoal $\nabla \vdash [b]t'_0 \approx_\alpha [a]t_0$. This is proved by application of the rule ($\approx_\alpha$ $[\mathbf{ab}]$) and then by a double application of Lem. 3.10 instantiated with $t_2 := (a\,b) \cdot t_0$ and $t_2 := [(a\,b)(a\,b)] \cdot t_0$. The remaining subgoals are treated using Lem. 3.8. □

3.2 Soundness of $\approx_\alpha$ without using $\sim_\omega$

This section describes the verification of $\approx_\alpha$ using the direct approach. Auxiliary Lems. 3.12 to 3.15 are used in the proofs of Lems. 3.16 and 3.17, respectively, the symmetry and the transitivity of $\approx_\alpha$.

Lemma 3.12 (Basic properties of freshness)**.**

i) $\nabla \vdash a \# \pi \cdot s$ *iff* $\nabla \vdash \pi^{-1} \cdot a \# s$

ii) $\nabla \vdash a \# s$ *iff* $\nabla \vdash \pi \cdot a \# \pi \cdot s$

Proof. These properties are formalised in file `Fresh.v`, Lems. `fresh_lemma_1` and `fresh_lemma_2`, respectively. Item (i) is proved by induction on the structure of s and item (ii) is proved as a corollary of item (i), using properties of the action of permutations π over atoms. The difficult case of item (i) is when $s = [b]t$ and $a \neq \pi \cdot b$. In this case, IH is given by $\nabla \vdash a \# \pi \cdot t$ iff $\nabla \vdash \pi^{-1} \cdot a \# t$ and the goal is $\nabla \vdash a \# [\pi \cdot b](\pi \cdot t)$ iff $\nabla \vdash \pi^{-1} \cdot a \# [b]t$. Splitting the goal in necessity and sufficiency, the first subgoal is $\nabla \vdash \pi^{-1} \cdot a \# [b]t$, and it has as hypothesis $\nabla \vdash a \# [\pi \cdot b](\pi \cdot t)$. Since $a \neq \pi \cdot b$, this hypothesis implies in $\nabla \vdash a \# (\pi \cdot t)$. One concludes applying rule ($\#\,\mathbf{a}[\mathbf{b}]$) followed by IH. Sufficiency is proved similarly. □

Lemma 3.13 (Basic properties over permutations with empty difference set)**.**
$ds(\pi, \pi') = \emptyset$ *implies*

i) $\nabla \vdash \pi \cdot a \# s$ *iff* $\nabla \vdash \pi' \cdot a \# s$

ii) $\nabla \vdash a \# \pi \cdot s$ *implies* $\nabla \vdash a \# \pi' \cdot s$*;*

iii) $\nabla \vdash \pi \cdot s \approx_\alpha t$ *implies* $\nabla \vdash \pi' \cdot s \approx_\alpha t$*; and*

iv) $\nabla \vdash s \approx_\alpha \pi \cdot t$ *implies* $\nabla \vdash s \approx_\alpha \pi' \cdot t$*.*

Proof. These results are formalised in file `Alpha_Equiv.v`, Lems. `ds_empty_fresh_1`, `ds_empty_fresh_2`, `ds_empty_equiv_1` and `ds_empty_equiv_2`, respectively. For item

(i), just observe that $ds(\pi, \pi') = \emptyset$ implies $\pi \cdot a = \pi' \cdot a$. Item (ii) is a consequence of item (i) and Lem. 3.12 (item (ii)), observing that if $ds(\pi, \pi') = \emptyset$ then also $ds(\pi^{-1}, \pi'^{-1}) = \emptyset$. Items (iii) and (iv) have similar proofs which are obtained by induction on the structure of s with case analysis over t, and on the structure of t with cases analysis over s, respectively. The non-trivial cases are those where $s = [a]s_0$ and $t = [b]t_0$. For instance, for item (iv), the hypotheses are: $ds(\pi, \pi') = \emptyset$, $a \neq \pi \cdot b$, $\nabla \vdash s_0 \approx_\alpha (a,\ \pi \cdot b)\,(\pi \cdot t_0)$ and $\nabla \vdash a \# \pi \cdot t_0$, and IH is: $\forall \pi_0, \forall \pi_1, ds(\pi_0, \pi_1) = \emptyset$ implies that $\forall u$, if $\nabla \vdash u \approx_\alpha (\pi_0 \cdot t_0)$ then $\nabla \vdash u \approx_\alpha (\pi_1 \cdot t_0)$. Having these premisses, one must prove that $\nabla \vdash [a]s_0 \approx_\alpha [\pi' \cdot b](\pi' \cdot t_0)$. After applying rule ($\approx_\alpha$ [**ab**]), three subgoals are generated: $a \neq \pi' \cdot b$, $\nabla \vdash s_0 \approx_\alpha (a\ \ \pi' \cdot b)(\pi' \cdot t_0)$ and $\nabla \vdash a \# \pi' \cdot t_0$. The first and the last are solved trivially, just observing that $\pi \cdot b = \pi' \cdot b$ (using the hypothesis $ds(\pi, \pi') = \emptyset$) and applying item (ii), respectively. The second subgoal is proved using the definition of composition of permutations and applying IH over $\nabla \vdash s_0 \approx_\alpha (\pi' \oplus (a\ \ \pi' \cdot b)) \cdot t_0$, with $\pi_0 = \pi \oplus (a\ \ \pi \cdot b)$ and $\pi_1 = \pi' \oplus (a\ \ \pi' \cdot b)$. The two new subgoals are $\nabla \vdash s_0 \approx_\alpha (\pi \oplus (a,\ \pi \cdot b)) \cdot t_0$ and $ds(\pi \oplus (a\ \ \pi \cdot b),\ \pi' \oplus (a,\ \pi' \cdot b)) = \emptyset$. The former is exactly one of the hypothesis and the latter is a consequence of the hypothesis $ds(\pi, \pi') = \emptyset$, after using results about difference sets. □

Lemma 3.14 (Inversion of permutations over $\approx_\alpha$)**.** *$\nabla \vdash \pi \cdot s \approx_\alpha t$ iff $\nabla \vdash s \approx_\alpha \pi^{-1} \cdot t$*

Proof. The formalisation of this result is in file `Alpha_Equiv.v`, Lem. `perm_inv_side`, and its proof is by induction on the structure of s with case analysis over t. The complicated case is when $s = [a]s_0$ and $t = [b]t_0$, with $\pi \cdot a \neq b$. In this case IH is $\forall \pi, \forall u, \nabla \vdash \pi \cdot s_0 \approx_\alpha u$ iff $\nabla \vdash s \approx_\alpha \pi^{-1} \cdot u$. Necessity has as hypotheses: $\nabla \vdash \pi \cdot s_0 \approx_\alpha (\pi \cdot a\ b)\, t_0$ and $\nabla \vdash \pi \cdot a \# t_0$. One applies rule ($\approx_\alpha$ [**ab**]) to the objective $\nabla \vdash [a]s_0 \approx_\alpha [\pi^{-1} \cdot b](\pi^{-1} \cdot t_0)$, generating three subgoals: $a \neq \pi^{-1} \cdot b$, $\nabla \vdash s_0 \approx_\alpha (a\ \pi^{-1} \cdot b)\,(\pi^{-1} \cdot t_0)$ and $\nabla \vdash a \# \pi^{-1} \cdot t_0$. The first and the last are trivially solved by manipulations of the permutation π and an application of Lem. 3.12, item (i), respectively. IH is applied to the first hypothesis, resulting in $\nabla \vdash s_0 \approx_\alpha \pi^{-1} \cdot (\pi \cdot a\ b)\, t_0$. Then the definition of composition of permutations and Lem. 3.13, item (iv), is applied with $\pi = (\pi \cdot a\ b) \oplus \pi^{-1}$ and $\pi' = \pi^{-1} \oplus (a\ \pi^{-1} \cdot b)$, only remaining to prove that $ds((\pi \cdot a\ b) \oplus \pi^{-1},\ \pi^{-1} \oplus (a\ \pi^{-1} \cdot b)) = \emptyset$, which is easily showed by manipulations of permutations over atoms. The proof of sufficiency is very similar. □

Lemma 3.15 (Equivariance of $\approx_\alpha$)**.** *$\nabla \vdash s \approx_\alpha t$ iff $\nabla \vdash \pi \cdot s \approx_\alpha \pi \cdot t$.*

Proof. The formalisation of this result is in file `Alpha_Equiv.v`, Lem. `alpha_equiv_equivariance`. For necessity and sufficiency, Lem. 3.14 is applied to the objective and in the hypothesis, respectively. Then Lem. 3.13, item (iv), is applied and one only needs to prove that $ds(\pi \oplus \pi^{-1}, id) = \emptyset$, and this statement is easily reached by an application of results of difference sets and manipulation of permutations over atoms. □

Lemma 3.16 (Symmetry of $\approx_\alpha$ by the direct approach)**.** *If* $\nabla \vdash s \approx_\alpha t$ *then* $\nabla \vdash t \approx_\alpha s$.

Proof. The formalisation is in file `Alpha_Equiv.v`, Lem. `alpha_equiv_sym`. The proof is by induction over the derivation cases of $\nabla \vdash s \approx_\alpha t$. The interesting case is given by rule ($\approx_\alpha$ [**ab**]). In this case, $\nabla \vdash [a]u \approx_\alpha [b]v$ whenever $\nabla \vdash u \approx_\alpha (a\,b) \cdot v$ and $\nabla \vdash a \# v$. By equivariance of freshness (Lem. 3.12, item (ii)), $\nabla \vdash b \# (a\,b) \cdot v$ is obtained. By IH, $\nabla \vdash (a\,b) \cdot v \approx_\alpha u$ and then $\nabla \vdash b \# u$, by Lem. 3.1. Finally, by inversion of permutations over $\approx_\alpha$ (Lem. 3.14), $\nabla \vdash v \approx_\alpha (a\,b) \cdot u$. This and $\nabla \vdash b \# u$ prove $\nabla \vdash [b]v \approx_\alpha [a]u$. □

Lemma 3.17 (Transitivity of $\approx_\alpha$ by the direct approach)**.** $\nabla \vdash t_1 \approx_\alpha t_2$ *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 size of t_1 and case analysis over $\nabla \vdash t_1 \approx_\alpha t_2$ and $\nabla \vdash t_2 \approx_\alpha t_3$. Its formalisation is in file `Alpha_Equiv.v`, Lem. `alpha_equiv_trans`. The subsequent steps show the abstraction case, which is the most interesting one due to the asymmetry of rule ($\approx_\alpha$ [**ab**]) (see Fig. 2.4). Consider $t_1 = [a]u$, $t_2 = [b]v$ and $t_3 = [c]w$. So one must analyse the following situations:

- $a = b = c$: thus the result follows by IH;
- $a = b \neq c$: by definition, $\nabla \vdash u \approx_\alpha v$ and $\nabla \vdash v \approx_\alpha (b\,c) \cdot w$ and $\nabla \vdash b \# w$. By IH, $\nabla \vdash u \approx_\alpha (b\,c) \cdot w$. As $a = b$, then freshness condition to a is satisfied as well;
- $a \neq b = c$: one has that $\nabla \vdash a \# v$, $\nabla \vdash u \approx_\alpha (a\,c) \cdot v$ and $\nabla \vdash v \approx_\alpha w$. By Lem. 3.15, $\nabla \vdash (a\,c) \cdot v \approx_\alpha (a\,c) \cdot w$ and, by IH, $\nabla \vdash u \approx_\alpha (a\,c) \cdot w$. By Lem. 3.12, $\nabla \vdash c \# (a\,c) \cdot v$ and $\nabla \vdash c \# (a\,c) \cdot w$ by Lem. 3.1. Finally, again by Lem. 3.12, $\nabla \vdash a \# w$;
- $b \neq a = c$: it is known that $\nabla \vdash u \approx_\alpha (b\,c) \cdot v$ and $\nabla \vdash v \approx_\alpha (b\,c) \cdot w$. Then $\nabla \vdash (b\,c) \cdot v \approx_\alpha w$ by Lem. 3.14. By IH $\nabla \vdash u \approx_\alpha w$;
- $a \neq b \neq c \neq a$: it is necessary to prove that $\nabla \vdash u \approx_\alpha (a\,c) \cdot w$ and $\nabla \vdash a \# w$. Let us prove first the freshness condition. by definition of $\approx_\alpha$, $\nabla \vdash a \# v$ and $\nabla \vdash v \approx_\alpha (b\,c) \cdot w$. By Lem. 3.1, $\nabla \vdash a \# (b\,c) \cdot w$ and, by Lem. 3.12, $\nabla \vdash a \# w$. Now let us prove $\approx_\alpha$: By Lem. 3.15, $\nabla \vdash (a\,b) \cdot v \approx_\alpha [(b\,c), (a\,b)] \cdot w$. As $ds([(b\,c), (a\,b)], (a\,c)) = \{a, b\}$ and both atoms are fresh in w, then $\nabla \vdash [(b\,c), (a\,b)] \cdot w \approx_\alpha (a\,c) \cdot w$ by Lem. 3.2. Now, applying IH twice, one obtains $\nabla \vdash u \approx_\alpha (a\,c) \cdot w$.

□

3.3 Comparing the two formalisation approaches

A brief comparison between the two formalisation approaches is summarised below:

- The proof of symmetry of $\approx_\alpha$ without using $\sim_\omega$ (Lem. 3.16) is independent of transitivity, unlike the proof that uses $\sim_\omega$, in which the formalisation of symmetry relies on transitivity;
- Lems. 3.4 to 3.7 w.r.t. $\sim_\omega$ are no longer necessary in the direct approach. Also the Second intermediate transitivity lemma (Lem. 3.9) is not needed. Counting all these lemmas, a total of 338 proof lines were eliminated;
- In the approach without $\sim_\omega$, the few auxiliary lemmas on $\approx_\alpha$ (Lems. 3.12 to 3.13) are proved by simple induction either on the structure of the terms or on the inference rules of $\approx_\alpha$. Only 177 proof lines were added with these new results;
- Although, in the direct approach sufficiency is also proved in the equivariance lemma of $\approx_\alpha$ (Lem. 3.15), the proof is slightly shorter than the presented in Lem. 3.8;
- In the approach without $\sim_\omega$, the proofs of Lems. 3.16 and 3.17, respectively, symmetry and transitivity of $\approx_\alpha$ are based only on nominal properties and properties of $\approx_\alpha$, but the number of proof lines of these two lemmas is almost the same of the approach with $\sim_\omega$;
- The formalisation without $\sim_\omega$ is shorter. Precisely, adopting this approach results in a reduction of 251 proof lines in the whole formalisation, which represents a reduction of approximately 33 % in comparison with the formalisation with $\sim_\omega$.

Chapter 4

Nominal α, A, C and AC equality-checking

The present chapter contains results published in [5, 6]. Those extend $\approx_\alpha$ by adding A, C and AC function symbols in the signature. To check α-equivalence modulo A, C and AC, denoted $\approx_{\{A,C,AC\}}$, one uses soundness of $\approx_\alpha$ to check $\approx_{\{A,C,AC\}}$. This extension is obtained via the specification of an inductive relation **equiv**(S) (see Fig. 4.8 and file `Equiv.v`) parameterised by a set S of indices, where each index is associated to a different equational theory. In particular, if $S = \emptyset$ the relation **equiv**(S) excludes from the specification of **equiv** all specialised inference rules for any equational theory. The relation **equiv**$(\emptyset)$ is formally proved be equivalent to the relation $\approx_\alpha$ (see file `Equiv.v`, Lem. alpha_equiv_eq_equiv):

$$\nabla \vdash t \approx_\alpha t' \Leftrightarrow \textbf{equiv}(\emptyset)(\nabla, t, t').$$

The generic relation **equiv** considers A, AC and C function symbols if $0, 1$ or 2 belong to S, respectively. Namely, **equiv**$(\{0\})$, **equiv**$(\{1\})$, **equiv**$(\{2\}))$ and **equiv**$(\{0, 1, 2\})$ select the specialised inductive rules in the definition of **equiv** for the relation $\approx_\alpha$ modulo A, AC, C and combinations of A, AC and C, respectively. In this way one builds the relations $\approx_{\alpha,A}$, $\approx_{\alpha,AC}$, $\approx_{\alpha,C}$ and $\approx_{\{A,C,AC\}}$. For readability, from now on, instead of indices 0, 1 and 2, the corresponding abbreviations A, AC and C will be used.

In Subsec. 4.5, an alternative recursive version of **equiv** is presented that has been proved be equivalent to the relation $\approx_{\{A,C,AC\}}$, and from which OCaml executable code has been automatically extracted.

4.1 Operations over tuples

The inductive rules for A and AC operators in the definition of the relation $\approx_{\{A,C,AC\}}$ use three auxiliary operators specified in file `Tuples.v` that deal with arguments of function symbols. Arguments of a function symbol f are terms built using the constructor for pairs and the arguments of terms headed by the same function symbol f. These operators, specified as in Figs. 4.1, 4.2 and 4.3 extract the relevant information of the arguments to which a(n A, C or AC) symbol f_n^E is applied and specify the *length or number of arguments*, $\|t\|_{f_n^E} := \mathsf{TPlength}\, t\, E\, n$, and the *selection* and *deletion of the ith argument*, respectively, $t_{(i)_{f_n^E}} := \mathsf{TPith}\, i\, t\, E\, n$ and $t_{[\star i]_{f_n^E}} := \mathsf{TPithdel}\, i\, t\, E\, n$.

```
Fixpoint TPlength (t: term) (E n: nat) : nat :=
match t with
  | (<| t1 , t2 |>) ⇒ (TPlength t1 E n) + (TPlength t2 E n)
  | (Fc E0 n0 t0) ⇒ if (E,n) = (E0,n0)
                       then (TPlength t0 E n)
                       else 1
  | _ ⇒ 1
end.
```

Figure 4.1: Specification of TPlength

```
Fixpoint TPith (i: nat) (t: term) (E n: nat) : term :=
match t with
  | (<| t1 , t2 |>) ⇒ let l1 := TPlength t1 E n in
                       if i ≤ l1
                       then TPith i t1 E n
                       else TPith (i-l1) t2 E n
  | (Fc E0 n0 t0) ⇒ if (E,n) = (E0,n0)
                       then TPith i t0 E n
                       else t

  | _ ⇒ t
end.
```

Figure 4.2: Specification of TPith

Exs. 4.1 to 4.3 illustrates the behaviour of the operators $\| _ \|_f$, $_\,{}_{(_)_f}$ and $_\,{}_{[\star _]_f}$.

Example 4.1. *For the number of arguments.*

1. $\|f\langle\rangle\|_f = \|\langle\rangle\|_f = 1$;

2. $\|f\,\langle\overline{a},\overline{b}\rangle\|_f = \|\langle\overline{a},\overline{b}\rangle\|_f = 2$, *but* $\|g\,\langle\overline{a},\overline{b}\rangle\|_f = 1$;

```
Fixpoint TPithdel (i: nat) (t: term) (E n: nat) : term :=
match t with
 | (<| t1 , t2 |>) ⇒ let l1 := (TPlength t1 E n) in
                     let l2 := (TPlength t2 E n) in
                       if i ≤ l1
                       then
                         if l1 = 1
                         then t2
                         else <| (TPithdel i t1 E n) , t2 |>
                       else
                         let ii := i-l1 in
                           if l2 = 1
                           then t1
                           else <| t1 , (TPithdel ii t2 E n) |>
 | (Fc E0 n0 t0) ⇒ if (TPlength (Fc E0 n0 t0) E n) = 1
                    then «»
                    else Fc E0 n0 (TPithdel i t0 E n)
 | _ ⇒ «»
end.
```

Figure 4.3: Specification of TPithdel

3. $\|f\,\langle [a](\pi.X), f\,\langle \bar{b}, g\,\langle \bar{a}, f\,\langle \bar{a}, \bar{b}\rangle\rangle\rangle\rangle\|_f =$
$\|[a](\pi.X)\|_f + \|\bar{b}\|_f + \|g\,\langle \bar{a}, f\,\langle \bar{a}, \bar{b}\rangle\rangle\|_f = 3$.

Example 4.2. *For the selection of the ith argument.*

1. $t_{(0)_f} = t_{(1)_f}$ *and, if* $i > \|t\|_f$ *then* $t_{(i)_f} = t_{(\|t\|_f)_f}$ *(cf. Lem. 4.1);*

2. *If* $\|t\|_f = 1$ *and* t *is not headed by* f *then* $t_{(1)_f} = t$*, but also* $(f\,f\,t)_{(1)_f} = t$*;*

3. $(f\,\langle [a](\pi.X), f\,\langle \bar{b}, g\,\langle \bar{a}, f\,\langle \bar{a}, \bar{b}\rangle\rangle\rangle\rangle)_{(3)_f} = (f\,\langle \bar{b}, g\,\langle \bar{a}, f\,\langle \bar{a}, \bar{b}\rangle\rangle\rangle)_{(2)_f} =$
$(g\,\langle \bar{a}, f\,\langle \bar{a}, \bar{b}\rangle\rangle)_{(1)_f} = g\,\langle \bar{a}, f\,\langle \bar{a}, \bar{b}\rangle\rangle$.

Example 4.3. *For the deletion of the ith argument.*

1. $t_{[\star 0]_f} = t_{[\star 1]_f}$ *and if* $i > \|t\|_f$ *then* $t_{[\star i]_f} = t_{[\star \|t\|_f]_f}$ *(cf. Lem. 4.1);*

2. *If* $\|t\|_f = 1$ *then* $t_{[\star 1]_f} = \langle\,\rangle$*;*

3. $(f\,\langle [a](\pi.X), f\,\langle \bar{b}, g\,\langle \bar{a}, f\,\langle \bar{a}, \bar{b}\rangle\rangle\rangle\rangle)_{[\star 2]_f} =$
$f\,\langle [a](\pi.X), (f\,\langle \bar{b}, g\,\langle \bar{a}, f\,\langle \bar{a}, \bar{b}\rangle\rangle\rangle)_{[\star 1]_f}\rangle = f\,\langle [a](\pi.X), f\,(g\,\langle \bar{a}, f\,\langle \bar{a}, \bar{b}\rangle\rangle)\rangle$.

Remark 4.1. *The use of operators the* $\|\,_\,\|_f$, $_\,(\,_\,)_f$ *and* $_\,[\star\,_\,]_f$ *has two advantages: first, neither an additional data structure to express associativity is necessary (e.g. lists, sequences, arrays) nor an operator for* flattening *terms; second, the adopted grammar permits the manipulation of arbitrary combinations of different function symbols with*

different equational properties, occurring simultaneously in a term, via the use of specialised rules which fit the given signature and its corresponding equational theory. This simplifies the treatment of α-equivalence modulo A, C and AC, and other equational theories.

Lem. 4.1, lists some results from a much longer list of formalised lemmas on tuples (see file `Tuples.v`). The proofs of these results are very technical and obtained by simple induction on the structure of terms, so they are omitted in the present text. These properties are applied mainly in the manipulation of elements of tuples in the proofs of Lems. 4.7 to 4.9.

Lemma 4.1 (Basic properties of the operators: $\| _ \|_f$, $_(_)_f$ and $_[\star _]_f$)**.**

i) *Selecting or removing the 0-th element of a tuple is the same as operating over the first element:* $t_{(0)} = t_{(1)}$ *and* $t_{[\star 0]} = t_{[\star 1]}$*;*

ii) *In a tuple with n elements, the operation of selecting or removing the i-th element, with $i \geq n$, is the same as operating over the n-th element: if* $i \geq \|t\|$ *then* $t_{(i)} = t_{(\|t\|)}$ *and* $t_{[\star i]} = t_{[\star \|t\|]}$*;*

iii) *If* $\|t\| = 1$ *then, for any i,* $t_{[\star i]} = \langle\rangle$*;*

iv) *If* $\|t\| \neq 1$ *then* $\|t_{[\star i]}\| = \|t\| - 1$*;*

v) *If* $0 < i < j$ *and* $i < \|t\|$ *then* $(t_{[\star j]})_{(i)} = t_{(i)}$*;*

vi) *If* $0 < i < j \leq \|t\|$ *then* $(t_{[\star j]})_{[\star i]} = (t_{[\star i]})_{[\star (j-1)]}$*;*

vii) *If* $0 < i < \|t\|$ *and* $i \geq j$ *then* $(t_{[\star j]})_{(i)} = t_{(i+1)}$ *and* $(t_{[\star j]})_{[\star i]} = (t_{[\star (i+1)]})_{[\star j]}$*.*

These results are formalised in the following corresponding Lemmas of file `Tuples.v`*:*

i) `TPith_0` *and* `TPithdel_0`*;*

ii) `TPith_overflow` *and* `TPithdel_overflow`*;*

iii) `TPithdel_TPlength_1`*;*

iv) `TPlength_TPithdel`*;*

v) `TPith_TPithdel_lt`*;*

vi) `TPithdel_lt_comm`*;*

vii) `TPith_TPithdel_geq` *and* `TPithdel_geq_comm`*.*

4.2 Extension of the rules for $\approx_\alpha$

New rules ($\approx_\alpha$ **A**), ($\approx_\alpha$ **C**), and ($\approx_\alpha$ **AC**) for A, C and AC are included. These rules are combined with those from Fig. 2.4 for $\approx_\alpha$, with the following modification: ($\approx_\alpha$ **app**) is replaced by ($\approx_\alpha \overline{\mathbf{app}}$) and applies whenever the function symbol f_k^E applied to s is such that $E \notin S$ or $E = C \in S$ and s is not a pair. Otherwise, when $E = A, C$ or AC and $E \in S$, rules ($\approx_\alpha$ **A**), ($\approx_\alpha$ **C**) or ($\approx_\alpha$ **AC**) apply. Therefore, if f is not an A, C or AC function symbol or $A, C, AC \notin S$, the behaviour of ($\approx_\alpha \overline{\mathbf{app}}$) and ($\approx_\alpha$ **app**) would be exactly the same. These rules define an extended calculus for *general* α-equivalence modulo A, C and AC. Other equational theories might be included similarly. In the following, $\nabla \vdash s \approx_{\{A,C,AC\}} t$ denotes that s and t are α-equivalent modulo A, C and AC under the context ∇.

$$\frac{\nabla \vdash s \approx_{\{A,C,AC\}} t}{\nabla \vdash f_k^E s \approx_{\{A,C,AC\}} f_k^E t} \begin{array}{l} E \notin S \text{ or} \\ E = C \text{ and } s \text{ is not a pair} \end{array} \quad (\approx_\alpha \overline{\mathbf{app}})$$

Figure 4.4: ($\approx_\alpha \overline{\mathbf{app}}$)-rule for $\approx_{\{A,C,AC\}}$

$$\frac{\begin{array}{c}\nabla \vdash (f_k^A\, s)_{(\mathbf{1})_{f_k^A}} \approx_{\{A,C,AC\}} (f_k^A\, t)_{(\mathbf{1})_{f_k^A}}, \\ \nabla \vdash (f_k^A\, s)_{[\star\mathbf{1}]_{f_k^A}} \approx_{\{A,C,AC\}} (f_k^A\, t)_{[\star\mathbf{1}]_{f_k^A}}\end{array}}{\nabla \vdash f_k^A\, s \approx_{\{A,C,AC\}} f_k^A\, t} \quad (\approx_\alpha \mathbf{A})$$

Figure 4.5: ($\approx_\alpha$ **A**)-rule for A function symbols

Rule ($\approx_\alpha$ **A**) applies when the terms compared are headed by the same A function symbol and $A \in S$. It verifies recursively if the first arguments on the *lhs* and *rhs* are related by $\approx_{\{A,C,AC\}}$ as well as the result of applying the root function symbol to the respective tuples without the first argument.

$$\frac{\nabla \vdash s_0 \approx_{\{A,C,AC\}} t_i,\ \nabla \vdash s_1 \approx_{\{A,C,AC\}} t_{1-i}}{\nabla \vdash f_k^C \langle s_0, s_1 \rangle \approx_{\{A,C,AC\}} f_k^C \langle t_0, t_1 \rangle}\ i = 0, 1\ (\approx_\alpha \mathbf{C})$$

Figure 4.6: ($\approx_\alpha$ **C**)-rule for C function symbols

Rule ($\approx_\alpha$ **C**) has two possibilities of application: for $i = 0$ (resp. $i = 1$) one must have $\nabla \vdash s_0 \approx_{\{A,C,AC\}} t_0$ and $\nabla \vdash s_1 \approx_{\{A,C,AC\}} t_1$ (resp. $\nabla \vdash s_0 \approx_{\{A,C,AC\}} t_1$ and $\nabla \vdash s_1 \approx_{\{A,C,AC\}} t_0$). The case where f_k^C is applied to a term different of a pair is considered in the ($\approx_\alpha \overline{\mathbf{app}}$)-rule.

$$\frac{\nabla \vdash (f_k^{AC}\, s)_{(\mathbf{1})_{f_k^{AC}}} \approx_{\{A,C,AC\}} (f_k^{AC}\, t)_{(\mathbf{i})_{f_k^{AC}}}, \quad \nabla \vdash (f_k^{AC}\, s)_{[\star\mathbf{1}]_{f_k^{AC}}} \approx_{\{A,C,AC\}} (f_k^{AC}\, t)_{[\star\mathbf{i}]_{f_k^{AC}}}}{\nabla \vdash f_k^{AC}\, s \approx_{\{A,C,AC\}} f_k^{AC}\, t}\; AC \in S\; (\approx_\alpha \mathbf{AC})$$

Figure 4.7: $(\approx_\alpha \mathbf{AC})$-rule for AC function symbols

Rule $(\approx_\alpha \mathbf{AC})$ behaves similarly to rule $(\approx_\alpha \mathbf{A})$: the fundamental difference is that the first argument on the *lhs* can be compared modulo $\approx_{\{A,C,AC\}}$ with any arbitrary argument on the *rhs*. If there exists such argument, say the ith, it remains to check that the terms obtained applying the function symbol to the tuples deleting the first and the ith arguments to the right and to the left are related by $\approx_{\{A,C,AC\}}$.

Example 4.4. *$\nabla \vdash f\langle \bar{a}, g^{AC}\langle \bar{b}, \langle \bar{c}, \bar{d}\rangle\rangle\rangle \approx_{\{A,C,AC\}} f\langle \bar{a}, g^{AC}\langle\langle \bar{d}, \bar{c}\rangle, \bar{b}\rangle\rangle$, where g is AC, f is a function symbol that allows only α-equivalence and $AC \in S$.*

$$\frac{(\approx_\alpha \mathbf{pair})\ \dfrac{\nabla \vdash \bar{a} \approx_{\{A,C,AC\}} \bar{a} \qquad \dfrac{\nabla \vdash \bar{b} \approx_{\{A,C,AC\}} \bar{b} \qquad \dfrac{\overline{\nabla \vdash \bar{c} \approx_{\{A,C,AC\}} \bar{c}} \qquad \dfrac{\overline{\nabla \vdash \bar{d} \approx_{\{A,C,AC\}} \bar{d}} \quad \overline{\nabla \vdash \langle\rangle \approx_{\{A,C,AC\}} \langle\rangle}}{\nabla \vdash g^{AC}\bar{d} \approx_{\{A,C,AC\}} g^{AC}\bar{d}}\,(\approx_\alpha \mathbf{AC})}{\nabla \vdash g^{AC}\langle \bar{c}, \bar{d}\rangle \approx_{\{A,C,AC\}} g^{AC}\langle \bar{d}, \bar{c}\rangle}\,(\approx_\alpha \mathbf{AC})}{\nabla \vdash g^{AC}\langle \bar{b}, \langle \bar{c}, \bar{d}\rangle\rangle \approx_{\{A,C,AC\}} g^{AC}\langle\langle \bar{d}, \bar{c}\rangle, \bar{b}\rangle}\,(\approx_\alpha \mathbf{AC})}{\nabla \vdash \langle \bar{a}, g^{AC}\langle \bar{b}, \langle \bar{c}, \bar{d}\rangle\rangle\rangle \approx_{\{A,C,AC\}} \langle \bar{a}, g^{AC}\langle\langle \bar{d}, \bar{c}\rangle, \bar{b}\rangle\rangle}}{\nabla \vdash f\langle \bar{a}, g^{AC}\langle \bar{b}, \langle \bar{c}, \bar{d}\rangle\rangle\rangle \approx_{\{A,C,AC\}} f\langle \bar{a}, g^{AC}\langle\langle \bar{d}, \bar{c}\rangle, \bar{b}\rangle\rangle}\,(\approx_\alpha \mathbf{app})$$

The key code fragment of the formalisation regarding α-equivalence modulo A, C, and AC is the inductive definition **equiv** in Fig. 4.8 (available in the file `Equiv.v` of the specification). This definition uses notations and operators given in Figs. 3.1, 4.1, 4.2 and 4.3, and specifies a relation that has type Context $\to$ **term** $\to$ **term** $\to$ `Prop` and a set of naturals S as parameter. This definition includes specific rules for each constructor of the nominal syntax, and a signature that may contain A, C and AC function symbols according to S.

The inference rules $(\approx_\alpha \langle\rangle)$, $(\approx_\alpha \mathbf{atom})$, $(\approx_\alpha \overline{\mathbf{app}})$, $(\approx_\alpha [\mathbf{aa}])$, $(\approx_\alpha [\mathbf{ab}])$, $(\approx_\alpha \mathbf{var})$, $(\approx_\alpha \mathbf{pair})$, $(\approx_\alpha \mathbf{A})$ and $(\approx_\alpha \mathbf{AC})$ are specified, respectively, by the following constructors of **equiv**: equiv_Ut; equiv_At; equiv_Fc; equiv_Ab_1; equiv_Ab_2; equiv_Su; equiv_Pr; equiv_A and equiv_AC. And additionally, the two cases of rule $(\approx_\alpha \mathbf{C})$ are specified by equiv_C1 and equiv_C2.

As in the specification of $\approx_\alpha$ (Fig. 3.3), the constructor equiv_Ut (resp. equiv_At) expresses that $\langle\rangle$ (resp. $\bar{a}$) is related with itself, for any S and any C. In equiv_Fc, Fc E n t is related to Fc E n t', if E does not belong to S, that means one is not dealing with an

```
Inductive equiv (S : set nat): Context → term → term → Prop :=

 | equiv_Ut : ∀ C, equiv S C («») («»)

 | equiv_At : ∀ C a, equiv S C (%a) (%a)

 | equiv_Pr : ∀ C t1 t2 t1' t2',
   (equiv S C t1 t1') → (equiv S C t2 t2') → equiv S C (<|t1, t2|>) (<|t1',t2'|>)

 | equiv_Fc : ∀ E n t t' C, (¬ set_In E S ∨ (E = 2 ∧ ((¬ is_Pr t) ∨ (¬ is_Pr t'))) →
   (equiv S C t t') → equiv S C (Fc E n t) (Fc E n t')

 | equiv_Ab_1 : ∀ C a t t', (equiv S C t t') → equiv S C ([a]^t) ([a]^t')

 | equiv_Ab_2 : ∀ C a a' t t', a ≠ a' →
   (equiv S C t (|[(a,a']| @ t')) → C ⊢ a # t' → equiv S C ([a]^t) ([a']^t')

 | equiv_Su : ∀ (C: Context) p p' (X: Var), (∀ a, (In_ds p p' a) → set_In (a, X) C) →
   equiv S C (p|.X) (p'|.X)

 | equiv_A : set_In 0 S → ∀ n t t' C,
   (equiv S C (TPith 1 (Fc 0 n t) 0 n) (TPith 1 (Fc 0 n t') 0 n)) →
   (equiv S C (TPithdel 1 (Fc 0 n t) 0 n) (TPithdel 1 (Fc 0 n t') 0 n)) →
   (equiv S C (Fc 0 n t) (Fc 0 n t'))

 | equiv_AC : set_In 1 S → ∀ n t t' i C,
   (equiv S C (TPith 1 (Fc 1 n t) 1 n) (TPith i (Fc 1 n t') 1 n)) →
   (equiv S C (TPithdel 1 (Fc 1 n t) 1 n) (TPithdel i (Fc 1 n t') 1 n)) →
   (equiv S C (Fc 1 n t) (Fc 1 n t'))

| equiv_C1 : set_In 2 S → ∀ n s0 s1 t0 t1 C,
   (equiv S C s0 t0) → (equiv S C s1 t1) →
   (equiv S C (Fc 2 n (<|s0, s1|>)) (Fc 2 n (<|t0, t1|>)))

| equiv_C2 : set_In 2 S → ∀ n s0 s1 t0 t1 C,
   (equiv S C s0 t1) → (equiv S C s1 t0) →
   (equiv S C (Fc 2 n (<|s0, s1|>)) (Fc 2 n (<|t0, t1|>))) .
```

Figure 4.8: Specification of **equiv**

A, AC or C function symbol, or if $E = 2$, that means one is dealing with a C function symbol which is not applied to a pair $((\neg$ is_Pr $t) \vee (\neg$ is_Pr $t'))$. Notice that, equiv_Fc was specified to cover the cases in which rules for A, C or an AC function symbols do not apply.

The constructor equiv_Ab_2 relates [a]^t to [a']^t', considering S and a freshness context C, whenever the atoms are different (i.e., $a \neq a'$), t is related with |[(a,a']]| @ t' (an application of swapping $(a\,a')$ to t') and a is fresh in t' in the context C, which uses notation $C \vdash a$ # t' in the specification.

The constructor equiv_AC relates Fc 1 n t to Fc 1 n t', given S and C, whenever 1 belongs to S (otherwise equiv_Fc is applied) and there exists an i such that TPith 1 (Fc 1 n t) 1 n is related to TPith i (Fc 1 n t') 1 n and TPithdel 1 (Fc 1 n t) 1 n is related to TPithdel i (Fc 1 n t') 1 n.

4.3 Formalisation of the soundness of $\approx_{\{A,C,AC\}}$

The following steps were performed in order to check that $\approx_{\{A,C,AC\}}$ is indeed an equivalence relation. After proving an intermediate transitivity lemma for $\approx_{\{A,C,AC\}}$ (Lem. 4.3), one proves *freshness preservation* and *equivariance* (Lems. 4.4 and 4.5) of $\approx_{\{A,C,AC\}}$ and then, transitivity before symmetry (Lems. 4.8 and 4.9). By using the parameter set S on the **equiv**(S) relation and renaming superscripts of function symbols, one obtains as corollary of the soundness of $\approx_{\{A,C,AC\}}$ the soundness of $\approx_{\alpha,A}$, $\approx_{\alpha,C}$ and $\approx_{\alpha,AC}$.

In addition to preservation of freshness and equivariance, the intermediate transitivity lemma (Lem. 4.3) is relevant to guarantee key properties on swappings and permutations acting over $\approx_{\{A,C,AC\}}$-related terms as, for instance, $\nabla \vdash t \approx_{\{A,C,AC\}} (a\,a') \cdot t' \Rightarrow \nabla \vdash (a'\,a) \cdot t \approx_{\{A,C,AC\}} t'$ (see file `AACC_Equiv.v`, Lem. `aacc_equiv_swap_inv_side`).

Lemma 4.2 (Reverse of $\approx_{\{A,C,AC\}}$)**.** *The inference rules of* $\approx_{\{A,C,AC\}}$ *are* invertible*, which means that:*

i) If $\nabla \vdash \langle s_0, s_1\rangle \approx_{\{A,C,AC\}} \langle t_0, t_1\rangle$ *then* $\nabla \vdash s_0 \approx_{\{A,C,AC\}} t_0$ *and* $\nabla \vdash s_1 \approx_{\{A,C,AC\}} t_1$*;*

ii) If $\nabla \vdash [a]s \approx_{\{A,C,AC\}} [a]t$ *or* $\nabla \vdash f_k^E\, s \approx_{\{A,C,AC\}} f_k^E\, t$ *(with* $E \notin \{A, C, AC\}$*) then* $\nabla \vdash s \approx_{\{A,C,AC\}} t$*;*

iii) If $\nabla \vdash [a]s \approx_{\{A,C,AC\}} [b]t$ *then* $\nabla \vdash s \approx_{\{A,C,AC\}} (a\,b) \cdot t$ *and* $\nabla \vdash a \# t$*;*

iv) If $\nabla \vdash f_k^A s \approx_{\{A,C,AC\}} f_k^A t$ *then* $\nabla \vdash (f_k^A s)_{(1)} \approx_{\{A,C,AC\}} (f_k^A t)_{(1)}$ *and* $\nabla \vdash (f_k^A s)_{[\star 1]} \approx_{\{A,C,AC\}} (f_k^A t)_{[\star 1]}$*;*

v) If $\nabla \vdash f_k^{AC} s \approx_{\{A,C,AC\}} f_k^{AC} t$ *then there exists* $i > 0, \leq \|t\|$*, such that* $\nabla \vdash (f_k^{AC} s)_{(1)} \approx_{\{A,C,AC\}} (f_k^{AC} t)_{(i)}$ *and* $\nabla \vdash (f_k^{AC} s)_{[\star 1]} \approx_{\{A,C,AC\}} (f_k^{AC} t)_{[\star i]}$*;*

vi) If $\nabla \vdash f_k^C\langle s_0, s_1\rangle \approx_{\{A,C,AC\}} f_k^C\langle t_0, t_1\rangle$ *then either* $\nabla \vdash s_0 \approx_{\{A,C,AC\}} t_0$ *and* $\nabla \vdash s_1 \approx_{\{A,C,AC\}} t_1$*, or* $\nabla \vdash s_0 \approx_{\{A,C,AC\}} t_1$ *and* $\nabla \vdash s_1 \approx_{\{A,C,AC\}} t_0$*;*

vii) If $\nabla \vdash \pi.X \approx_{\{A,C,AC\}} \pi'.X$ *then* $ds(\pi, \pi') \subseteq \nabla$*.*

Proof. Items *i)* to *vii)* are proved using a tactic of the Coq named `inversion` that is applied to the hypothesis, resulting exactly on the objectives that one is trying to prove. □

Remark 4.2. *In the proofs Lems. 4.3 to 4.5 the treatment given to the cases of rules* ($\approx_\alpha$ **C**)*,* ($\approx_\alpha$ **A**) *and* ($\approx_\alpha$ **AC**) *are very similar to the treatment of rule* ($\approx_\alpha$ $\overline{\textbf{app}}$)*. Thus the interesting case is always given by rule* ($\approx_\alpha$ [**ab**])*.*

Lemma 4.3 (Intermediate transitivity for $\approx_{\{A,C,AC\}}$ with $\approx_\alpha$)**.** *If* $\nabla \vdash s \approx_{\{A,C,AC\}} t$ *and* $\nabla \vdash t \approx_\alpha u$ *then* $\nabla \vdash s \approx_{\{A,C,AC\}} u$*.*

Proof. The formalisation of this result is in file `AACC_Equiv.v`, Lem. `aacc_alpha_equiv_trans` and it is obtained as follows: after generalisation of u, induction is applied on deduction rules of $\approx_{\{A,C,AC\}}$ for $\nabla \vdash s \approx_{\{A,C,AC\}} t$. Some cases require analysis over the premisse $\nabla \vdash t \approx_\alpha u$; for instance, in the case in which one has $t = \langle t_1, t_2\rangle$, the `inversion` tactic of Coq is applied to obtain that u must be equal to $\langle u_1, u_2\rangle$ with $\nabla \vdash t_1 \approx_\alpha u_1$ and $\nabla \vdash t_2 \approx_\alpha u_2$, according to the inference rule ($\approx_\alpha$ **pair**). □

Lemma 4.4 (Freshness preservation under $\approx_{\{A,C,AC\}}$)**.** *If* $\nabla \vdash a \# s$ *and* $\nabla \vdash s \approx_{\{A,C,AC\}} t$ *then* $\nabla \vdash a \# t$*.*

Proof. The proof is by induction on $\approx_{\{A,C,AC\}}$ (see Lem. `aacc_equiv_fresh` of file `AACC_Equiv.v`), using technical results about the freshness relation for dealing with cases related with rules ($\approx_\alpha$ [**aa**]) and ($\approx_\alpha$ [**ab**]) for the case in which s and t are abstractions. □

Lemma 4.5 (Equivariance of $\approx_{\{A,C,AC\}}$)**.** *If* $\nabla \vdash s \approx_{\{A,C,AC\}} t$ *then* $\nabla \vdash \pi \cdot s \approx_{\{A,C,AC\}} \pi \cdot t$*.*

Proof. Equivariance follows by induction in the inference rules of $\approx_{\{A,C,AC\}}$, its formalisation is in file `AACC_Equiv.v`, Lem. `aacc_equivariance`. For the case of abstractions, specifically for the case of the rule ($\approx_\alpha$ [**ab**]), Lem. 4.3 is required; indeed, when one has $\nabla \vdash [a]s' \approx_{\{A,C,AC\}} [b]t'$, initially it is necessary to prove that $\nabla \vdash \pi \cdot s' \approx_{\{A,C,AC\}} \pi \cdot ((a\ b) \cdot t')$ and $\nabla \vdash \pi \cdot ((a\ b) \cdot t') \approx_\alpha (\pi \cdot a\ \ \pi \cdot b) \cdot (\pi \cdot t')$ and then apply that lemma to obtain $\nabla \vdash \pi \cdot s' \approx_{\{A,C,AC\}} (\pi \cdot a\ \ \pi \cdot b) \cdot (\pi \cdot t')$. □

Remark 4.3. *Lems. 4.2 to 4.5 that are proved for* $\approx_{\{A,C,AC\}}$ *are not automatically inherited for the specific cases of* $\approx_{\alpha,A}$*,* $\approx_{\alpha,C}$ *and* $\approx_{\alpha,AC}$*. These properties are formalised in particular for* $\approx_{\alpha,C}$ *in file* `c_Equiv.v` *of the specification and they are used in Lem. 5.4 and Thm. 5.2. Further references to these lemmas are directed to the corresponding general results.*

Lemma 4.6 (Reflexivity of $\approx_{\{A,C,AC\}}$). $\nabla \vdash t \approx_{\{A,C,AC\}} t$.

Proof. Reflexivity is formalised in file `AACC_Equiv.v`, Lem. `aacc_equiv_refl`, and it is easily proved by induction on t. □

The next lemma generalises the way in which arguments used in the rule ($\approx_\alpha$ **AC**) are combined.

Lemma 4.7 (Combination of AC arguments). *If* $\nabla \vdash t \approx_{\{A,C,AC\}} t'$ *then* $\forall_{(0<i\leq \|t\|_f)} \exists_{(0<j\leq \|t\|_f)} \nabla \vdash t_{(i)_f} \approx_{\{A,C,AC\}} t'_{(j)_f}$ *and* $\nabla \vdash t_{[\star i]_f} \approx_{\{A,C,AC\}} t'_{[\star j]_f}$.

Proof. The formalisation of this result is in file `AACC_Equiv.v`, Lem. `aacc_equiv_TPith_l`, and its proof is by induction on $\|t\|_f$ using simple auxiliary lemmas and properties of the operators $\|t\|_f, t_{(i)_f}$ and $t_{[\star i]_f}$. The proof of the particular case $i = 1$ is explained as follows: $\nabla \vdash t \approx_{\{A,C,AC\}} t' \Rightarrow \exists_{(0<j\leq\|t'\|_f)}, \nabla \vdash t_{(1)_f} \approx_{\{A,C,AC\}} t'_{(j)_f} \;\wedge\; \nabla \vdash t_{[\star 1]_f} \approx_{\{A,C,AC\}} t'_{[\star j]}$. The complicated case happens when $\|t\|_f > 2$: after applying the auxiliary lemma for terms $f\,t$ and $f\,t'$ one obtains for some valid i_0, $\nabla \vdash t_{(1)_f} \approx_{\{A,C,AC\}} t'_{(i_0)_f}$ and $\nabla \vdash f\,t_{[\star 1]_f} \approx_{\{A,C,AC\}} f\,t'_{[\star i_0]_f}$. Notice that if $i = 1$, the result follows trivially. For $i > 1$, induction applies for the terms $t_0 = f\,t_{[\star 1]_f}$ and $t'_0 = f\,t'_{[\star i_0]_f}$ with argument $i_1 = i - 1$. Notice that the IH is given as $\forall_{(\|t_0\|_f < \|t\|_f,\, t'_0,\, 0<i_1\leq\|t_0\|_f)} \exists j_1, \nabla \vdash t_{0(i_1)_f} \approx_{\{A,C,AC\}} t'_{0(j_1)_f}$ and $\nabla \vdash t_{0[\star i_1]_f} \approx_{\{A,C,AC\}} t'_{0[\star j_1]_f}$. Then, applying IH, a witness j is obtained such that, with the pre-conditions: $\|f\,t_{[\star 1]_f}\|_f < \|t\|_f$ and $\nabla \vdash f\,t_{[\star 1]_f} \approx_{\{A,C,AC\}} f\,t'_{[\star i_0]_f}$, one obtains $\nabla \vdash f\,t_{(i)_f} \approx_{\{A,C,AC\}} f\,t'_{(j)_f}$ and $\nabla \vdash f\,t_{[\star (i)]_f} \approx_{\{A,C,AC\}} f\,t'_{[\star j]_f}$. The first pre-condition is solved by an application of the definition of $\|_\|$ and an auxiliary lemma for the operators $\|t\|_f$ and $t_{[\star i]_f}$. The second is exactly the assumption. Then one just needs to consider two cases: $i_0 \leq j_1$ or $i_0 > j_1$. One instantiates j respectively as $j_1 + 1$ or j_1 and concludes using properties of the operators $\|t\|_f, t_{(i)_f}$ and $t_{[\star i]_f}$. □

Lemma 4.8 (Transitivity of $\approx_{\{A,C,AC\}}$). *If* $\nabla \vdash t_1 \approx_{\{A,C,AC\}} t_2$ *and* $\nabla \vdash t_2 \approx_{\{A,C,AC\}} t_3$ *then* $\nabla \vdash t_1 \approx_{\{A,C,AC\}} t_3$.

Proof. The formalisation is in file `AACC_Equiv.v`, Lem. `aacc_equiv_trans`, and it is by induction on the size of the term t_1. The terms t_2 and t_3 are generalised, and inversions from the equational inference rules are applied to both $\nabla \vdash t_1 \approx_{\{A,C,AC\}} t_2$ and $\nabla \vdash t_2 \approx_{\{A,C,AC\}} t_3$. The difficult cases are those of rules ($\approx_\alpha$ [**ab**]) and ($\approx_\alpha$ **A**) or ($\approx_\alpha$ **AC**). For ($\approx_\alpha$ [**ab**]), an interesting subcase is when $a \neq a' \neq a'_0 \neq a$: the premisses are $\nabla \vdash t \approx_{\{A,C,AC\}} (a\,a') \cdot t' \;\wedge\; \nabla \vdash a \# t'$ and $\nabla \vdash t' \approx_{\{A,C,AC\}} (a'\,a'_0) \cdot t'_0 \;\wedge\; \nabla \vdash a'_0 \# t'_0$, the IH is given as $\forall_{(s_1,s_2,s_3)}, |s_1| < |t| \wedge (\nabla \vdash s_1 \approx_{\{A,C,AC\}} s_2 \wedge \nabla \vdash s_2 \approx_{\{A,C,AC\}} s_3) \Rightarrow \nabla \vdash s_1 \approx_{\{A,C,AC\}} s_3$, and one should conclude that $\nabla \vdash [a]t \approx_{\{A,C,AC\}} [a'_0]t'_0$. Applying ($\approx_\alpha$ [**ab**]) it remains to prove that $\nabla \vdash a \# t'_0$ and $\nabla \vdash t \approx_{\{A,C,AC\}} (a\,a'_0) \cdot t'_0$. The former is obtained by freshness

preservation, and the latter by IH with application of Lem. 4.3, equivariance and freshness preservation.

In the case of rules ($\approx_\alpha$ **A**) or ($\approx_\alpha$ **AC**), the following proof context is reached at some point of the formalisation, where for the case of ($\approx_\alpha$ **A**), the indices i and i_0 are equal to 1: the premisses are $\nabla \vdash t_{(1)_f} \approx_{\{A,C,AC\}} t'_{(i)_f} \wedge \nabla \vdash f\, t_{[\star 1]_f} \approx_{\{A,C,AC\}} f\, t'_{[\star i]_f}$, and $\nabla \vdash t'_{(1)_f} \approx_{\{A,C,AC\}} t'_{0(i_0)_f} \wedge \nabla \vdash f\, t'_{[\star 1]_f} \approx_{\{A,C,AC\}} f\, t'_{0[\star i_0]_f}$, the IH is given by $\forall_{(s_1,s_2,s_3)}, |s_1| < |f\, t| \wedge (\nabla \vdash s_1 \approx_{\{A,C,AC\}} s_2 \wedge \nabla \vdash s_2 \approx_{\{A,C,AC\}} s_3) \Rightarrow \nabla \vdash s_1 \approx_{\{A,C,AC\}} s_3$, and one should conclude that $\nabla \vdash f\, t \approx_{\{A,C,AC\}} f\, t'_0$. Applying ($\approx_\alpha$ **A**) and the IH one concludes easily for the case in which $E = A$. When $E = AC$ one uses the Lem. 4.7 and the second premise above, obtaining a third premise: $\exists i_1, \nabla \vdash t'_{(i)_f} \approx_{\{A,C,AC\}} t'_{0(i_1)_f} \wedge \nabla \vdash t'_{[\star i]_f} \approx_{\{A,C,AC\}} t'_{0[\star i_1]_f}$. Then, applying the ($\approx_\alpha$ **AC**) rule instantiated with i_1. The resulting subgoals are $\nabla \vdash t_{(1)_f} \approx_{\{A,C,AC\}} t'_{0(i_1)_f}$ and $\nabla \vdash f\, t_{[\star 1]_f} \approx_{\{A,C,AC\}} f\, t'_{0[\star i_1]_f}$, and from the first and third premises above, both subgoals are solved by application of IH. □

Lemma 4.9 (Symmetry of $\approx_{\{A,C,AC\}}$)**.** *If $\nabla \vdash t \approx_{\{A,C,AC\}} t'$ then $\nabla \vdash t' \approx_{\{A,C,AC\}} t$.*

Proof. Symmetry is easily formalised in file `AACC_Equiv.v`, Lem. `aacc_equiv_sym`, by induction on $\approx_{\{A,C,AC\}}$ applying lemmas 4.3, 4.6 and 4.8, freshness preservation and equivariance.

In particular, the use of the Lem. 4.8 is crucial: in the ($\approx_\alpha$ **[ab]**) case one should prove that $\nabla \vdash [b]t' \approx_{\{A,C,AC\}} [a]t$ having as hypotheses $\nabla \vdash t \approx_{\{A,C,AC\}} (a\, b) \cdot t'$ and $\nabla \vdash a \# t'$, with IH $\nabla \vdash (a\, b) \cdot t' \approx_{\{A,C,AC\}} t$. Then, Lem. 4.8 is applied twice instantiating t_2 as $(a\, b) \cdot t$ and as $(a\, b) \oplus (a\, b) \cdot t'$, this allows the use of Lems. 4.3 (with properties of $\approx_\alpha$) and equivariance to conclude.

The following corollary is used to derive, from Lems. 4.6, 4.8 and 4.9 the proofs that $\approx_{\alpha,A}$, $\approx_{\alpha,C}$, $\approx_{\alpha,AC}$ and $\approx_{\{A,C,AC\}}$ are indeed equivalence relations. Remember the parameterisation used in the specification, in which **equiv** with argument set of indices $\{0\}$, $\{1\}$, $\{2\}$ and $\{0,1,2\}$ correspond respectively to $\approx_{\alpha,A}$, $\approx_{\alpha,AC}$, $\approx_{\alpha,C}$ and $\approx_{\{A,C,AC\}}$. □

Corollary 4.1 (Equivalence of **equiv**(S), for $S \subseteq \{0,1,2\}$)**.** *For $S \subseteq \{0,1,2\}$,* ***equiv***(S) *is also an equivalence relation.*

Proof. The formalisation is in file `Equiv.v`, Lem. `subset_equivalence`, and is obtained by the manipulation of the superscripts in $S^{-1} = \{0,1,2\} - S$. For a general equivalence problem **equiv**(S)(∇, t_1, t_2), one replaces all superscripts of the operators in the terms t_1 and t_2 inside the set S^{-1} by new ones that neither belong to $\{0,1,2\}$ nor occur in t_1 and t_2 obtaining respectively t'_1 and t'_2. Then, by induction on the inference rules for **equiv**, one easily proves that **equiv**(S)$(\nabla, t_1, t_2) \Leftrightarrow$ **equiv**(S)$(\nabla, t'_1, t'_2) \Leftrightarrow$ **equiv**($\{0,1,2\}$)(∇, t'_1, t'_2). Thus, using that **equiv**($\{0,1,2\}$) is an equivalence relation one concludes. □

4.4 A naive implementation of the $\approx_{\{A,C,AC\}}$ equality-checking algorithm

An algorithm to check a problem $\langle\nabla, P\rangle$ modulo A/C/AC is defined by the recursive function `Check` given in Algorithm 1. This algorithm simply distinguishes the cases that should be considered to deal with A/C/AC function symbols. For instance, assuming that $+$ is an AC function symbol, the equality $\nabla \vdash +\langle s, +\langle t, [a]X\rangle\rangle \approx_{\alpha,AC} +\langle +\langle [b]X, s\rangle, t\rangle$ holds whenever the freshness constraints $a \# X$, $b \# X$ belong to ∇. Equational problems will be written as pairs $\langle\nabla, P\rangle$, where ∇ is a set of freshness constraints and P a set of equations. For simplicity, when no confusion arises brackets will be omitted.

Example 4.5. *Assuming $\nabla = \{a \# X, b \# X\}$ and using Algorithm 1, where g is a syntactic function symbol, it follows that*

$$\begin{aligned}
&\nabla, \{[a]g\langle\bar{a}, X\rangle \approx [b]g\langle\bar{b}, X\rangle\} \Longrightarrow_{Line\,12} \nabla, \{g\langle\bar{a}, X\rangle \approx (a\,b)\cdot g\langle\bar{b}, X\rangle\} \\
&= \nabla, \{g\langle\bar{a}, X\rangle \approx g\langle\bar{a}, (a\,b).X\rangle\} \Longrightarrow_{Line\,42} \nabla, \{\langle\bar{a}, X\rangle \approx \langle\bar{a}, (a\,b)X\rangle\} \\
&\Longrightarrow_{Line\,8} \nabla, \{\bar{a} \approx \bar{a}, X \approx (a\,b).X\} \Longrightarrow_{Line\,6,16} \nabla, \emptyset \Longrightarrow_{Line\,2} \top
\end{aligned}$$

Example 4.6. *Consider the problem $\langle\emptyset, \{f_k^A\langle\bar{a}, \langle\bar{b}, [a]\bar{a}\rangle\rangle \approx f_k^A\langle\langle\bar{a}, \bar{b}\rangle, [b]\bar{b}\rangle\}\rangle$.*

$$\begin{aligned}
&\emptyset, \{f_k^A\langle\bar{a}, \langle\bar{b}, [a]\bar{a}\rangle\rangle \approx f_k^A\langle\langle\bar{a}, \bar{b}\rangle, [b]\bar{b}\rangle\} \\
&\Longrightarrow_{Line\,19,21} \emptyset, \{f_k^A\langle\bar{b}, [a]\bar{a}\rangle \approx f_k^A\langle\bar{b}, [b]\bar{b}\rangle\}, \quad \textit{since } \texttt{Check}(\emptyset, \bar{a} \approx \bar{a}) \ \textit{(Line 20)} \\
&\Longrightarrow_{Line\,19,21} \emptyset, \{f_k^A[a]\bar{a} \approx f_k^A[b]\bar{b}\}, \quad \textit{since } \texttt{Check}(\emptyset, \bar{b} \approx \bar{b}) \ \textit{(Line 20)} \\
&\Longrightarrow_{Line\,19,21} \emptyset, \{\langle\rangle \approx \langle\rangle\}, \quad \textit{since } \texttt{Check}(\emptyset, [a]\bar{a} \approx [b]\bar{b}) \ \textit{(Line 20)} \\
&\Longrightarrow_{Line\,5,2} \top
\end{aligned}$$

Notice that, in the third step above, one calls `Check`$(\emptyset, \langle\rangle \approx \langle\rangle)$ *since* $(f_k^A[a]\bar{a})_{[\star 1]_{f_k^A}} = (f_k^A[b]\bar{b})_{[\star 1]_{f_k^A}} = \langle\rangle$ *(see Figs. 4.1 to 4.3).*

Example 4.7. *Consider the problem $\langle\emptyset, \{f_k^C\langle\bar{b}, [a]\bar{a}\rangle \approx f_k^C\langle[b]\bar{b}, \bar{b}\rangle\}\rangle$.*

$$\begin{aligned}
&\emptyset, \{f_k^C\langle\bar{b}, [a]\bar{a}\rangle \approx f_k^C\langle[b]\bar{b}, \bar{b}\rangle\} \\
&\Longrightarrow_{Line\,29} \emptyset, \{\bar{b} \approx \bar{b},\ [a]\bar{a} \approx [b]\bar{b}\}, \\
&\qquad\qquad \textit{since } \texttt{Check}(\emptyset, \{\bar{b} \approx [b]\bar{b},\ [a]\bar{a} \approx \bar{b}\}) = \bot \ \textit{(L. 27)} \\
&\Longrightarrow_{Line\,6} \emptyset, \{[a]\bar{a} \approx [b]\bar{b}\} \Longrightarrow_{Line\,10,12,2} \top
\end{aligned}$$

Example 4.8. *Let $*$ and f be a C and a syntactic function symbol respectively. Consider the problem $\langle\nabla, \{[a]f\langle[b](X * \bar{b}), Y\rangle \approx [b]f\langle[a](\bar{a} * X), Y\rangle\}\rangle$ and assume that $\{a\#X, b\#X,$*

Algorithm 1 Checking α-equivalence modulo A, C and AC

```
1: function Check(∇, P)
2:    if P = ∅ then ⊤
3:    else let s ≈ t ∈ P and P' = P \ {s ≈ t} in
4:     case s ≈ t of
5:       ⟨⟩ ≈ ⟨⟩ :  Check(∇, P')                                   // rule (≈_α ⟨⟩)
6:       ā ≈ ā :  Check(∇, P')                                     // rule (≈_α atom)
7:       ⟨s_1, s_2⟩ ≈ ⟨t_1, t_2⟩ :
8:        Check(∇, {s_1 ≈ t_1, s_2 ≈ t_2} ∪ P')                    // rule (≈_α pair)
9:       [a]s' ≈ [a]t' :  Check(∇, {s' ≈ t'} ∪ P')                 // rule (≈_α [aa])
10:      [a]s' ≈ [b]t' :                                           // rule (≈_α [ab])
11:        if ∇ ⊢ a # t' then
12:            Check(∇, {s' ≈ (a b) · t'} ∪ P')
13:        else ⊥
14:        end if
15:      π.X ≈ π'.X :                                              // rule (≈_α var)
16:        if For all a ∈ ds(π, π'), a # X ∈ ∇ then Check(∇, P')
17:        else ⊥
18:        end if
19:      f_k^A s' ≈ f_k^A t' :                                     // rule (≈_α A)
20:        if Check(∇, {(f_k^A s')_(1)_{f_k^A} ≈ (f_k^A t')_(1)_{f_k^A}}) then
21:            if Check(∇, {(f_k^A s)_[⋆1]_{f_k^A} ≈ (f_k^A t)_[⋆1]_{f_k^A}}) then Check(∇, P')
22:            else ⊥
23:            end if
24:        else ⊥
25:        end if
26:      f_k^C ⟨s_0, s_1⟩ ≈ f_k^C ⟨t_0, t_1⟩ :                     // rule (≈_α C)
27:        if Check(∇, {s_0 ≈ t_0, s_1 ≈ t_1}) then  Check(∇, P')
28:        else
29:            if Check(∇, {s_0 ≈ t_1, s_1 ≈ t_0}) then  Check(∇, P')
30:            else ⊥
31:            end if
32:        end if
33:      f_k^AC s' ≈ f_k^AC t' :                                   // rule (≈_α AC)
34:        let Branch(i) :=
35:        if  Check(∇, {(f_k^AC s)_(1)_{f_k^AC} ≈ (f_k^AC t)_(i)_{f_k^AC}}) then
36:            Check(∇, {(f_k^AC s)_[⋆1]_{f_k^AC}) ≈ (f_k^AC t)_[⋆i]_{f_k^AC})})
37:        else ⊥
38:        end if  in
39:        if Iter(Branch, 1, ||f_k^AC t||) then Check(∇, P')
40:        else ⊥
41:        end if
42:      f s' ≈ f t' :  Check(∇, {s' ≈ t'} ∪ P')                   // rule (≈_α app)
43:      _ _ :  ⊥                                                  // otherwise
44:    end if
45: end function
```

$a\#Y, b\#Y\} \subseteq \nabla$. *The algorithm* `Check` *proceeds as follows:*

$$\begin{aligned}
&\nabla, \{[a]f\langle[b](X * \bar{b}), Y\rangle \approx [b]f\langle[a](\bar{a} * X), Y\rangle\}\\
&\Longrightarrow_{Line\,10,11,12} \nabla, \{f\langle[b](X * \bar{b}), Y\rangle \approx f\langle[b](\bar{b} * (a\,b).X), (a\,b).Y\rangle\}\\
&\qquad\qquad\qquad \textit{since } \nabla \vdash a \# f\langle[a](\bar{a} * X), Y\rangle\\
&\Longrightarrow_{Line\,42} \nabla, \{\langle[b](X * \bar{b}), Y\rangle \approx \langle[b](\bar{b} * (a\,b).X), (a\,b).Y\rangle\}\\
&\Longrightarrow_{Line\,7,8} \nabla, \{[b](X * \bar{b}) \approx [b](\bar{b} * (a\,b).X),\ Y \approx (a\,b).Y\}\\
&\Longrightarrow_{Line\,9} \nabla, \{X * \bar{b} \approx \bar{b} * (a\,b).X,\ Y \approx (a\,b).Y\}\\
&\Longrightarrow_{Line\,26,27,28,29} \nabla, \{X \approx (a\,b).X,\ \bar{b} \approx \bar{b},\ Y \approx (a\,b).Y\}\\
&\qquad\qquad\qquad \textit{since } \texttt{Check}(\nabla, \{X \approx \bar{b},\ \bar{b} \approx (a\,b).X,\ Y \approx (a\,b).Y\}) = \bot\\
&\Longrightarrow_{Line\,15,16} \nabla, \{\bar{b} \approx \bar{b},\ Y \approx (a\,b).Y\},\ \ \textit{since } ds(id, (a\,b))\#X \subseteq \nabla\\
&\Longrightarrow_{Line\,6} \nabla, \{Y \approx (a\,b).Y\}\\
&\Longrightarrow_{Line\,15,16,2} \top,\ \ \textit{since } ds(id, (a\,b))\#Y \subseteq \nabla
\end{aligned}$$

Lines 33 to 41 in Algorithm 1 deal with the case of equations headed by AC-function symbols. The algorithm checks equality of the first argument on the *lhs* of the equation with the first, second, third, etc. of the *rhs* until this check succeeds and then recursively checks equality of the whole term obtained by eliminating the first argument on the *lhs* and the successful i^{th} argument on the *rhs*; otherwise, the search continues recursively increasing i^{th} until it exceeds the number of arguments of the heading function symbol in $f_k^{AC}s'$, and the check fails. This is specified in Coq through a simple recursive implementation of an iteration function `iter` (see Fig. 4.9).

Example 4.9. *Consider the problem* $\langle\nabla, \{f_k^{AC}([a]a, \pi.X) \approx f_k^{AC}(\pi'.X, [b]b)\}\rangle$ *and assume that* $ds(\pi, \pi')\#X \subseteq \nabla$. *The algorithm* `Check` *proceeds as follows:*

$$\begin{aligned}
&\nabla, \{f_k^{AC}([a]\bar{a}, \pi.X) \approx f_k^{AC}(\pi'.X, [b]\bar{b})\}\\
&\Longrightarrow_{Line\,36} \texttt{Check}(\nabla, \{[a]\bar{a} \approx [b]\bar{b}\})\\
&\qquad\qquad\qquad \textit{since } \texttt{Check}(\nabla, \{[a]\bar{a} \approx \pi'.X\}) = \bot \textit{ (Line 35, 37)}\\
&\Longrightarrow_{Line\,36} \nabla, \{f_k^{AC}\ \pi.X \approx f_k^{AC}\ \pi'.X\},\\
&\qquad\qquad\qquad \textit{since } \texttt{Check}(\nabla, \{[a]\bar{a} \approx [b]\bar{b}\}) = \top \textit{ (Line 35, 10, 6, 2)}\\
&\Longrightarrow_{Line\,35} \nabla, \{\langle\rangle \approx \langle\rangle\},\\
&\qquad\qquad\qquad \textit{since } \texttt{Check}(\nabla, \pi.X \approx \pi'.X) = \top \textit{ (Line 15)}\\
&\Longrightarrow_{Line\,5,\,2} \top
\end{aligned}$$

Note that the proposed algorithm can check validity of α-equivalence constraints modulo A and/or C and/or AC ($\approx_{\{A,C,AC\}}$) with multiple occurrences of function symbols, some that might be A and some C and some other AC, all at once. This is due to the fact that there are no interactions between A, C, and AC symbols since distributive properties are not considered.

4.5 Automatic code extraction

To obtain executable code from the inductive definition of **equiv** (Fig. 4.8), an equivalent recursive function, called equiv_rec (Fig. 4.11), has been specified in file `AACC_Equiv_rec.v`. This recursive function applies the α, A, C and AC equivalence inference rules. Since the applications of rules ($\approx_\alpha$ **A**) and ($\approx_\alpha$ **AC**) require recursive applications of inference rules to equations over terms that are not subterms of the input equation, the standard `Fixpoint` definition of Coq is not applicable. Thus, a more powerful recursive combinator was used that allows well-founded recursion, having as the decreasing measure the size of the terms $|t|$. Also a recursive version of **fresh** (Fig. 4.8), called fresh_rec (Fig. 4.10), was defined in file `AACC_Equiv_rec.v` using the Coq `Fixpoint` combinator. This recursive function is used in the specification of equiv_rec (Fig. 4.11).

In equiv_rec, the expression "equiv_rec C s t `by rec` (term_size s) lt" specifies that the recursion is performed with the decreasing measure term_size s. Then each recursive call generates a proof obligation, checking that this measure is in fact decreasing. For instance, in the pattern matching "equiv_rec C (Ab a u) (Ab b v) := ...", the recursive call equiv_rec C u v is executed in such way that term_size (Ab a u) $>$ term_size u. The interesting recursion case occurs in the pattern matching "equiv_rec C (Fc 1 n u) (Fc E n' v) := ...", that specifies the application of rule ($\approx_\alpha$ **AC**). In this case the auxiliary iterator iter (Fig. 4.9) is used and the recursive calls are performed in equiv_rec C (TPith 1 u 1 n) (TPith i v 1 n) and equiv_rec C (TPithdel 1 (Fc 1 n u) 1 n) (TPithdel i (Fc 1 n v) 1 n). The proof that the specified measure decreases uses Lems. `term_size_TPith` and `term_size_TPithdel`, of file `Tuples.v`, showing that:

term_size (Fc 1 n u) $>$ term_size (TPith 1 u 1 n) and

term_size (Fc 1 n u) $>$ term_size (TPithdel 1 (Fc 1 n u) 1 n).

The recursive calls generated by the application of rule ($\approx_\alpha$ **AC**) need to be specified through an auxiliary (recursively defined) iteration operator iter (see Fig. 4.9) that makes the application of the fixed point Coq mechanisms difficult.

For the verification of equiv_rec, the techniques given by Sozeau [66] were adopted. The applied strategy uses a new type of definition named `Equations` that allows the simultaneous use of well-founded and iterative recursion, automatically generating the

```
Fixpoint iter (P : nat → bool) (i j : nat) {struct j} : bool :=
 match j with
 | 0 ⇒ false
 | S j0 ⇒ if P i
           then true
           else iter P (S i) j0
 end .
```

Figure 4.9: Specification of the recursive iterator iter

```
Fixpoint fresh_rec (C : Context) (a : Atom) (t : term) :=
 match t with

  | «» ⇒ true

  | %a0 ⇒ if eq_atom_rec a a0
           then false
           else true

  | [a0]^s ⇒ if eq_atom_rec a a0
              then true
              else fresh_rec C a s

  | <|u, v|> ⇒ fresh_rec C a u && fresh_rec C a v

  | Fc m n s ⇒ fresh_rec C a s

  | pi|.X ⇒ if in_context_dec (!pi $ a, X) C
             then true
             else false

  end.
```

Figure 4.10: Specification of fresh_rec

```
Equations equiv_rec (C: Context) (s t : term) : bool :=
equiv_rec C s t by rec (term_size s) lt :=

equiv_rec C Ut Ut := true;

equiv_rec C (At a) (At b) :=
 eq_atom_rec a b;

equiv_rec C (Ab a u) (Ab b v) :=
 if eq_atom_rec a b
 then equiv_rec C u v
 else
  if fresh_rec C a v
  then equiv_rec C u (((a,b)::nil)@v)
  else false;

equiv_rec C (Pr u0 u1) (Pr v0 v1) :=
 if equiv_rec C u0 v0
 then equiv_rec C u1 v1
 else false;

equiv_rec C (Fc 0 n u) (Fc E n' v) :=
 if eq_nat_rec n n' && eq_nat_rec 0 E
 then
  if equiv_rec C (TPith 1 u 0 n) (TPith 1 v 0 n)
  then equiv_rec C (TPithdel 1 (Fc 0 n u) 0 n)
                   (TPithdel 1 (Fc 0 n v) 0 n)
  else false
 else false;

equiv_rec C (Fc 1 n u) (Fc E n' v) :=
 if eq_nat_rec n n' && eq_nat_rec 1 E
 then

  let branch (i : nat) :=
   if equiv_rec C (TPith 1 u 1 n) (TPith i v 1 n)
   then equiv_rec C (TPithdel 1 (Fc 1 n u) 1 n)
                    (TPithdel i (Fc 1 n v) 1 n)
   else false
  in iter branch 1 (TPlength v 1 n)

 else false;

equiv_rec C (Fc 2 n (Pr u0 u1))
            (Fc E n' (Pr v0 v1)) :=
 if eq_nat_rec n n' && eq_nat_rec 2 E
 then
  if
   if equiv_rec C u0 v0
   then equiv_rec C u1 v1
   else false
  then true
  else
   if equiv_rec C u0 v1
   then equiv_rec C u1 v0
   else false
 else false;

equiv_rec C (Fc E n u) (Fc E' n' v) :=
 if eq_nat_rec E E' && eq_nat_rec n n'
 then equiv_rec C u v
 else false;

equiv_rec C (Su pi X) (Su pi' Y) :=
 if eq_var_rec X Y
 then sub_context_rec
       (fresh_context (disgr pi pi') X) C
 else false;

equiv_rec C _ _ := false.
```

Figure 4.11: Specification of equiv_rec

simplification lemmas required in the inductive formalisation of the correctness of equiv_rec. Other techniques are available in Coq for defining more complex recursive functions, such as the `Program Fixpoint` definition [65]. This allows the specification of functions with well-founded and iterative recursion, but the simplification lemmas are not automatically generated.

Another way of building recursive functions from inductive definitions in Coq is proposed in the recent work by Larchey-Wendling and Monin [47]. The strategy consists in defining first the graph of the inductive definition; then, one proves termination, functionality and totality (over a specific domain) of the graph. This strategy also allows the use of Coq code extraction, but the process of constructing the recursive definition is not as straightforward as the `Equation` approach applied in the present work.

The following lemma verifies the recursive function equiv_rec stating its equivalence to the inductive definition $\approx_{\{A,C,AC\}}$.

Lemma 4.10 (Correctness of equiv_rec)**.** $\nabla \vdash s \approx_{\{A,C,AC\}} t$ *iff* $(\mathsf{equiv_rec}\, \nabla\, s\ t = \mathsf{true})$.

Proof. The formalisation is in file `AACC_Equiv_rec.v`, Lem. `equiv_rec_eq`. Necessity is proved by induction on the derivation rules of $\nabla \vdash s \approx_{\{A,C,AC\}} t$. Each case uses a previous result based on an automatically generated simplification lemma for equiv_rec. For instance, for the case of rule ($\approx_\alpha$ **[ab]**) the hypotheses are $a \neq b$, $\nabla \vdash u \approx_{\{A,C,AC\}} (a\, b) \cdot v$ and $\nabla \vdash a \# v$, and IH is given by $(\mathsf{equiv_rec}\ \ \nabla\, u\ ((a\, b)\cdot v) = \mathsf{true})$. A previous result allows to rewrite the objective $(\mathsf{equiv_rec}\, \nabla\ ([a]u)\ \ ([b]v)\ =\ \mathsf{true})$ to $((\mathsf{fresh_rec}\, \nabla\, a\, v\ =\ \mathsf{true})\ \wedge$ $(\mathsf{equiv_rec}\ \ \nabla\, u\ ((a\, b)\cdot v) = \mathsf{true}))$. After rewriting IH, one concludes using a previous correctness lemma for fresh_rec which states that $\nabla \vdash a \# v$ if and only if $(\mathsf{fresh_rec}\, \nabla\, a\, v = \mathsf{true})$.

Sufficiency is reached by induction on the size of s and case analysis over s and t. One of the non-trivial cases is when both terms s and t are headed by the same AC function symbol f. In this case the hypotheses are $l = |u| + 1$ and $(\mathsf{equiv_rec}\, \nabla\, (f\, u)\ (f\, v) = \mathsf{true})$, and IH is given by $\forall m, m < l \Rightarrow \forall u_0, \forall v_0$, $(\mathsf{equiv_rec}\, \nabla\, u_0\ v_0 = \mathsf{true}) \Rightarrow \nabla \vdash s_0 \approx_{\{A,C,AC\}} v_0$. A previous result allows rewriting the premisse $(\mathsf{equiv_rec}\, \nabla\, (f\, u)\ \ (f\, v)\ =\ \mathsf{true})$ to $\exists i$, $(\mathsf{equiv_rec}\, \nabla\, u_{(1)}\ \ v_{(i)}\ =\ \mathsf{true})\ \wedge\ (\mathsf{equiv_rec}\ \ \nabla\, (f\, u)_{[\star 1]}\ \ (f\, v)_{[\star i]}\ =\ \mathsf{true})$. Splitting this conjunction and applying IH in both generated premisses results in two new hypotheses $\nabla \vdash u_{(1)} \approx_{\{A,C,AC\}} v_{(i)}$ and $\nabla \vdash (f\, u)_{[\star 1]} \approx_{\{A,C,AC\}} (f\, v)_{[\star i]}$. Notice that in the applications of IH the condition $m < l$ needs to be verified through arithmetic properties of the operators $|_|$, $\|_\|$, $_{(_)}$ and $_{[\star _]}$. Then, one concludes with an application of rule ($\approx_\alpha$ **AC**). □

Executable OCaml code was automatically extracted from equiv_rec, using the built-in code extraction mechanism of Coq. The generated code is available as the file `Impl/Original_Generated_Equiv.ml` inside the specification folder.

The extracted code uses Coq naturals to represent atoms, variables and indices of function symbols: n is represented as n applications of the successor constructor `S` to zero `O`. For execution tests (see Subsec. 4.6), an adjusted version of the generated code that just replaces Coq naturals by OCaml integers, was used. The adjusted code is available as the file `Impl/Adjusted_Generated_Equiv.ml`.

In addition to the extracted naive algorithm, a manually generated one was implemented that essentially improves the representation of terms by flattening arguments of A and AC function symbols, and by a simpler analysis for the AC case than the one given by the Algorithm 1. By flattening terms, application of the selection and deletion operators, $_\,(_)_f$ and $_\,[\star\,_]_f$, over arguments of A and AC operators is avoided and arguments of these operator are then sequentially analysed.

The improved analysis of the AC case is inspired by the translation to the problem of finding a perfect matching in a bipartite graph given in [17] initially proposed for solving AC-matching. For a given equational problem over terms headed by an AC function symbol, a graph whose vertices are labelled by the arguments of the AC function in the *lhs* and *rhs* of the equation is built. There is an edge between two vertices labelled by arguments in opposite sides of the equation if they match. A perfect matching in the bipartite graph is a solution for the initial problem. In the case of AC-equational check, for solving the flattened equational problem $f(s_1, \ldots, s_k) \approx f(t_1, \ldots, t_k)$, if the answer is positive, and if s_i is known to be equivalent to t_l and t_j, there should be another *lhs* argument s_m that is also equivalent to these three arguments. Thus, the improved implementation essentially searches imperatively for *rhs* arguments that are equivalent to the first, second and so on *lhs* arguments (see the complexity analysis given in Thm. 4.1). This implementation is available as files `Impl/Basics.ml` and `Impl/Improved_Equiv.ml`.

4.6 Execution tests

Experiments were performed with the extracted and improved algorithms, over an iMAC server with 16GB of RAM and with a processor Intel Xeon CPU, model W3530 2.80GHz, providing randomly recursively generated ground equational problems as inputs. Terms were generated using only tuples with arguments associated to the right as arguments for A and AC function symbols. Also, subterms headed by an associative function symbol, say either f_k^A or f_k^{AC}, do not have arguments headed by the same function symbol. For example, $f_0^A\langle \bar{a}, \langle \bar{b}, \langle \bar{c}, \langle \bar{d}, \bar{e}\rangle\rangle\rangle\rangle$ and $f_0^{AC}\langle f_0^A\langle \bar{a}, \bar{b}\rangle, f_4^A\langle \bar{c}, f_0^{AC}\langle \bar{d}, \bar{e}\rangle\rangle\rangle$ are in this class of terms.

Although flattened terms mitigate the required effort for manipulation of arguments of associative operators, it should be stressed that the adequate data structure to deal with flattened arguments of these operators should allow random access as arrays and

sequences do. Also, having only ground terms mitigates the negative effects of inefficient procedures for dealing with permutation operations, such as queries about their support, inversion and composition, which are used in the extracted algorithm for application of rule ($\approx_\alpha$ **var**).

The number of different syntactic, A, C and AC symbols were restricted to ten (each class), and atoms were chosen among a set of ten thousand. In the randomly recursive generation of an equational problem, whenever abstractions are generated as subterms, the choice of different atoms in the *lhs* and in the *rhs* of the equation is enforced. This strategy was adopted to prioritise the use of rule ($\approx_\alpha$ [**ab**]) against ($\approx_\alpha$ [**aa**]), since the latter has lower cost. In this case, to guarantee that the equality-checking results in `true`, avoiding collisions and ensuring the condition $\nabla \vdash a \# t'$ of the rule ($\approx_\alpha$ [**ab**]), a list of used atoms that are not allowed to occur in the body of the abstractions is kept.

Four different sets of input problems were generated. The first one, uses only abstractions and syntactic function symbols; the second uses also A symbols; the third uses syntactic, A and C symbols; and, the fourth allows all four kinds of symbols. For each set, problems with positive answer of sizes from 100 to 10000, with intervals of length 100, were generated; for each size twenty five different problems were generated. Each problem was tested with both, the extracted and the improved implementations.

For the improved algorithm inputs were translated by representing tuples as lists. The cost of this syntactic translation was not considered, but of course the time required for the flattening operation was considered in the evaluation. This operation consists in the elimination of nested occurrences of A and AC function symbols.

Time performance of the experiments is given in Figs. 4.12 to 4.15. Plots in each figure correspond to tests with the same set of inputs, and *lhs* and *rhs* plots correspond respectively to experiments with the extracted and the improved implementations. These figures plot also the regressions computed using the generalised additive model (GAM) generated using the `ggplot2` library of R. For all sets of inputs the performance of the improved implementation was better than the performance of the extracted implementation.

As expected, from the required uniformity of known worst case inputs, which even for α-syntactic problems results in exponential running time for the extracted algorithm, in all cases it could be observed that only a few isolated cases present running time much higher than the regression curve.

The α-syntactic case (Fig. 4.12) shows a linear behaviour for both implementations. Notice that the improved implementation was, approximately, 15% faster than the extracted one. This can be explained by the fact that the recursive calls in nested tuples are more time consuming than operating on more efficient data structures, such as lists, used to represent arguments of function symbols.

Adding A and C-function symbols (respectively, Fig. 4.13 and Fig. 4.14) increases the running time as expected, but the relative behaviour is very similar. In this case the performance of the improved implementation was approximately thirty times faster than the performance of the extracted algorithm. This could be explained since the bottleneck of the extracted algorithm resides in the inefficient manipulation of permutation operations as well as inefficient data structure for the representation of function arguments, incrementing in this way the running time required for the analysis of A and C operators over problems randomly generated as explained. The small effect caused by the addition of C function symbols, for both implementations, is explained by the fact that the inputs with high cost attributed to the C checking are artificial and with low probability of occurrence in the random input generator.

Substantial additional running time is required if AC-symbols are included (Fig. 4.15). Indeed, in the extracted algorithm, one moves from milliseconds to seconds. This is explained because of the required exhaustive application of the linear running time implementations for the operators $\| _ \|_f$, $_ \, (_)_f$ and $_ \, [\star _]_f$ (see Figs. 4.1, 4.2 and 4.3) used to deal with AC terms. This happens since these operators were implemented straightforwardly over tuples that are in fact built as combinations of nominal pairs. In addition, the approach adopted in the improved implementation is more efficient regarding recursive computation of equality-checking for arguments of AC operators. The advantages of this approach can be observed in Fig. 4.15 where the maximum execution time was less than 3 milliseconds for inputs of size around five thousand and less than one hundredth of a second for inputs of size around ten thousand.

Accentuation of the curves for inputs in the fourth set of size greater than 8300, for both implementations can be explained by memory saturation; larger terms could be treated by improving the data structures used for representing arguments of nominal AC operators.

4.7 Upper bounds

This section is concerned with providing upper bounds to the problem of checking the validity of α-equivalence constraints in the presence of A, C and AC function symbols, by applying simplification rules. Several techniques from [24], originally implemented to deal polynomially with nominal α-equivalence as well as with nominal matching, should be adopted in order to obtain efficient algorithms. Among these techniques, it is necessary to use adequate data structures, such as trees for terms and random access structures for maintaining and answering in constant time queries about the images of permutations and their inverses, as well as for updating compositions of swappings and permutations (and

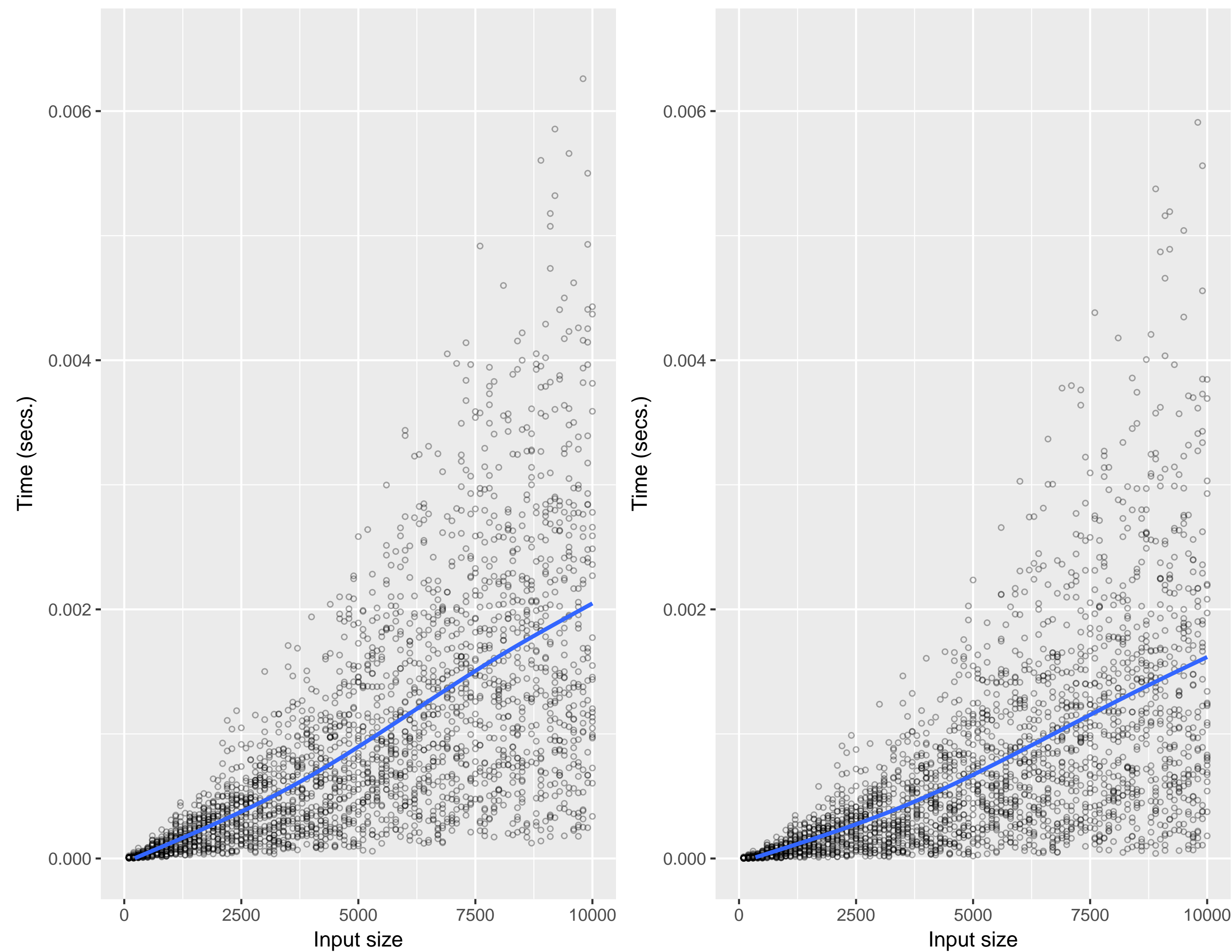

Figure 4.12: Tests with only α operators. The same scale was used in the left- and right-hand side plots

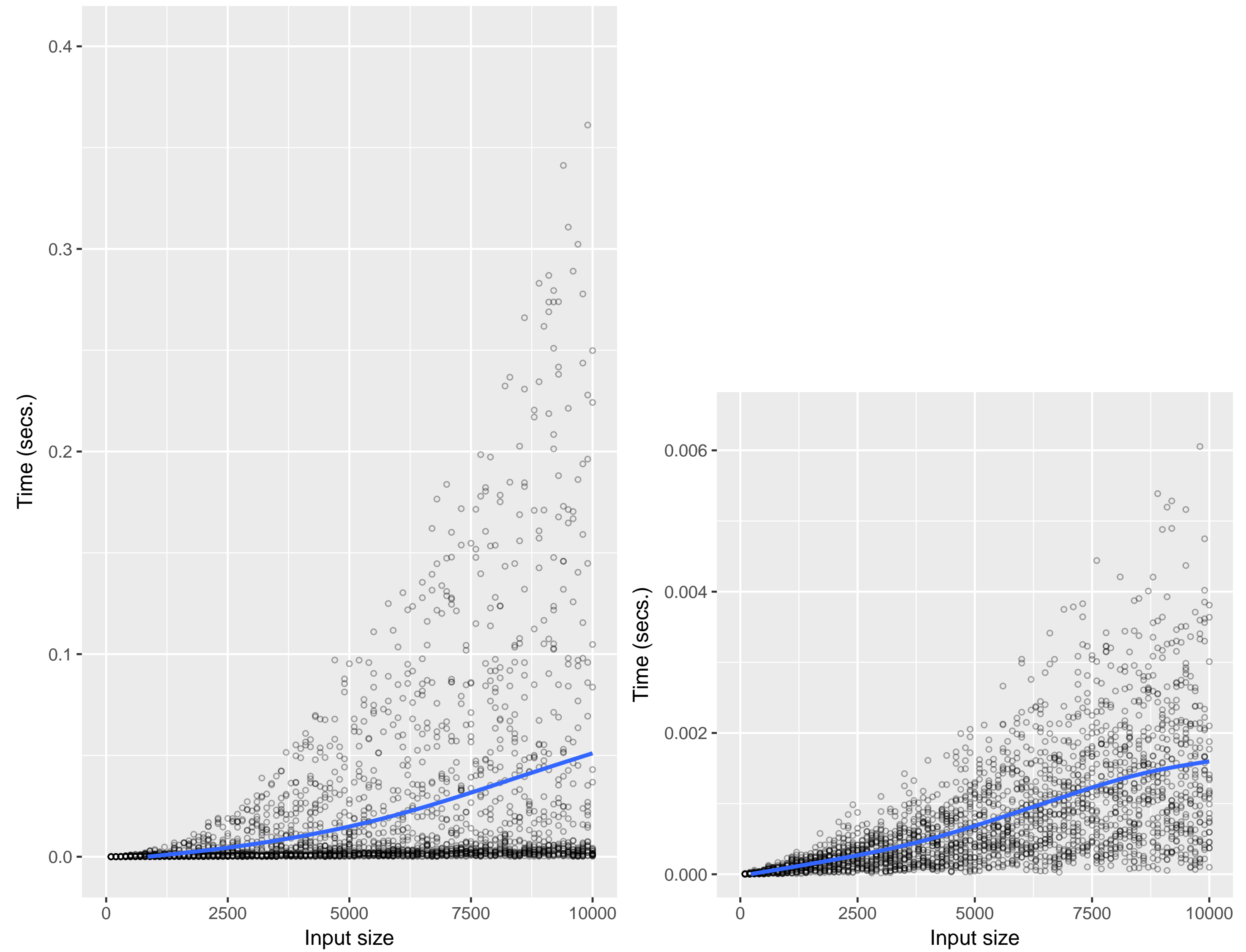

Figure 4.13: Tests with α-A operators. The scale of the time-axis on the right-hand side is 33.3 times bigger than on the left-hand side

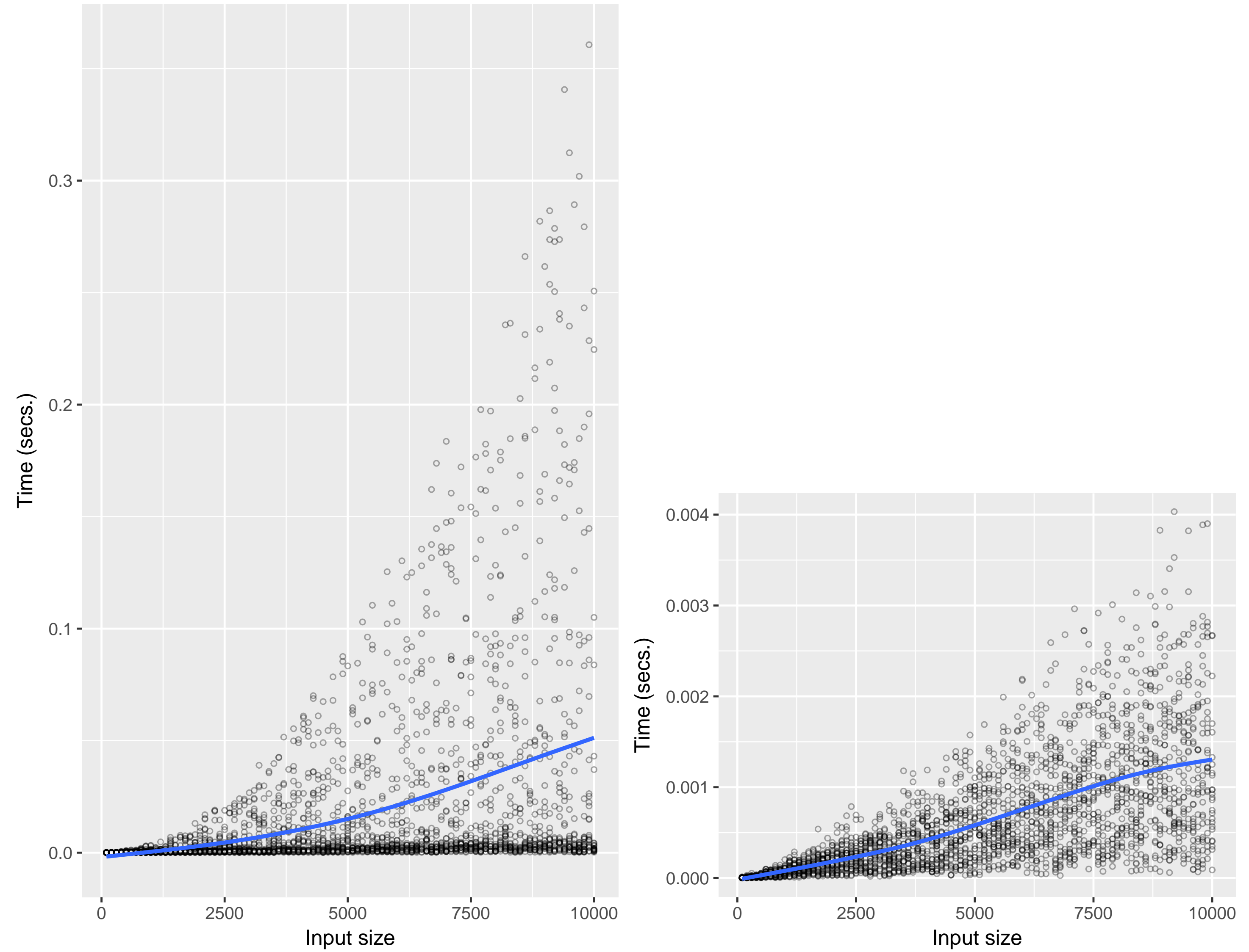

Figure 4.14: Tests with α-A-C operators. The scale of the time-axis on the right-hand side is 37.5 times bigger than on the left-hand side

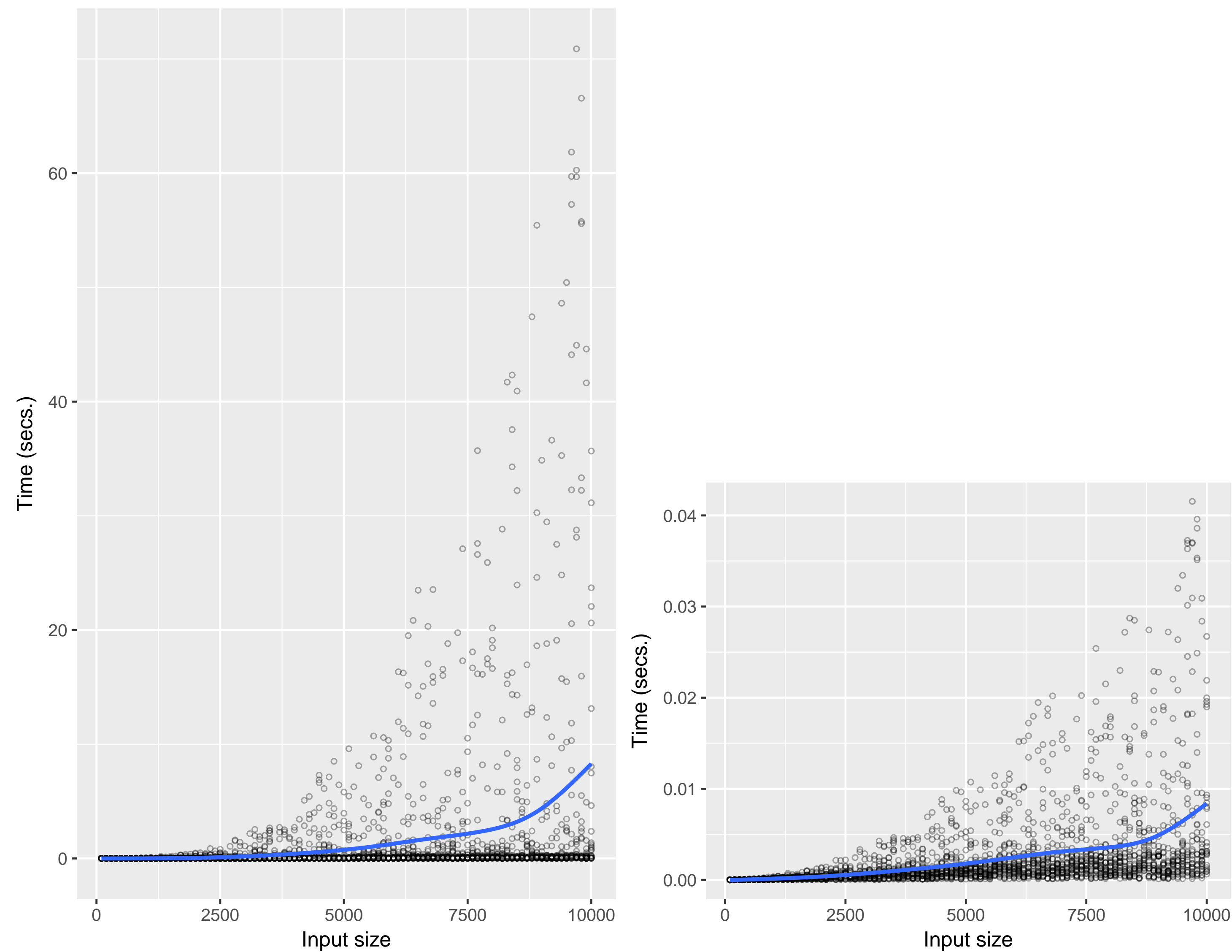

Figure 4.15: Tests with α-A-C-AC operators. The scale of the time-axis on the right-hand side is 750 times bigger than on the left-hand side

their inverses). The log-linear algorithm defined in [24] to check α-equivalence relies on the use of "lazy permutations": permutations, their inverses and supports are "suspended" over terms and updated eagerly whenever swappings have to be applied, but they are only pushed down one level in the tree structure of the terms when a transformation rule is applied, and they are applied to terms only when necessary.

Remark 4.4. *To illustrate why such an approach is used, consider lines 10 to 14 in Algorithm 1, related with the application of the rule* ($\approx_\alpha$ [**ab**]). *Special care has to be taken with* $(a\ b)\,t'$ *(line 12, rule* ($\approx_\alpha$ [**ab**])*), since it is not a term in our syntax, the permutation has to be propagated in* t' *and this implies an additional linear factor on the complexity of checking* α*-equivalence. However, adopting the above-mentioned approach, where the syntax is extended with "suspended" permutations over terms, which are propagated in a "lazy" way, this linear factor is avoided. Also, notice that there is a secondary freshness checking in* $\nabla \vdash a \# t'$*. This requires an algorithm for validating freshness constraints based on simplification rules for freshness (Fig. 2.5 bottom up) which is linear in* $\langle \nabla, a \# t' \rangle$*. To avoid repeated computations (for instance, the check for* $\nabla \vdash a \# t'$ *may appear several times in the computation) one could keep a list of atoms that need to be fresh to a term. This was exactly the propose of [24], where lists of atoms and permutations are appended to terms.*

Theorem 4.1 (Running time bounds)**.** *Let* n *be the size of a problem* $\langle \nabla, P \rangle$*, given as* $|\langle \nabla, P \rangle| := |\nabla| + |P|$*, where* $|\nabla|$ *is the number of atoms and variables occurring in* ∇ *and* $|P|$ *the sum of the size of terms in equations in* P*. The validity of* $\langle \nabla, P \rangle$ *modulo A, C and AC can be checked in time*

i) $O(n \log n)$*, if the problem includes neither C nor AC-function symbols;*

ii) $O(n^2 \log n)$*, if the problem does not contain AC function symbols; and*

iii) $O(n^3 \log n)$*, otherwise.*

Proof. (sketch)

To obtain these bounds, first the use of suspended permutations over terms and of lazy propagation of permutations and freshness checks (see Rmk. 4.4) is assumed; second, that terms in the problems are pre-computed providing a flat representation of the arguments of A and AC-function symbols. All maximal subterms should be linearly pre-computed to provide their arguments. This can be done for instance using sequences or arrays of terms in which arguments are *flattened* and might be accessed randomly (in constant time).

i) Consider a problem of the form $\langle \nabla, \{s \approx t\} \rangle$ where s and t contain neither C- nor AC-function symbols. Since A-function symbols are assumed to be flattened, the

problem can be log-linearly solved through a simple adaptation of the solution for α-equivalence checking given in [24]. For the A case, the problem can be directly decomposed, according to the number n_s of flattened arguments, into a new problem with n_s new disjoint equational sub-problems, that is, a problem of the form $\langle \nabla, P \cup \{f_k^A s' \approx f_k^A t'\}\rangle$ becomes directly a problem of the form $\langle \nabla, P \cup \{s'_{(1)_{f_k^A}} \approx t'_{(1)_{f_k^A}}, \ldots, s'_{(n_s)_{f_k^A}} \approx t'_{(n_s)_{f_k^A}}\}\rangle$.

ii) Let $\langle \nabla, \{s \approx t\}\rangle$ be a problem without AC-function symbols. A regular worst case happens when the problem has k nested C-function symbols. Assume that $n = m\, 2^k$, where $m \ll n$. In this case, considering terms with the same commutative symbol at the root, and with arguments of the same size, an upper bound for the running time is given by the recurrence relation:

$$T(n) = 4\,T(n/2) + O(n\ \log n)\ \textit{where}\ T(m) = O(m \log m).$$

In the first recurrence equation, the first summand has a factor 4 because it is necessary to check four sub problems of half of the original size, and the second summand provides a bound, according to the previous item, if the term does not have C-operators. Notice that both summands are included since the objective is to give an upper bound. The initial condition of the recurrence relation also assumes that sub-problems of size m have no occurrences of C-function symbols. Thus one has,

$$\begin{array}{rcl} T(n) & = & 4\,T(n/2) + O(n \log n) \\ & = & 4^k\, T(m) + O(n) \sum_{i=0}^{k-1} 2^i (\log n - i \log 2) \\ & = & (\frac{n}{m})^2 O(m \log m) + O(n \log n) \sum_{i=0}^{k-1} 2^i - O(n) \log 2 \sum_{i=0}^{k-1} 2^i i \\ & = & O(n^2 \frac{\log m}{m}) + O(n \log n)(2^k - 1) - O(n) \log 2 (2^k (k-2) + 2) \\ & = & O(n^2) + O(n^2 \log n) \\ & = & O(n^2 \log n) \end{array}$$

Factors related with m can be omitted since $m \ll n$ was assumed.

iii) First, notice that terms headed by C-function symbols can be considered as a particular case of AC symbols whose tuples (arguments) have always exactly two elements. Thus, the complexity analysis for C- and AC-function symbols could be unified.

Let $\langle \nabla, \{s \approx t\}\rangle$ be a problem that contains AC-function symbols. Assuming the flat representation of all maximal subterms of s and t that are headed with A and

AC-function symbols is pre-computed, the relevant part of the analysis is related with the verification of α-equivalence between subterms s' and t' of s and t headed by an AC-function symbol, say f_k^{AC}. This involves checking whether the tuple of n_s arguments in s' contains arguments that are related by α-equivalence modulo AC to arguments of the tuple of arguments in t'. These arguments are not necessarily in the same positions in the tuples of arguments of s' and t'. In the worst case scenario, for each argument of the tuple of arguments of f_k^{AC} in s', say $s'_{(i)_{f_k^{AC}}}$, one has to go over the whole tuple of arguments of f_k^{AC} in t', checking $\langle \nabla, \{s'_{(i)_{f_k^{AC}}} \approx t'_{(j)_{f_k^{AC}}}\}\rangle$, for $i, j \leq ||s'||_{f_k^{AC}}$. In case this is true, $\approx_{\{A,C,AC\}}$-equivalence eliminating these two arguments of the tuples should be checked. By item i), one already knows that the procedure without C and AC symbols is log-linear.

The problem essentially boils down to the problem of searching a perfect matching in the bipartite graph that consists of vertices V labelled by the n_s arguments of the *lhs*'s and *rhs*'s and edges, E, between vertices labelled with terms that match, as proved in [17] for solving AC-matching in the usual first-order syntax. This problem is known to have solutions of complexity $\mathrm{O}(|V| \times |E|)$, that is the same as $\mathrm{O}(|V|^3)$ since in the worst case one has $\mathrm{O}(|V|^2)$ edges (see for instance [33]).

One concludes that searching for a perfect matching is bounded cubically on the size of the problem, since the number of arguments, $||s'||_{f_k^{AC}}$, is linearly bounded in the size of the problem. But notice that for the case of just AC-equivalence, applying this method requires only complexity $\mathrm{O}(|V|^2)$ since after having checked two terms to be equivalent, the corresponding edge can be fixed and checking for other equivalences for these terms is unnecessary. Thus, an upper bound for the whole problem is $\mathrm{O}(n^3 \log n)$.

$\square$

Chapter 5

Nominal C-unification and matching

In this chapter the techniques to specify and formalise the main properties of nominal unification and matching in a signature with C function symbols are described. Results of the present chapter were published in [3] and in [7].

5.1 Formalisation of nominal C-unification with protected variables

This section describes a formalisation of an algorithm to solve nominal unification problems in a signature with commutative function symbols. This algorithm was specified using a rule-based approach that follows the same strategy presented in Subsec. 2.2.2. Two sets of transformation rules are used, one to deal with equations (Fig. 5.1) and the other to deal with freshness constraints (Fig. 5.2). These rules act over quadruples of the form $\langle \nabla, \mathcal{X}, \sigma, P \rangle$, where: ∇ is a freshness context; $\mathcal{X}$ is a set of protected variables (variables that are forbidden to be instantiated); σ is a substitution; and P is a set of nominal constraints. From now, calligraphic uppercase letters (e.g., $\mathcal{P}, \mathcal{Q}, \mathcal{R}$, etc) will denote quadruples.

The inference rules transform a quadruple into a finite family of *fixed point problems* $\{\mathcal{Q}_0, ..., \mathcal{Q}_n\}$. These fixed point (or just FP) problems are quadruples whose set of nominal constraints is composed only by FP equations of form $\pi.X \approx_? X$.

Example 5.1. *Given the nominal unification problem* $\mathcal{P} = \langle \emptyset, \{[a][b]X \approx_? [b][a]X\} \rangle$, *the standard unification algorithm given in Subsec. 2.2.2 reduces it to* $\langle \emptyset, \{X \approx_? (a\,b).X\} \rangle$, *which gives the solution* $\langle \{a\#X, b\#X\}, id \rangle$. *However, as will be showed in Chap. 6, independent solutions are feasible when there is at least one commutative function symbol in the signature. The derivation form* $\mathcal{P}$ *to* $\langle \{a\#X, b\#X\}, id \rangle$ *is given below.*

$$\mathcal{P} = \langle \emptyset, id, \{ [a][b]X \approx_? [b][a]X \} \rangle$$

$$\Rightarrow_{(\approx_?[\mathbf{ab}])} \langle \emptyset, id, \{ [b]X \approx_? [b](a\,b).X ,\; a \#_? [a]X \} \rangle$$

$$\Rightarrow_{(\approx_?[\mathbf{aa}])} \langle \emptyset, id, \{ X \approx_? (a\,b).X ,\; a \#_? [a]X \} \rangle$$

$$\Rightarrow_{(\approx_?\mathbf{var})} \langle \{a\#X, b\#X\}, id, \{ a \#_? [a]X \} \rangle$$

$$\Rightarrow_{(\#_?[\mathbf{aa}])} \langle \{a\#X, b\#X\}, id, \emptyset \rangle$$

5.1.1 Basic formalised notions and results on nominal C-unification

Definition 5.1 (Nominal C-unification problem with protected variables)**.** *A* nominal C-unification problem with protected variables *is a triple* $\langle \nabla, \mathcal{X}, P \rangle$*, where* ∇ *is a* freshness context*,* $\mathcal{X}$ *a set of protected variables, and* P *a finite set of nominal constraints.*

From now, $\nabla \vdash \sigma \approx \sigma'$ will be replaced by $\nabla \vdash \sigma \approx_{\alpha,C} \sigma'$ and denote that $\nabla \vdash X\sigma \approx_{\alpha,C} X\sigma'$ for all X in $dom(\sigma) \cup dom(\sigma')$.

Definition 5.2 (Solution for quadruples and unification problems)**.** *A* solution for a quadruple $\mathcal{P} = \langle \Delta, \mathcal{X}, \delta, P \rangle$ *is a pair* $\langle \nabla, \sigma \rangle$*, where the domain of* σ *has no variables in* $\mathcal{X}$*, and the following conditions are satisfied:*

i) $\nabla \vdash \Delta\sigma$*;*

ii) if $a \#_? t \in P$ *then* $\nabla \vdash a \# t\sigma$*;*

iii) if $s \approx_? t \in P$ *then* $\nabla \vdash s\sigma \approx_{\alpha,C} t\sigma$*;*

iv) there exists λ *such that* $\nabla \vdash \delta\lambda \approx_{\alpha,C} \sigma$*.*

A solution for a unification problem $\langle \Delta, \mathcal{X}, P \rangle$ *is a solution for the associated quadruple* $\langle \Delta, \mathcal{X}, id, P \rangle$*. The* solution set *for a unification problem or quadruple* $\mathcal{P}$ *is denoted by* $\mathcal{U}_C(\mathcal{P})$*.*

Definition 5.3 (Variables of a quadruple)**.** *The set of variables of* $\mathcal{P} = \langle \nabla, \mathcal{X}, \sigma, P \rangle$ *is defined as* $Var(P)$ *and also denoted by* $Var(\mathcal{P})$*.*

The only rule of Fig. 5.1 that can generate branches is $(\approx_? \mathbf{C})$. This rule is indeed an abbreviation for two rules providing the different forms in which one can relate the arguments s and t in an equation $f_k^C\, s \approx_? f_k^C\, t$ for a commutative function symbol: either $\langle s_0, s_1 \rangle \approx_? \langle t_0, t_1 \rangle$ or $\langle s_0, s_1 \rangle \approx_? \langle t_1, t_0 \rangle$.

Definition 5.4 (Proper problem)**.** *A quadruple* $\mathcal{P} = \langle \Delta, \mathcal{X}, \delta, P \rangle$ *is called a* proper problem *if every commutative function symbol in* P *has a tuple as argument.*

$$\frac{\langle \nabla, \mathcal{X}, \sigma, P \uplus \{s \approx_? s\}\rangle}{\langle \nabla, \mathcal{X}, \sigma, P\rangle} (\approx_? \mathbf{refl}) \qquad \frac{\langle \nabla, \mathcal{X}, \sigma, P \uplus \{\langle s_1, t_1\rangle \approx_? \langle s_2, t_2\rangle\}\rangle}{\langle \nabla, \mathcal{X}, \sigma, P \cup \{s_1 \approx_? s_2, t_1 \approx_? t_2\}\rangle} (\approx_? \mathbf{pair})$$

$$\frac{\langle \nabla, \mathcal{X}, \sigma, P \uplus \{\pi.X \approx_? \pi'.X\}\rangle,\ \pi' \neq id}{\langle \nabla, \mathcal{X}, \sigma, P \cup \{\pi \oplus (\pi')^{-1}.X \approx_? X\}\rangle} (\approx_? \mathbf{inv}) \qquad \frac{\langle \nabla, \mathcal{X}, \sigma, P \uplus \{f_k^E\, s \approx_? f_k^E\, t\}\rangle,\ E \neq C}{\langle \nabla, \mathcal{X}, \sigma, P \cup \{s \approx_? t\}\rangle} (\approx_? \mathbf{app})$$

$$\frac{\langle \nabla, \mathcal{X}, \sigma, P \uplus \{f_k^C\, \langle s_0, s_1\rangle \approx_? f_k^C\, \langle t_0, t_1\rangle\}\rangle,\ i = 0, 1}{\langle \nabla, \mathcal{X}, \sigma, P \cup \{s_0 \approx_? t_i, s_1 \approx_? t_{1-i}\rangle} (\approx_? \mathbf{C})$$

$$\frac{\langle \nabla, \mathcal{X}, \sigma, P \uplus \{[a]s \approx_? [a]t\}\rangle}{\langle \nabla, \mathcal{X}, \sigma, P \cup \{s \approx_? t\}\rangle} (\approx_? [\mathbf{aa}]) \qquad \frac{\langle \nabla, \mathcal{X}, \sigma, P \uplus \{[a]s \approx_? [b]t\}\rangle}{\langle \nabla, \mathcal{X}, \sigma, P \cup \{s \approx_? (a\, b) \cdot t, a \#_? t\}\rangle} (\approx_? [\mathbf{ab}])$$

$$\frac{\langle \nabla, \mathcal{X}, \sigma, P \uplus \{\pi.X \approx_? t\} \text{ or } \{t \approx_? \pi.X\}\rangle \ \ \text{let } \sigma' := \sigma\{X/\pi^{-1} \cdot t\},\ X \notin \mathtt{var}(t) \cup \mathcal{X}}{\left\langle \nabla, \mathcal{X}, \sigma', P\{X/\pi^{-1} \cdot t\} \cup \bigcup_{\substack{Y \in dom(\sigma'),\\ a\#Y \in \nabla}} \{a \#_? Y\sigma'\} \right\rangle} (\approx_? \mathbf{inst})$$

Figure 5.1: Reduction rules for equations

Remark 5.1. *Terms in the equations preserve the syntactic restriction that commutative symbols are only applied to tuples. This restriction is not crucial since any equation of the form* $f_k^C \pi.X \approx_? t$ *can be translated into an equation of form* $f_k^C \langle \pi.X_1, \pi.X_2\rangle \approx_? t$*, where* X_1 *and* X_2 *are new variables and* ∇ *is extended to* ∇' *in such a way that both* X_1 *and* X_2 *inherit all freshness constraints of* X *in* ∇*:* $\nabla' = \nabla \cup \{a\#X_i \mid i = 1, 2, \text{ and } a\#X \in \nabla\}$*.*

In the rule ($\approx_?$ **inst**) of Fig. 5.1 notice that the inclusion of new constraints in the problem, given in $\bigcup_{(Y \in dom(\sigma'),\ a\#Y \in \nabla)}\{a \#_? Y\sigma'\}$ is necessary to guarantee that the new substitution σ' is *compatible* with the freshness context ∇. Also, checking whether the variable X is a protected variable is a condition for the application of rule ($\approx_?$ **inst**).

Notation 5.1. $\mathcal{P} \Rightarrow_\approx \mathcal{Q}$ *(resp.* $\mathcal{P} \Rightarrow_\# \mathcal{Q}$*) will denote that* $\mathcal{P}$ *reduces to* $\mathcal{Q}$ *by the rules of Fig. 5.1 (resp. Fig. 5.2). As usual, the relation* $\Rightarrow_\approx^*$ *(resp.* $\Rightarrow_\#^*$*) denotes zero or more reduction steps of* $\Rightarrow_\approx$ *(resp.* $\Rightarrow_\#$*).*

Notation 5.2 (Set of $\Rightarrow_\approx$ and $\Rightarrow_\#$-normal forms)**.** $\mathcal{P}_{\Rightarrow_\approx}$ *(resp.* $\mathcal{P}_{\Rightarrow_\#}$*) denotes the set of normal forms of* $\mathcal{P}$ *with respect to* $\Rightarrow_\approx$ *(resp.* $\Rightarrow_\#$*).*

The relations $\Rightarrow_\#$ and $\Rightarrow_\approx$ are specified in file `C_Unif.v` through propositional relations fresh_sys and equ_sys in Figs. 5.4 and 5.5. Each constructor of these definitions represents

$$\frac{\langle \nabla, \mathcal{X}, \sigma, P \uplus \{a \#_? \langle\rangle\}\rangle}{\langle \nabla, \mathcal{X}, \sigma, P\rangle} (\#_? \langle\rangle) \qquad \frac{\langle \nabla, \mathcal{X}, \sigma, P \uplus \{a \#_? \bar{b}\}\rangle}{\langle \nabla, \mathcal{X}, \sigma, P\rangle} (\#_? \mathbf{a\bar{b}})$$

$$\frac{\langle\langle \nabla, \mathcal{X}, \sigma, P \uplus \{a \#_? f\, t\}\rangle}{\langle \nabla, \mathcal{X}, \sigma, P \cup \{a \#_? t\}\rangle} (\#_? \mathbf{app}) \qquad \frac{\langle \nabla, \mathcal{X}, \sigma, P \uplus \{a \#_? [a]t\}\rangle}{\langle \nabla, \mathcal{X}, \sigma, P\rangle} (\#_? \mathbf{a}[\mathbf{a}])$$

$$\frac{\langle \nabla, \mathcal{X}, \sigma, P \uplus \{a \#_? [b]t\}\rangle}{\langle \nabla, \mathcal{X}, \sigma, P \cup \{a \#_? t\}\rangle} (\#_? \mathbf{a}[\mathbf{b}]) \qquad \frac{\langle \nabla, \mathcal{X}, \sigma, P \uplus \{a \#_? \pi.X\}\rangle}{\langle \{(\pi^{-1} \cdot a)\# X\} \cup \nabla, \mathcal{X}, \sigma, P\rangle} (\#_? \mathbf{var})$$

$$\frac{\langle \nabla, \mathcal{X}, \sigma, P \uplus \{a \#_? \langle s, t\rangle\}\rangle}{\langle \nabla, \mathcal{X}, \sigma, P \cup \{a \#_? s, a \#_? t\}\rangle} (\#_? \mathbf{pair})$$

Figure 5.2: Reduction rules for freshness constraints

one rule, except for the case of rule ($\approx_?$ **C**) that is specified by two constructors: equ_sys_C1 and equ_sys_C2. For instance, rules ($\approx_?$ [**aa**]) and ($\approx_?$ [**ab**]) are specified, respectively, by equ_sys_Ab1 and equ_sys_Ab1, and rules ($\#_?$ **a**[**a**]) and ($\#_?$ **a**[**b**]), respectively, by fresh_sys_Ab_1 and fresh_sys_Ab_2.

Remark 5.2. *Notice that, in* equ_sys *(Fig. 5.4) the set of protected variables $\mathcal{X}$ is given by the parameter varSet in the definition of the relation. Then simplification rules are applied over triples $\langle \nabla, \sigma, P\rangle$, instead of quadruples as presented in the theory. This difference has no effect in the behaviour of the algorithm or in the formal proofs. The set $\mathcal{X}$ (or varSet) remains unchanged during derivations, having action only in the application of rule (*$\approx_?$ **inst**)*. The condition of this rule of checking whether $X \in Var(t) \cup \mathcal{X}$ is specified in the expression* (¬ set_In X (set_union var_eqdec (term_vars t) *varSet*)) *of constructor* equ_sys_inst *(of relation* equ_sys*).*

Normal forms (*NF*) and the transitive-reflexive closure of a generic propositional relation (**tr_clos**) are specified also in file `C_Unif.v` as presented in Fig. 5.3.

```
Definition NF (T:Type) (R:T→T→Prop) (s:T) := ∀ t, ¬ R s t.

Inductive tr_clos (T:Type) (R:T→T→Prop) : T→T→Prop :=
| tr_rf : ∀ s, tr_clos T R s s
| tr_os : ∀ s t, R s t → tr_clos T R s t
| tr_ms : ∀ s t u, R s t → tr_clos T R t u → tr_clos T R s u
```

Figure 5.3: Specification of NF and tr_clos

```
Inductive equ_sys (varSet : set Var) : Triple → Triple → Prop :=

 | equ_sys_refl : ∀ C S P s, (set_In (s ∼? s) P) →
   equ_sys varSet (C,S,P) (C,S,P\(s ∼? s))

 | equ_sys_Pr : ∀ C S P s0 s1 t0 t1,
   (set_In ((<|s0,s1|> ∼? <|t0,t1|>)) P) →
   equ_sys varSet (C,S,P) (C,S,(((P|+(s0 ∼? t0))|+(s1 ∼? t1))\((<|s0,s1|> ∼? <|t0,t1|>))))

 | equ_sys_Fc : ∀ C S P E n t t', (set_In ((Fc E n t ∼? Fc E n t')) P) → E ≠ 2 →
   equ_sys varSet (C,S,P) (C,S,(P|+(t ∼? t'))\((Fc E n t ∼? Fc E n t')))

 | equ_sys_C1 : ∀ C S P n s0 s1 t0 t1, (set_In ((Fc 2 n (<|s0,s1|>) ∼? Fc 2 n (<|t0,t1|>)))) P →
   equ_sys varSet (C,S,P)
   (C,S,((P|+(s0 ∼? t0))|+(s1 ∼? t1))\(Fc 2 n (<|s0,s1|>) ∼? Fc 2 n (<|t0,t1|>))))

 | equ_sys_C2 : ∀ C S P n s0 s1 t0 t1, (set_In ((Fc 2 n (<|s0,s1|>) ∼? Fc 2 n (<|t0,t1|>)))) P →
   equ_sys varSet (C,S,P)
   (C,S,((P|+(s0 ∼? t1))|+(s1 ∼? t0))\(Fc 2 n (<|s0,s1|>) ∼? Fc 2 n (<|t0,t1|>))))

 | equ_sys_Ab1 : ∀ C S P a t t', (set_In (([a]^t ∼? [a]^t')) P) →
   equ_sys varSet (C,S,P) (C,S,(P|+(t ∼? t'))\(([a]^t ∼? [a]^t')))

 | equ_sys_Ab2 : ∀ C S P a b t t', a ≠ b → (set_In (([a]^t ∼? [b]^t')) P) →
   equ_sys varSet (C,S,P)
   (C,S,((P|+(t ∼? ((a,b)::nil)@t'))|+(a #? t')))\(([a]^t ∼? [b]^t')))

 | equ_sys_inst : ∀ C S S' P pi X t,
   (¬ set_In X (set_union var_eqdec (term_vars t) varSet)) →
   ((set_In (pi|.X ∼? t) P) ∨ (set_In (t ∼? pi|.X)) P)) →
   S' = S©((X,(!pi)@t)::nil) →
   equ_sys varSet (C,S,P)
   (C,S',((P\(pi|.X ∼? t)\(t ∼? pi|.X)))|^^((X,(!pi)@t)::nil))\cup(C/?S'))

 | equ_sys_inv : ∀ C S P pi pi' X, pi ≠ pi' → pi' ≠ [] → (set_In ((pi|.X ∼? pi'|.X)) P) →
   equ_sys varSet (C,S,P)
   (C,S,(P|+((pi++(!pi'))|.X ∼? []|.X)))\((pi|.X ∼? pi'|.X))) .
```

Figure 5.4: Specification of equ_sys

```
Inductive fresh_sys : Triple → Triple → Prop :=
 | fresh_sys_Ut : ∀ C S P a, (set_In (a #? («»)) P) →
   fresh_sys (C,S,P) (C,S,P\(a #? («»)))
 | fresh_sys_At : ∀ C S P a b, a ≠ b → (set_In (a #? (%b)) P) →
   fresh_sys (C,S,P) (C,S,(P\(a #? (%b))))
 | fresh_sys_Fc : ∀ C S P a E n s, (set_In (a #? (Fc E n s)) P) →
   fresh_sys (C,S,P) (C,S,(P|+(a #? s))\(a #? (Fc E n s)))
 | fresh_sys_Ab_1 : ∀ C S P a s, (set_In (a #? ([a]^s)) P) →
   fresh_sys (C,S,P) (C,S,(P\(a #? ([a]^s))))
 | fresh_sys_Ab_2 : ∀ C S P a b s, a ≠ b → (set_In (a #? ([b]^s)) P) →
   fresh_sys (C,S,P) (C,S,(P|+(a #? s))\(a #? ([b]^s)))
 | fresh_sys_Su : ∀ C S P a pi X, (set_In (a #? (pi|.X)) P) →
   fresh_sys (C,S,P) (C|++((!pi) $ a,X),S,(P\(a #? (pi|.X))))
 | fresh_sys_Pr : ∀ C S P a s t, (set_In (a #? (<|s,t|>)) P) →
   fresh_sys (C,S,P) (C,S,(((P|+(a #? s))|+(a #? t))\(a #? (<|s,t|>)))) .
```

Figure 5.5: Specification of fresh_sys

Definition 5.5 (Valid quadruple)**.** $\mathcal{P} = \langle \nabla, \mathcal{X}, \sigma, P\rangle$ is valid *if* $im(\sigma) \cap dom(\sigma) = \emptyset$ *and* $dom(\sigma) \cap Var(P) = \emptyset$*.*

Remark 5.3. *A substitution σ in a valid quadruple $\mathcal{P}$ is* idempotent*, that is, $\sigma\sigma = \sigma$.*

Remark 5.4. *The relation $\Rightarrow_{\approx}$, starts from a quadruple with the identity substitution and always maintains a quadruple $\langle \nabla, \mathcal{X}, \sigma', P'\rangle$ in which the substitution σ' does not affect the current problem P'. The same happens for $\Rightarrow_{\#}$ since the substitution does not change with this relation. This motivates the next definition and lemma.*

Lemma 5.1 (Preservation of valid quadruples by $\Rightarrow_{\approx}$ or $\Rightarrow_{\#}$)**.** *If $\mathcal{P} = \langle \nabla, \mathcal{X}, \sigma, P\rangle$ is valid and $\mathcal{P} \Rightarrow_{\approx} \cup \Rightarrow_{\#} \mathcal{P}' = \langle \nabla', \mathcal{X}, \sigma', P'\rangle$, then $\mathcal{P}'$ is also valid.*

Proof. The proof is by case analysis on the derivation rules used in the relation $\Rightarrow_{\approx} \cup \Rightarrow_{\#}$, and its formalisation is in file `C_Unif.v`, Lems. `fresh_valid_preservation` and `equ_valid_preservation`. The only rule that enlarges the domain of σ is ($\approx_?$ **inst**) building a new substitution $\sigma' = \sigma\{X/\pi^{-1} \cdot t\}$, for t such that $X \notin Var(t)$. Since $dom(\sigma) \cap Var(P) = \emptyset$ and t occurs in P, $dom(\sigma') \cap Var(t) = \emptyset$; the first property is obtained: $im(\sigma') \cap dom(\sigma') = \emptyset$. For the second property, notice that P' consists of two parts:

1. $(P - \{\pi.X \approx_? t\})\{X/\pi^{-1} \cdot t\}$ that includes equations and freshness constraints whose variables do not intersect $dom(\sigma')$; and

2. $\bigcup_{(Y \in dom(\sigma'),\ a\#Y \in \nabla)} \{a \#_? Y\sigma'\}$ that also does not include variables in $dom(\sigma')$, since $im(\sigma') \cap dom(\sigma') = \emptyset$.

Consequently, $dom(\sigma') \cap Var(P') = \emptyset$ and then $\mathcal{P}'$ is a valid quadruple. □

From now on, only valid quadruples will be considered.

Notation 5.3. *Extending the Not. 2.7, $P_\approx$, $P_\#$, $P_{fp_\approx}$ and $P_{nfp_\approx}$ will, respectively, denote the sets of equations, freshness constraints, FP and non FP equations in the set P.*

5.1.2 Main formalised results for C-unification

Lemma 5.2 (Termination of $\Rightarrow_\approx$ and $\Rightarrow_\#$)**.** *The relations $\Rightarrow_\approx$ and $\Rightarrow_\#$ are terminating.*

Proof. The proof is by well-founded induction on $\mathcal{P}$ using the measure

$$|\mathcal{P}| = \langle |Var(P_\approx)|, |P_\approx|, |P_{\mathrm{nfp}_\approx}|, |P_\#| \rangle$$

with a lexicographic ordering. Its formalisation is in file `C_Unif_Termination`, Lems. `equ_sys_termination` and `fresh_sys_termination`.

i) Note that this measure decreases after each step $\langle \nabla, \mathcal{X}, \sigma, P \rangle \Rightarrow_\approx \langle \nabla, \mathcal{X}, \sigma', P' \rangle$:

   - for ($\approx_?$ **inst**), $|Var(P_\approx)| > |Var(P'_\approx)|$;
   - for ($\approx_?$ **refl**), ($\approx_?$ **pair**), ($\approx_?$ **app**), ($\approx_?$ [**aa**]), ($\approx_?$ [**ab**]) and ($\approx_?$ **C**), $|Var(P_\approx)| \geq |Var(P'_\approx)|$, but $|P_\approx| > |P'_\approx|$;
   - and, for ($\approx_?$ **inv**), both $|Var(P_\approx)| = |Var(P' \approx)|$ and $|P_\approx| = |P'_\approx|$, but $|P_{\mathrm{nfp}_\approx}| > |P'_{\mathrm{nfp}_\approx}|$.

ii) For $\langle \nabla, \mathcal{X}, \sigma, P \rangle \Rightarrow_\# \langle \nabla, \mathcal{X}, \sigma', P' \rangle$, $|Var(P_\approx)| \geq |Var(P'_\approx)|$, $|P_\approx| = |P'_\approx|$, $|P_{\mathrm{nfp}_\approx}| = |P'_{\mathrm{nfp}_\approx}|$, but $|P_\#| > |P'_\#|$.

□

Next Def. 5.6 gives the strategy of application of $\Rightarrow_\approx$ and $\Rightarrow_\#$. Since both systems are terminating (Lem. 5.2), this strategy results also in a terminating process.

Definition 5.6 (Derivation tree for $\langle \Delta, \mathcal{X}, P \rangle$)**.** *A* derivation tree *for the unification problem $\langle \Delta, \mathcal{X}, P \rangle$, denoted as $\mathcal{T}_{\langle \Delta, \mathcal{X}, P \rangle}$, is a tree with root label $\mathcal{P} = \langle \Delta, \mathcal{X}, id, P \rangle$ built in two stages:*

i) Initially, a tree is built, whose branches end in leaf nodes labelled with the quadruples in $\mathcal{P}_{\Rightarrow_\approx}$. The labels in each path from the root to a leaf correspond to a $\Rightarrow_\approx$-derivation.

ii) Further, for each leaf labelled with a quadruple $\mathcal{Q}$ in $\mathcal{P}_{\Rightarrow_\approx}$, such that $Q_\approx = Q_{fp_\approx}$, the tree is extended with a path to a new leaf that is labelled with a $\bar{\mathcal{Q}} \in \mathcal{Q}_{\Rightarrow_\#}$. The labels in the extended path from the node with label $\mathcal{Q}$ to the new leaf correspond to a $\Rightarrow_\#$-derivation.

For $\langle \Delta, \mathcal{X}, P\rangle$ a unification problem, all labels in the nodes of $\mathcal{T}_{\langle \Delta,\mathcal{X},P\rangle}$ are *valid* by Lem. 5.1.

An OCaml implementation of the nominal C-unification algorithm with protected variables is available at `Impl-Unif/`, inside the specification folder. This implementation is based on a straightforward algorithmic strategy in which simplification paths are built choosing equations and freshness constraints in the order in which they appear in the unification problem (using a list data structure). The rules in $\Rightarrow_\approx$ and $\Rightarrow_\#$ are applied in this order, according to the construction of derivation trees in Def. 5.6. As soon as an equation or a freshness constraint can not be simplified the algorithm fails. The implementation outputs the derivation tree in `TikZ` latex code. Examples of derivation trees generated with the implementation are given in Figs. 5.6, 5.7, 5.11, 5.15 to 5.18 and 6.1. The symbols $(\perp)$ and $(\top)$ are added to the trees to discriminate, respectively, failure and successful leaves. The draw of trees in the examples were manually adjusted to fit the page.

Example 5.2 (Continuing Ex. 5.1)**.**
$\mathcal{P} = \langle \emptyset, \emptyset, id, \{[a][b]X \approx_? [b][a]X\}\rangle$ reduces to $\langle \emptyset, \emptyset, id, \{(a\,b).X \approx_? X\}\rangle$ via rules in Figs. 5.1 and 5.2 (see derivation tree in Fig. 5.6).

In the following, infix notation is adopted for commutative symbols: $s * t$ abbreviates $*\langle s, t\rangle$.

Example 5.3. *Let $*$ be a commutative function symbol. Below, it is showed how the problem*

$$\mathcal{P} = \langle \emptyset, \emptyset, id, \{[a'](a\,b).X * Y \approx_? [b'](a\,c)(c\,d).X * Y\}\rangle$$

reduces via rules in Figs. 5.1 and 5.2 (see derivation tree in Fig. 5.7). Application of rule $(\approx_? \mathbf{C})$ gives two branches that reduce into two FP problems: $\mathcal{Q}_1$ and $\mathcal{Q}_2$. Observe that, in the first step of the derivation, the action of swapping $(a'\,b')$ (cf. Def. 2.20) over suspension $(a\,c)(c\,d).X$ has as result $(a\,c)(c\,d)(a'\,b').X$.

$$\mathcal{P} = \langle \emptyset, \emptyset, id, \{ [a'](a\,b).X * Y \approx_? [b'](a\,c)(c\,d).X * Y \}\rangle$$

$$\Rightarrow_{(\approx_?[\mathbf{ab}])} \langle \emptyset, \emptyset, id, \{ (a\,b).X * Y \approx_? (a\,c)(c\,d)(a'\,b').X * (a'\,b').Y , a' \#_? (a\,c)(c\,d).X * Y\}\rangle$$

1.

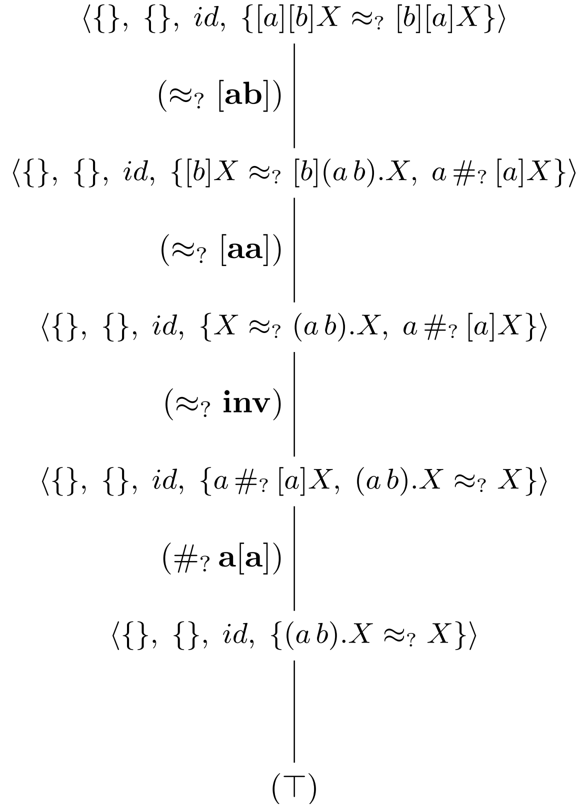

Figure 5.6: Derivation tree of Ex. 5.2.

$$\Rightarrow_{(\approx_?\mathbf{C})} \langle \emptyset, \emptyset, id, \{ (a\,b).X \approx_? (a\,c)(c\,d)(a'\,b').X,\ Y \approx_? (a'\,b').Y,\ a' \#_? (a\,c)(c\,d).X * Y\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{inv})(2\times)} \langle \emptyset, \emptyset, id, \{(a\,b)[(a\,c)(c\,d)(a'\,b')]^{-1}.X \approx_? X,\ [(a'\,b')]^{-1}.Y \approx_? Y,\ a' \#_? (a\,c)(c\,d).X * Y\}\rangle$$

$$\Rightarrow_{\substack{(\#_?\mathbf{app}),\\ (\#_?\mathbf{pair})}} \langle \emptyset, \emptyset, id, \{(a\,b)(a'\,b')(c\,d)(a\,c).X \approx_? X, (a'\,b').Y \approx_? Y,\ a' \#_? (a\,c)c\ d).X,\ a' \#_? Y\}\rangle$$

$$\Rightarrow_{(\#_?\mathbf{var})(2\times)} \langle \{a'\#X, a'\#Y\}, \emptyset, id, \{(a\,b)(a'\,b')(c\,d)(a\,c).X \approx_? X, (a'\,b').Y \approx_? Y\}\rangle = \mathcal{Q}_1$$

2.

$$\Rightarrow_{(\approx_?\mathbf{C})} \langle \emptyset, \emptyset, id, \{ (a\,b).X \approx_? (a'\,b').Y,\ Y \approx_? (a\,c)(c\,d)(a'\,b').X,\ a' \#_? (a\,c)(c\,d).X * Y\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{inst})} \langle \emptyset, \emptyset, X/(a'\,b')(a\,b).Y, \{ Y \approx_? (a'\,b')(a\,b)(a\,c)(c\,d)(a'\,b').Y,\ a' \#_? (a'\,b')(a\,b)(a\,c)(c\,d)(a'\,b').Y * Y\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{inv})} \langle \emptyset, \emptyset, X/(a'\,b')(a\,b).Y, \{(a'\,b')(a\,b)(a\,c)(c\,d)(a'\,b').Y,\ a' \#_? (a'\,b')(a\,b)(a\,c)(c\,d)(a'\,b').Y * Y\}\rangle$$

$$\Rightarrow_{\substack{(\#_?\mathbf{app}),\\ (\#_?\mathbf{pair})}} \langle \emptyset, \emptyset, X/(a'\,b')(a\,b).Y, \left\{ \begin{array}{l} (a'\,b')(a\,b)(a\,c)(c\,d)(a'\,b').Y \approx_? Y, \\ a' \#_? (a'\,b')(a\,b)(a\,c)(c\,d)(a'\,b').Y, \\ a' \#_? Y \end{array} \right\}\rangle$$

$$\Rightarrow_{(\#_?\mathbf{var})(2\times)} \langle \{a'\#Y, b'\#Y\}, \emptyset, X/(a'\,b')(a\,b).Y, \{(a'\,b')(a\,b)(a\,c)(c\,d)(a'\,b').Y \approx_? Y\}\rangle = \mathcal{Q}_2$$

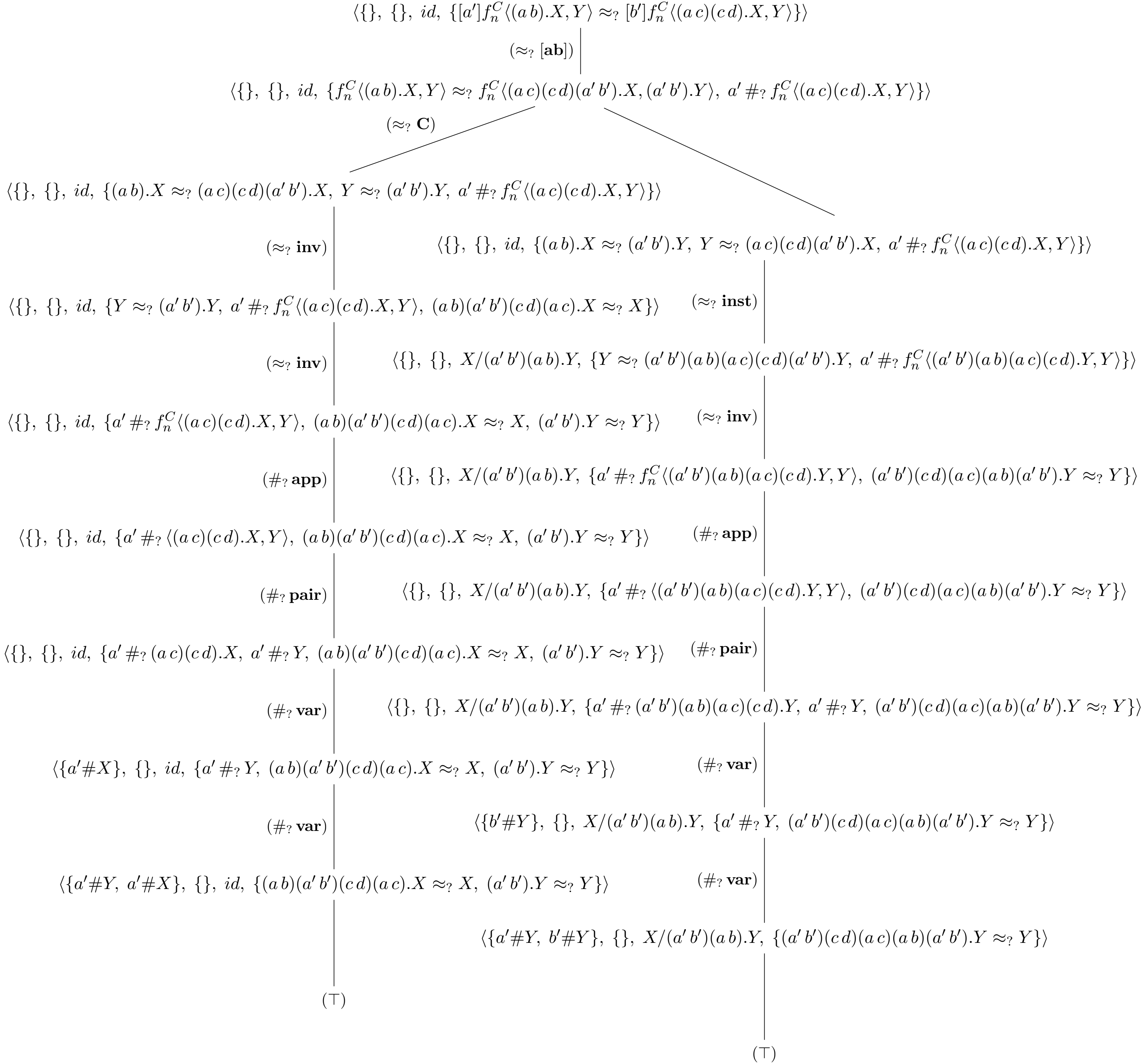

Figure 5.7: Derivation tree of Ex. 5.3.

Lemma 5.3 (Preservation of solutions by $\Rightarrow_{\#}$). *If $\mathcal{P} \Rightarrow_{\#} \mathcal{P}'$ then $\mathcal{U}_C(\mathcal{P}) = \mathcal{U}_C(\mathcal{P}')$.*

Proof. The proof is by case analysis on one-step $\Rightarrow_{\#}$-reduction and is formalised in file `C_Unif_Soundness`, Lem. `fresh_sys_compl`. The interesting case is rule $(\#_? \, \mathbf{var})$. For rules $(\#_? \, \langle\rangle)$, $(\#_? \, \mathbf{a\bar{b}})$, $(\#_? \, \mathbf{app})$, $(\#_? \, \mathbf{a[a]})$ $(\#_? \, \mathbf{a[b]})$ and $(\# \, \mathbf{pair})$ the analysis is simple and similar.

- For example, the analysis of case of rule $(\#_? \, \mathbf{a[b]})$ is given by:

$$\mathcal{P} = \langle \nabla, \mathcal{X}, \sigma, P \uplus \{a \#_? [b]t\} \rangle \Rightarrow_{\#} \langle \nabla, \mathcal{X}, \sigma, P \cup \{a \#_? t\} \rangle = \mathcal{P}' \rangle$$

  Supposing $\langle \nabla', \sigma' \rangle \in \mathcal{U}_C(\mathcal{P})$, except for the second condition in Def. 5.2, all other three conditions hold trivially for $\mathcal{P}'$. The same happens when $\langle \nabla', \sigma' \rangle \in \mathcal{U}_C(\mathcal{P}')$: conditions first, third and fourth in Def. 5.2 hold for $\mathcal{P}$. For the second condition in Def. 5.2, by application of rule $(\# \, \mathbf{a[b]})$ of the freshness relation (Fig. 2.5) and inversion property of these inference rule and, substitution action (Def. 2.24), one has that $\nabla' \vdash a \# t\sigma'$ if and only if $\nabla' \vdash a \# [b]t\sigma'$ if and only if $\nabla' \vdash a \# ([b]t\sigma')$. Hence, $\mathcal{U}_C(\mathcal{P}) = \mathcal{U}_C(\mathcal{P}')$.

- Now, consider the interesting case of $(\#_? \, \mathbf{var})$:

$$\mathcal{P} = \langle \nabla, \mathcal{X}, \sigma, P \uplus \{a \#_? \pi.X\} \rangle \Rightarrow_{\#} \langle \{(\pi^{-1} \cdot a) \# X\} \cup \nabla, \mathcal{X}, \sigma, P \rangle = \mathcal{P}'$$

  On the one hand, if $\langle \nabla', \sigma' \rangle \in \mathcal{U}_C(\mathcal{P})$, the second, third and fourth conditions in Def. 5.2 hold trivially for $\mathcal{P}'$. To prove the first condition for $\mathcal{P}'$, by the second condition in Def. 5.2 for $\mathcal{P}$ one has that $\nabla' \vdash a \# \pi.X\sigma'$, which, by nominal properties and substitution action (Def. 2.24), implies that $\nabla' \vdash \pi^{-1} \cdot a \# X\sigma'$. Since by hypothesis $\nabla' \vdash \nabla\sigma'$ (first condition of Def. 5.2 for $\mathcal{P}$), one has that $\nabla' \vdash (\{(\pi^{-1} \cdot a) \# X\} \cup \nabla)\sigma'$. Therefore, $\mathcal{U}_C(\mathcal{P}) \subseteq \mathcal{U}_C(\mathcal{P}')$.

  On the other hand, if $\langle \nabla', \sigma' \rangle \in \mathcal{U}_C(\mathcal{P}')$, the first, third and fourth conditions in Def. 5.2 hold trivially for $\mathcal{P}$. For proving the second condition for $\mathcal{P}$, by the first condition in Def. 5.2 one has that $\nabla' \vdash (\pi^{-1} \cdot a) \# X\sigma'$, which again by nominal properties and Def. 2.24 implies that $\nabla' \vdash a \# (\pi.X)\sigma'$. Thus, by the last and the second condition in Def. 5.2 for $\mathcal{P}'$, one obtains the second condition for $\mathcal{P}$. Therefore, $\mathcal{U}_C(\mathcal{P}') \subseteq \mathcal{U}_C(\mathcal{P})$.

□

Lemma 5.4 (Preservation of solutions by $\Rightarrow_{\approx}$). *If $\mathcal{P} \Rightarrow_{\approx} \mathcal{P}'$ then $\mathcal{U}_C(\mathcal{P}') \subseteq \mathcal{U}_C(\mathcal{P})$.*

Proof. The proof is by case analysis on one-step $\Rightarrow_{\approx}$-reduction and it is formalised in file `C_Unif_Soundness.v`, Lem. `equ_sol_preserv`.

- Rule ($\approx_?$ [**aa**]):

  $$\mathcal{P} = \langle \nabla, \mathcal{X}, \sigma, P \uplus \{[a]s \approx_? [a]t\}\rangle \Rightarrow_{\approx} \langle \nabla, \mathcal{X}, \sigma, P \cup \{s \approx_? t\}\rangle = \mathcal{P}'$$

  Let $\langle \nabla', \sigma' \rangle$ be a solution in $\mathcal{U}_C(\mathcal{P}')$. Then, according to the definition of solution (Def. 5.2) four conditions are satisfied: first, for all $a\#X \in \nabla$, $\nabla' \vdash a \# X\sigma'$; second, for all $a \#_? w \in P$, $\nabla' \vdash a \# w\sigma'$; third, for all $u \approx_? v \in P \cup \{s \approx_? t\}$, $\nabla' \vdash u\sigma' \approx_{\alpha,C} v\sigma'$, and; fourth, there exists λ such that $\nabla' \vdash \sigma\lambda \approx \sigma'$.

  Except for the third condition, all other conditions hold trivially. The third condition also holds, since (by the inference rules of $\approx_{\alpha,C}$ and Lem. 4.2) $\nabla' \vdash s\sigma' \approx_{\alpha,C} t\sigma'$ if only if $\nabla' \vdash [a]s\sigma' \approx_{\alpha,C} [a]t\sigma'$; hence, $\mathcal{U}_C(\mathcal{P}') \subseteq \mathcal{U}_C(\mathcal{P})$. Notice that a solution $\langle \nabla', \sigma' \rangle$ in $\mathcal{U}_C(\mathcal{P})$, satisfies the four conditions for $\mathcal{P}'$, hence one has that $\mathcal{U}_C(\mathcal{P}') = \mathcal{U}_C(\mathcal{P})$ indeed.

- A similar analysis of case of rule ($\approx_?$ [**aa**]) shows that $\mathcal{U}_C(\mathcal{P}) = \mathcal{U}_C(\mathcal{P}')$ also for rules ($\approx_?$ **refl**), ($\approx_?$ **pair**), ($\approx_?$ **app**).

- Rule ($\approx_?$ **C**): Consider a derivation of the form below, where $i = 0$ or $i = 1$:

  $$\mathcal{P} = \langle \nabla, \mathcal{X}, \sigma, P \uplus \{f_k^C\langle s_0, s_1\rangle \approx_? f_k^C\langle t_0, t_1\rangle\}\rangle \Rightarrow_{\approx} \langle \nabla, \mathcal{X}, \sigma, P \cup \{s_0 \approx_? t_i, s_1 \approx_? t_{1-i}\}\rangle = \mathcal{P}'$$

  As for the previous rules, for $\langle \nabla', \sigma' \rangle \in \mathcal{U}_C(\mathcal{P}')$, the first, second and fourth conditions in Def. 5.2 are preserved trivially. Regarding the third condition, it also holds since $\nabla' \vdash s_0 \approx_{\alpha,C} t_i$, and $\nabla' \vdash s_1 \approx_{\alpha,C} t_{1-i}$ implies that $\nabla' \vdash f_k^C\langle s_0, s_1\rangle\sigma' \approx_{\alpha,C} f_k^C\langle t_0, t_1\rangle\sigma'$. Thus, $\mathcal{U}_C(\mathcal{P}') \subseteq \mathcal{U}_C(\mathcal{P})$.

- Rule ($\approx_?$ [**ab**]):

  $$\mathcal{P} = \langle \nabla, \mathcal{X}, \sigma, P \uplus \{[a]s \approx_? [b]t\}\rangle \Rightarrow_{\approx} \langle \nabla, \mathcal{X}, \sigma, P \cup \{s \approx_? (a\,b) \cdot t, a \#_? t\}\rangle = \mathcal{P}'$$

  Let $\langle \nabla', \sigma' \rangle$ be a solution in $\mathcal{U}_C(\mathcal{P}')$. Again, the interesting condition to be checked is the third condition in Def. 5.2. Since $\langle \nabla', \sigma' \rangle$ is a solution of $\mathcal{P}'$, one has that $\nabla' \vdash a \# t\sigma'$ and $\nabla' \vdash s\sigma' \approx_{\alpha,C} ((a\,b) \cdot t)\sigma'$. By Lem. 2.3, one has $((a\,b) \cdot t)\sigma' = (a\,b) \cdot (t\sigma')$. Hence, by application of the α-equivalence rule ($\approx_{\alpha,C}$ [**ab**]) in Fig. 2.4, one concludes that $\nabla' \vdash [a](s\sigma') \approx_{\alpha,C} [b](t\sigma')$, which by the definition of substitution action (Def. 2.24) can be written as $\nabla' \vdash ([a]s)\sigma' \approx_{\alpha,C} ([b]t)\sigma'$. Thus, $\mathcal{U}_C(\mathcal{P}') \subseteq \mathcal{U}_C(\mathcal{P})$.

Notice that in this case $\mathcal{U}_C(\mathcal{P}) \subseteq \mathcal{U}_C(\mathcal{P}')$ too; indeed, if $\nabla' \vdash ([a]s)\sigma' \approx_{\alpha,C} ([b]t)\sigma'$, by Def. 2.24, reverse application of the α-equivalence rule ($\approx_\alpha$ [**ab**]) (Lem. 4.2) and Lem. 2.3, one has that $\nabla' \vdash s\sigma' \approx_{\alpha,C} ((a\,b)\cdot t)\sigma'$ and $\nabla' \vdash a \# t\sigma'$.

- Rule ($\approx_?$ **inst**). Consider the reduction

$$\mathcal{P} = \langle \nabla, \mathcal{X}, \sigma, P \uplus \{\pi.X \approx_? t\}\rangle \Rightarrow_{\approx} \left\langle \nabla, \mathcal{X}, \sigma'', P\{X/\pi^{-1}\cdot t\} \cup \bigcup_{\substack{Y \in dom(\sigma''),\\ a\#Y\in\nabla}} \{a \#_? Y\sigma''\} \right\rangle = \mathcal{P}'$$

where $\sigma'' := \sigma\{X/\pi^{-1}\cdot t\}$ and $X \notin Var(t) \cup \mathcal{X}$.

Let $\langle \nabla', \sigma'\rangle \in \mathcal{U}_C(\mathcal{P}')$. First, the third condition in Def. 5.2 is analysed. Let $u \approx_? v$ be an equation in P. One has that $\nabla' \vdash u\{X/\pi^{-1}\cdot t\}\sigma' \approx_{\alpha,C} v\{X/\pi^{-1}\cdot t\}\sigma'$ and $\nabla' \vdash \sigma' \approx \sigma\{X/\pi^{-1}\cdot t\}\lambda$, for some λ. Thus, $\nabla' \vdash u\{X/\pi^{-1}\cdot t\}(\sigma\{X/\pi^{-1}\cdot t\}\lambda) \approx_{\alpha,C} v\{X/\pi^{-1}\cdot t\}(\sigma\{X/\pi^{-1}\cdot t\}\lambda)$, which implies $\nabla' \vdash u\{X/\pi^{-1}\cdot t\}\lambda \approx_{\alpha,C} v\{X/\pi^{-1}\cdot t\}\lambda$. For the last part the general assumption that $\mathcal{P}$ and $\mathcal{P}'$ are valid quadruples is used; hence, by Lem. 5.1, $dom(\sigma)$ does not intersect the set $Var(P) \cup Var(t) \cup \{X\}$. Thus, $\nabla' \vdash u\sigma\{X/\pi^{-1}\cdot t\}\lambda \approx_{\alpha,C} v\sigma\{X/\pi^{-1}\cdot t\}\lambda$, and finally, by the hypothesis $\nabla' \vdash \sigma' \approx \sigma\{X/\pi^{-1}\cdot t\}\lambda$, one concludes that $\nabla' \vdash u\sigma' \approx_{\alpha,C} v\sigma'$.

To conclude the analysis of the third condition, $\pi.X \approx_? t$ should be considered. One has that $\pi.X(\sigma\{X/\pi^{-1}\cdot t\}\lambda) = \pi\cdot(\pi^{-1}\cdot t)\lambda$. The last term corresponds to $t\lambda$ that is equal to $t(\sigma\{X/\pi^{-1}\cdot t\}\lambda)$. Since $\approx_{\{\alpha,C\}}$ is an equivalence relation (Cor. 4.1), by reflexivity one concludes that $\nabla' \vdash \pi.X\sigma' \approx_{\alpha,C} t\sigma'$.

The first condition in Def. 5.2 is immediate. The second condition is more interesting and depends on the third one. One needs to prove that for any $a \#_? u \in P$, $\nabla' \vdash a \# u\sigma'$. The proof proceeds by induction in u. The cases in which $u = \langle\rangle$, $u = \bar{b}$, $u = [a]v$ and $u = \phi.Y$, for $Y \neq X$, are immediate. The case in which $u = \bar{a}$ is not possible since it contradicts the hypothesis. The cases in which $u = fv$ or $u = \langle u_1, u_2\rangle$ and $u = [b]v$ follow by direct application of the induction hypothesis. The interesting case is when $u = \phi.X$ for which the hypothesis is that $\nabla' \vdash a \# \phi.X\{X/\pi^{-1}\cdot t\}\sigma'$. By application of the substitution one has $\nabla' \vdash a \# \phi\cdot(\pi^{-1}\cdot t)\sigma'$; by application of nominal properties for the freshness relation # this gives $\nabla' \vdash \pi\cdot\phi^{-1}\cdot a \# t\sigma'$. By freshness preservation (Lem. 4.4) and the fact that $\nabla' \vdash \pi.X\sigma' \approx_{\alpha,C} t\sigma'$ (proved in the analysis of the third condition of Def. 5.2) one obtains $\nabla' \vdash \pi\,\phi^{-1}\cdot a \# \pi.X\sigma'$; thus, by nominal properties one obtains $\nabla' \vdash \phi^{-1}\cdot a \# X\sigma'$ and finally that $\nabla' \vdash a \# \phi.X\sigma'$.

The analysis of the fourth condition in Def. 5.2 uses the hypothesis that there exists a λ such that $\nabla' \vdash \sigma''\lambda \approx \sigma'$ and, given that $\sigma'' := \sigma\{X/\pi^{-1}\cdot t\}$, for $\mathcal{P}$ one has that the substitution $\{X/\pi^{-1}\cdot t\}\lambda$ satisfies the requirement that $\nabla' \vdash \sigma\{X/\pi^{-1}\cdot t\}\lambda \approx \sigma'$.

- Finally, for the rule ($\approx_?$ **inv**) consider a derivation:

$$\mathcal{P} = \langle \nabla, \mathcal{X}, \sigma, P \uplus \{\pi.X \approx_? \pi'.X\}\rangle \Rightarrow_\approx \langle \nabla, \mathcal{X}, \sigma, P \cup \{\pi \oplus (\pi')^{-1}.X \approx_? X\}\rangle = \mathcal{P}'$$

Suppose that $\langle \nabla', \sigma' \rangle \in \mathcal{U}_C(\mathcal{P}')$. All conditions in Def. 5.2 hold trivially (in both directions) except the third. Suppose $\nabla' \vdash (\pi \oplus (\pi')^{-1}.X)\sigma' \approx_{\alpha,C} X\sigma'$, by substitution properties (Def. 2.24), this holds if and only if $\nabla' \vdash \pi \oplus (\pi')^{-1} \cdot (X\sigma') \approx_{\alpha,C} X\sigma'$, and by equivariance (Lem. 4.5) and the definition of action of substitutions (Def. 2.24), the last holds if and only if, $\nabla' \vdash \pi.X\sigma' \approx_{\alpha,C} \pi'.X\sigma'$. One concludes that $\mathcal{U}_C(\mathcal{P}) = \mathcal{U}_C(\mathcal{P}')$.

□

Definition 5.7 (Successful leaves). *Let $\langle \Delta, \mathcal{X}, P \rangle$ be a unification problem. A leaf in $\mathcal{T}_{\langle \Delta, \mathcal{X}, P \rangle}$ that is labelled with a quadruple $\mathcal{Q}$, where $\mathcal{U}_c(Q) \neq \emptyset$, is called a* successful leaf. *The set of successful leaves of $\mathcal{T}_{\langle \Delta, \mathcal{X}, P \rangle}$ is denoted by $SL(\mathcal{T}_{\langle \Delta, \mathcal{X}, P \rangle})$.*

Remark 5.5. *Observe that in the second phase of construction of the tree $\mathcal{T}_{\langle \Delta, \mathcal{X}, P \rangle}$ (item (ii) of Def. 5.6), for any quadruple, all equations are FP. In this case $\Rightarrow_\#$ acts only over freshness constraints $a \#_? t$ that are reducible, except when $t = \bar{a}$. Then the generated leaves in this phase must contain only FP equations and possibly freshness constraints in the form $a \#_? \bar{a}$. The following Lem. 5.5 uses this idea to characterise the set of successful leaves.*

Lemma 5.5 (Characterisation of successful leaves of $\mathcal{T}_{\langle \Delta, \mathcal{X}, P \rangle}$). *Let $\mathcal{P} = \langle \Delta, \mathcal{X}, P \rangle$ be a unification problem and $\mathcal{Q} = \langle \nabla, \mathcal{X}, \sigma, Q \rangle \in SL(\mathcal{T}_{\langle \Delta, \mathcal{X}, P \rangle})$, then Q contains only FP equations (and no freshness constraints), i.e., $Q = Q_{fp_\approx}$.*

Proof. This result is proved by case analysis on Q, and is formalised in file `C_Unif_Soundness`, Cor. `gen_successfull_leaves`. If there is a non FP equation in Q, say $s \approx_? t$, then by definition of $\mathcal{T}_{\langle \nabla, \mathcal{X}, P \rangle}$, $\mathcal{Q}$ should be a $\Rightarrow_\approx$-nf in $\mathcal{P}_{\Rightarrow_\approx}$. Since no rule of the relation $\Rightarrow_\approx$ applies to the equation $s \approx_? t$ in Q, one has only the following possibilities:

i) Neither s nor t are suspended variables and s and t are terms of different grammatical type (for instance an abstraction and a pair, or an atom term and a functional term).

ii) s and t are functional terms rooted by different function symbols.

iii) s and t are different atom terms.

iv) s is a suspension, say $\pi.X$, and $t \neq \pi'.X$, but $X \in Var(t) \cup \mathcal{X}$.

Since for all these cases there is no $\langle \Delta, \delta \rangle$ such that $\Delta \vdash s\delta \approx_{\alpha,C} t\delta$, one can conclude that $\mathcal{U}_C(\mathcal{Q}) = \emptyset$, which contradicts the hypothesis $\mathcal{Q} \in SL(\mathcal{T}_{\langle \Delta, \mathcal{X}, P \rangle})$.

In the case in which $Q_{\approx} = Q_{\text{fp}_{\approx}}$ and Q contains freshness constraints, there exists a quadruple $\mathcal{P}' = \langle \Delta', \mathcal{X}, \sigma', P' \rangle \in \mathcal{P}_{\Rightarrow_{\approx}}$ and $\mathcal{Q}$ is a $\Rightarrow_{\#}$-nf of $\mathcal{P}'$. The set P' can be split into sets of freshness constraints $P'_{\perp}$, and FP equations. The relation $\Rightarrow_{\#}$ changes only $P'_{\perp}$, and since freshness constraints in $\mathcal{Q}$ should be of the form $a \#_? \bar{a}$ (see Rmk. 5.5), and there is no $\langle \Delta, \delta \rangle$ such that $\Delta \vdash a \# \bar{a}\delta$, that is $\Delta \vdash a \# \bar{a}$, and also in this case $\mathcal{U}_C(\mathcal{Q}) = \emptyset$, contradicting again $\mathcal{Q} \in SL(\mathcal{T}_{\langle \Delta, \mathcal{X}, P \rangle})$.

Finally, by contradiction, one concludes that $Q = Q_{\text{fp}_{\approx}}$.

$\square$

Theorem 5.1 (Soundness of $\mathcal{T}_{\langle \Delta, \mathcal{X}, P \rangle}$)**.** *$\mathcal{T}_{\langle \Delta, \mathcal{X}, P \rangle}$ is correct, i.e., if $\mathcal{P}' = \langle \nabla, \mathcal{X}, \sigma, P' \rangle$ is the label of a leaf in $\mathcal{T}_{\langle \Delta, \mathcal{X}, P \rangle}$, then*

1. *$\mathcal{U}_C(\mathcal{P}') \subseteq \mathcal{U}_C(\langle \Delta, id, P, \rangle)$, and*

2. *if P' contains non FP equations or freshness constraints then $\mathcal{U}_C(\mathcal{P}') = \emptyset$.*

Proof. The first is proved by induction on the number of steps of $\Rightarrow_{\approx}$ and $\Rightarrow_{\#}$, using Lems. 5.3 and 5.4. The second is a direct application of Lem. 5.5. $\square$

To prove completeness (Thm. 5.2), a few auxiliary properties, given in Lem. 5.6 on the relation $_ \vdash _ \approx_{\alpha,C} _$ between substitutions required. These properties are formalised in file `Subst.v`, in Lems. `subst_sym`, `subst_trans`, `c_equiv_unif`, `c_equiv_unif_fresh`, `c_equiv_unif_2` and `subst_cancel_left`. Their proofs are obtained using the properties of equivalence and freshness preservation of $\approx_{\alpha,C}$ (Cor. 4.1 and Lem. 4.4) and Lem. 2.3.

Lemma 5.6 (Basic properties of nominal C-equivalence between substitutions)**.**

i) (Symmetry) $\Delta \vdash \sigma \approx_{\alpha,C} \delta$ implies $\Delta \vdash \delta \approx_{\alpha,C} \sigma$;

ii) (Transitivity) If $\Delta \vdash \sigma \approx_{\alpha,C} \delta$ and $\Delta \vdash \delta \approx_{\alpha,C} \lambda$, then $\Delta \vdash \sigma \approx_{\alpha,C} \lambda$;

iii) (Composition with instantiation) $\Delta \vdash \pi.X\sigma \approx_{\alpha,C} t\sigma$ implies $\Delta \vdash \{X/\pi^{-1} \cdot t\}\sigma \approx_{\alpha,C} \sigma$;

iv) (Freshness preservation) If $\Delta \vdash \sigma \approx_{\alpha,C} \delta$ and $\Delta \vdash a \# t\sigma$, then $\Delta \vdash a \# t\delta$;

v) (Compatibility with $\approx_{\alpha,C}$) If $\Delta \vdash \sigma \approx_{\alpha,C} \delta$ and $\Delta \vdash s\sigma \approx_{\alpha,C} t\sigma$, then $\Delta \vdash s\delta \approx_{\alpha,C} t\delta$;

vi) (Equivariance by substitutions) $\Delta \vdash \sigma \approx_{\alpha,C} \delta$ implies $\Delta \vdash \lambda\sigma \approx_{\alpha,C} \lambda\delta$.

The completeness theorem guarantees that the set of successful leaves provides a complete set of solutions. Its proof uses case analysis on the rules of the relations $\Rightarrow_{\approx}$ and $\Rightarrow_{\#}$ by an argumentation similar to the one used for Thm. 5.1. For $\Rightarrow_{\#}$ one has indeed

equivalence: $\mathcal{P} \Rightarrow_{\#} \mathcal{P}'$, implies $\mathcal{U}_C(\mathcal{P}) = \mathcal{U}_C(\mathcal{P}')$. The same is true for all rules of the relation $\Rightarrow_{\approx}$ except the branching rule ($\approx_?$ **C**), for which it is necessary to prove that all solutions of a quadruple reduced by ($\approx_?$ **C**) must belong to the set of solutions of one of its children quadruples.

Theorem 5.2 (Completeness of $\mathcal{T}_{\langle\Delta,\mathcal{X},P\rangle}$)**.** *Let $\mathcal{P} = \langle\Delta, \mathcal{X}, P\rangle$ and $\mathcal{T}_{\langle\Delta,\mathcal{X},P\rangle}$ be a unification problem and its derivation tree. Then $\mathcal{U}_C(\mathcal{P}) = \bigcup_{\mathcal{Q}\in SL(\mathcal{T}_{\langle\Delta,\mathcal{X},P\rangle})} \mathcal{U}_C(\mathcal{Q})$.*

Proof. This result is formalised in file `C_Unif_Completeness`, Lem. `equ_sys_compl`. From soundness (Thm. 5.1), $\mathcal{U}_C(\mathcal{P}) \supseteq \bigcup_{\mathcal{Q}\in SL(\mathcal{T}_{\langle\nabla,\mathcal{X},P\rangle})} \mathcal{U}_C(\mathcal{Q})$. The other inclusion is proved by induction on the size of subtrees of $\mathcal{T}_{\langle\nabla,\mathcal{X},P\rangle}$. This is done verifying that in the preservation lemmas for $\Rightarrow_{\#}$ and $\Rightarrow_{\approx}$ (Lems. 5.3 and 5.4) all rules, except ($\approx_?$ **C**) and ($\approx_?$ **inst**) preserve exactly the same sets of solutions.

- For one-step application of rule ($\approx_?$ **C**) on a quadruple $\mathcal{Q}$, labelling a node in $\mathcal{T}_{\langle\nabla,P\rangle}$, one has two sibling nodes labelled with quadruples $\mathcal{Q}_1$ and $\mathcal{Q}_2$ such that

$$\mathcal{Q} = \langle\Delta, \mathcal{X}, \sigma, Q \uplus \{f^C\langle s_0, s_1\rangle \approx_? f^C\langle t_0, t_1\rangle\}\rangle,$$
$$\mathcal{Q}_1 = \langle\Delta, \mathcal{X}, \sigma, Q \cup \{s_0, \approx_? t_0, s_1 \approx_? t_1\}\rangle \text{ and } \mathcal{Q}_2 = \langle\Delta, \mathcal{X}, \sigma, Q \cup \{s_0, \approx_? t_1, s_1 \approx_? t_0\}\rangle.$$

  If $\langle\Delta', \sigma'\rangle \in \mathcal{U}_C(\mathcal{Q})$ then it satisfies the four conditions in Def. 5.2 for $\mathcal{Q}$. It can be easily checked that $\langle\Delta', \sigma'\rangle$ satisfies the first, second and fourth conditions for both $\mathcal{Q}_1$ and $\mathcal{Q}_2$. Regarding the third condition, since it holds for $\mathcal{Q}$, it holds for any equation in Q and also $\Delta' \vdash f^C\langle s_0, s_1\rangle\sigma' \approx_{\alpha,C} f^C\langle t_0, t_1\rangle\sigma'$. From the last, by substitution action one has that $\Delta' \vdash f^C\langle s_0\sigma', s_1\sigma'\rangle \approx_{\alpha,C} f^C\langle t_0\sigma', t_1\sigma'\rangle$. Then, by Lem. 4.2 either $\Delta' \vdash s_0\sigma' \approx_{\alpha,C} t_0\sigma'$ and $\Delta' \vdash s_1\sigma' \approx_{\alpha,C} t_1\sigma'$, or $\Delta' \vdash s_0\sigma' \approx_{\alpha,C} t_1\sigma'$ and $\Delta' \vdash s_1\sigma' \approx_{\alpha,C} t_0\sigma'$. Thus, the third condition holds for $\mathcal{Q}_1$ or for $\mathcal{Q}_2$, and it can be concluded that $\langle\Delta', \sigma'\rangle \in \mathcal{U}_C(\mathcal{Q})$ if and only if $\langle\Delta', \sigma'\rangle \in \mathcal{U}_C(\mathcal{Q}_1)$ or $\langle\Delta', \sigma'\rangle \in \mathcal{U}_C(\mathcal{Q}_2)$. Therefore $\mathcal{U}_C(\mathcal{Q}) = \mathcal{U}_C(\mathcal{Q}_1) \cup \mathcal{U}_C(\mathcal{Q}_2)$. By induction hypothesis, the solutions of the successful leaves in the subtrees rooted by $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are exactly the set $\mathcal{U}_C(\mathcal{Q})$.

- For one-step application of rule ($\approx_?$ **inst**) on a quadruple $\mathcal{Q}$, labelling a node in $\mathcal{T}_{\langle\nabla,\mathcal{X},P\rangle}$, one has a sibling node labelled with a quadruple $\mathcal{Q}'$ such that

$$\mathcal{Q} = \langle\Delta, \mathcal{X}, \sigma, Q \uplus \{\pi.X \approx_? t\}\rangle \Rightarrow_{\approx} \left\langle \Delta, \mathcal{X}, \sigma'', Q\{X/\pi^{-1} \cdot t\} \cup \bigcup_{\substack{Y\in dom(\sigma''),\\ a\#Y\in\Delta}} \{a \#_? Y\sigma''\} \right\rangle = \mathcal{Q}'$$

  where $\sigma'' := \sigma\{X/\pi^{-1} \cdot t\}$ and $X \notin Var(t) \cup \mathcal{X}$.

  Suppose that $\langle\Delta', \sigma'\rangle \in \mathcal{U}_C(\mathcal{Q})$. It is checked that $\langle\Delta', \sigma'\rangle$ satisfies the four conditions in Def. 5.2 for $\mathcal{Q}'$.

– The **first condition**, that is $\Delta' \vdash \Delta\sigma'$, is the same for $\mathcal{Q}$ and $\mathcal{Q}'$.

For the **second**, **third** and **fourth condition**, observe that, by the third condition of the hypothesis $\langle \Delta', \sigma' \rangle \in \mathcal{U}_C(\mathcal{Q})$, one has $\Delta' \vdash \pi.X\sigma' \approx_{\alpha,C} t\sigma'$.

– Proving that the **second condition** holds for $\mathcal{Q}'$, requires proving that:

1. $a \#_? u \in Q$ implies $\Delta' \vdash a \# (u\{X/\pi^{-1} \cdot t\})\sigma'$ and
2. for all $Y \in dom(\sigma'')$ such that $a\#Y \in \Delta$, $\Delta' \vdash a \# Y\sigma''\sigma'$.

For the first subcase, applying Lem. 5.6, item (iii), one concludes that $\Delta' \vdash \{X/\pi^{-1} \cdot t\}\sigma' \approx_{\alpha,C} \sigma'$. Then, by symmetry and freshness preservation of equivalence of substitutions (Lem. 5.6, items (i) and (iv)), $\Delta' \vdash a \# u\sigma'$ implies $\Delta' \vdash a \# (u\{X/\pi^{-1} \cdot t\})\sigma'$. Then, using the second condition of hypothesis $\langle \Delta', \sigma' \rangle \in \mathcal{U}_C(\mathcal{Q})$ one concludes. For the second subcase, applying the first condition of hypothesis $\langle \Delta', \sigma' \rangle \in \mathcal{U}_C(\mathcal{Q})$ to hypothesis $a\#Y \in \Delta$, and using manipulations over the composition of substitutions, one obtains $\Delta' \vdash a \# Y\sigma''\sigma'$.

– The **third condition** of $\mathcal{Q}'$ is proved as follows: For $u \approx_? v \in Q'$, one must shows that $\Delta' \vdash s\sigma' \approx_{\alpha,C} t\sigma'$. Since $u \approx_? v$ must be in $Q\{X/\pi^{-1} \cdot t\}$ (because $\bigcup_{\substack{Y \in dom(\sigma''),\\ a\#Y \in \Delta}} \{a \#_? Y\sigma''\}$ contains only freshness constraints), there exists u_0 and v_0 such that $u_0 \approx_? v_0 \in Q$, $u = u_0\{X/\pi^{-1} \cdot t\}$ and $v = v_0\{X/\pi^{-1} \cdot t\}$. Then the objective became $\Delta' \vdash u_0\{X/\pi^{-1} \cdot t\}\sigma' \approx_{\alpha,C} v_0\{X/\pi^{-1} \cdot t\}\sigma'$. Applying Lem. 5.6, item (v), one must shows that $\Delta' \vdash \sigma' \approx_{\alpha,C} \{X/\pi^{-1} \cdot t\}\sigma'$ and $\Delta' \vdash u_0\sigma' \approx_{\alpha,C} v_0\sigma'$. The former is proved using Lem. 5.6, items (i) and (iii), and the latter is obtained by the third condition of hypothesis $\langle \Delta', \sigma' \rangle \in \mathcal{U}_C(\mathcal{Q})$.

– In the **fourth condition** of $\mathcal{Q}'$, using the fourth condition of the hypothesis $\langle \Delta', \sigma' \rangle \in \mathcal{U}_C(\mathcal{Q})$, that is given by $\Delta' \vdash \sigma\lambda \approx_{\alpha,C} \sigma'$, the objective is to provide a substitution δ such that $\Delta' \vdash \sigma\{X/\pi^{-1} \cdot t\}\delta \approx_{\alpha,C} \sigma'$. The substitution δ is chosen to be σ', and the objective became $\Delta' \vdash \sigma\{X/\pi^{-1} \cdot t\}\sigma' \approx_{\alpha,C} \sigma'$. Applying Lem. 5.6, item (ii), with the intermediate substitution $\sigma\sigma'$, one needs to show that $\Delta' \vdash \sigma\{X/\pi^{-1} \cdot t\}\sigma' \approx_{\alpha,C} \sigma\sigma'$ and $\Delta' \vdash \sigma\sigma' \approx_{\alpha,C} \sigma'$. For the former, one applies item (vi) followed by item (iii) of Lem. 5.6. In the latter, is used Lem. 5.6, item (ii), with the intermediate substitution $\sigma\sigma\lambda$. The new generated subgoals are $\Delta' \vdash \sigma\sigma' \approx_{\alpha,C} \sigma\sigma\lambda$ and $\Delta' \vdash \sigma\sigma\lambda \approx_{\alpha,C} \sigma'$. Lem. 5.6, item (vi) followed by (i), is applied to the former. The property of idempotence of σ is used to conclude the latter.

□

Corollary 5.1 (Generality of sucessful leaves)**.** *Let $\mathcal{P} = \langle \Delta, \mathcal{X}, P \rangle$ be a unification problem and $\langle \nabla'', \sigma' \rangle \in \mathcal{U}_C(\mathcal{P})$. Then there exists a successful leaf $\mathcal{Q} \in SL(\mathcal{T}_{\langle \Delta, \mathcal{X}, P \rangle})$ where*

$\mathcal{Q} = \langle \nabla, \mathcal{X}, \sigma, Q\rangle$ *such that* $\langle \nabla'', \sigma'\rangle \in \mathcal{U}_C(\mathcal{Q})$*, and hence,* $\nabla'' \vdash \nabla\sigma'$ *and there exists* λ *such that* $\nabla'' \vdash \sigma\lambda \approx_{\alpha,C} \sigma'$*.*

Proof. By Thm. 5.2, $\mathcal{U}_C(\mathcal{P}) = \bigcup_{\mathcal{P}' \in SL(\mathcal{T}_{\langle\Delta,\mathcal{X},P\rangle})} \mathcal{U}_C(\mathcal{P}')$. Then there exists $\mathcal{Q} \in SL(\mathcal{T}_{\langle\Delta,\mathcal{X},P\rangle})$ such that $\langle \nabla'', \sigma'\rangle \in \mathcal{U}_C(\mathcal{Q})$. Suppose $\mathcal{Q} = \langle \nabla, \mathcal{X}, \sigma, Q\rangle$. Then by the first and fourth conditions of the definition of solution (Def. 5.2) one has that $\nabla'' \vdash \nabla\sigma'$ and there exists λ such that $\nabla'' \vdash \sigma\lambda \approx_{\alpha,C} \sigma'$. □

In the formalisation, the strategy of application of $\Rightarrow_{\#}$ and $\Rightarrow_{\approx}$ was specified inductively through a relation $\Rightarrow_{\upsilon}$ denominated *unif-step* (Fig. 5.8). In this definition, $\Rightarrow_{\approx}$ is applied without restrictions by use of rule ($\upsilon_{\approx}$), but freshness constraints are reduced only when all equations of the problem are FP equations. This fact is expressed by the condition $P_{\approx} = P_{fp_{\approx}}$ in rule ($\upsilon_{\#}$). In the specification, the relation $\Rightarrow_{\upsilon}$ is in file `C_Unif.v` and it is given by the relation of Fig. 5.9. The definitions of *leaf* and *unif_path* (Fig. 5.10) are also specified in file `C_Unif.v`, respectively, as a normal form w.r.t. $\Rightarrow_{\upsilon}$, and as a reduction of zero or more steps from a unification problem to a *leaf*.

$$(\upsilon_{\approx})\frac{\mathcal{P} \Rightarrow_{\approx} \mathcal{Q}}{\mathcal{P} \Rightarrow_{\upsilon} \mathcal{Q}} \qquad\qquad (\upsilon_{\#})\frac{\mathcal{P} \Rightarrow_{\#} \mathcal{Q}}{\mathcal{P} \Rightarrow_{\upsilon} \mathcal{Q}}, P_{\approx} = P_{fp_{\approx}}$$

Figure 5.8: Unification-step

```
Inductive unif_step (varSet : set Var) : Triple → Triple → Prop :=
 | equ_unif_step : ∀ T T', equ_sys varSet T T' → unif_step varSet T T'
 | fresh_unif_step : ∀ T T', fixpoint_Problem (equ_proj (snd T)) →
   fresh_sys T T' → unif_step varSet T T' .
```

Figure 5.9: Specification of unif_step

```
Definition leaf (varSet : set Var) (T : Triple) :=
  NF _ (unif_step varSet) T .

Definition unif_path (varSet : set Var) (T T' : Triple) :=
  tr_clos _ (unif_step varSet) T T' ∧ leaf varSet T'.
```

Figure 5.10: Specification of leaf and unif_path

Theorem 5.3 (Main properties of $\Rightarrow_{\upsilon}$).

i) (Termination of $\Rightarrow_{\upsilon}$*) The relation* $\Rightarrow_{\upsilon}$ *is terminating;*

ii) (Decidability of $\Rightarrow_v$) Given a quadruple $\mathcal{P}$, it is possible to decide if $\mathcal{P}$ is a normal form w.r.t. $\Rightarrow_v$, or there exists $\mathcal{Q}$ such that $\mathcal{P} \Rightarrow_v \mathcal{Q}$;

iii) (Soundness of $\Rightarrow_v^*$) If $\mathcal{P} \Rightarrow_v^* \mathcal{Q}$, $\mathcal{Q}$ is a leaf and $\langle \nabla, \sigma \rangle \in \mathcal{U}_C(\mathcal{Q})$ then $\langle \nabla, \sigma \rangle \in \mathcal{U}_C(\mathcal{P})$;

iv) (Completeness of $\Rightarrow_v^*$) $\langle \nabla, \sigma \rangle \in \mathcal{U}_C(\mathcal{P})$ if and only if there exists a leaf $\mathcal{Q}$ such that $\mathcal{P} \Rightarrow_v^* \mathcal{Q}$ and $\langle \nabla, \sigma \rangle \in \mathcal{Q}$.

Proof.

i) The termination of $\Rightarrow_v$ is formalised in Cor. `unif_step_termination` of file `C_Unif_Termination` by case analysis on the derivation rules of $\Rightarrow_v$. Its proof is a direct consequence of Lem. 5.2;

ii) This result is formalised in Lem. `unif_step_NF_dec` of file `C_Unif_Termination`. Its proof is based on case analysis over $\mathcal{P} = \langle \nabla, \mathcal{X}, \sigma, P \rangle$. More specifically, the analysis is done over the set of nominal constraints P, verifying whether it is or not possible to reduce $\mathcal{P}$ by $\Rightarrow_v$;

iii) Soundness of $\Rightarrow_v^*$ is formalised in Thm. `c_unif_path_soundness` of file `C_Unif_Termination`, and its proof is a consequence of Lems. 5.3 and 5.4;

iv) Finally, the completeness of $\Rightarrow_v^*$ is formalised in Thm. `unif_path_compl` of file `C_Unif_Completeness` using item i), Lem. 5.3 and Thm. 5.2, and well-founded induction on derivations $\Rightarrow_v$.

□

Example 5.4. *This example illustrates the execution of the nominal C-unification algorithm for the initial problem*

$$\mathcal{P} = \langle \emptyset, \emptyset, id, \{[a]f\langle [b](X * Y), Z\rangle \approx_? [b]f\langle [a](\overline{a} * X), Z\rangle\}\rangle,$$

*where the set of protected variables is empty. Notice that the application of rule ($\approx_?$ **C**) generates two branches that are represented by items (i) and (ii) in the example (see Fig. 5.11). The algorithm generates the leaves*

$$\langle \{a\#Z\}, \emptyset, \{X/\overline{b}, Y/(a\,b).X\}, \{(a\,b).Z \approx_? Z\}\rangle, \text{ and}$$

$$\langle \{a\#Z\}, \emptyset, \{Y/\overline{b}\}, \{(a\,b).X \approx_? X,\ (a\,b).Z \approx_? Z\}\rangle.$$

By Thm. 5.3, the union of the solutions of these two leaves is equal to the set of solutions of the initial problem $\mathcal{P}$. As will be showed in Chap. 6, the complete set of

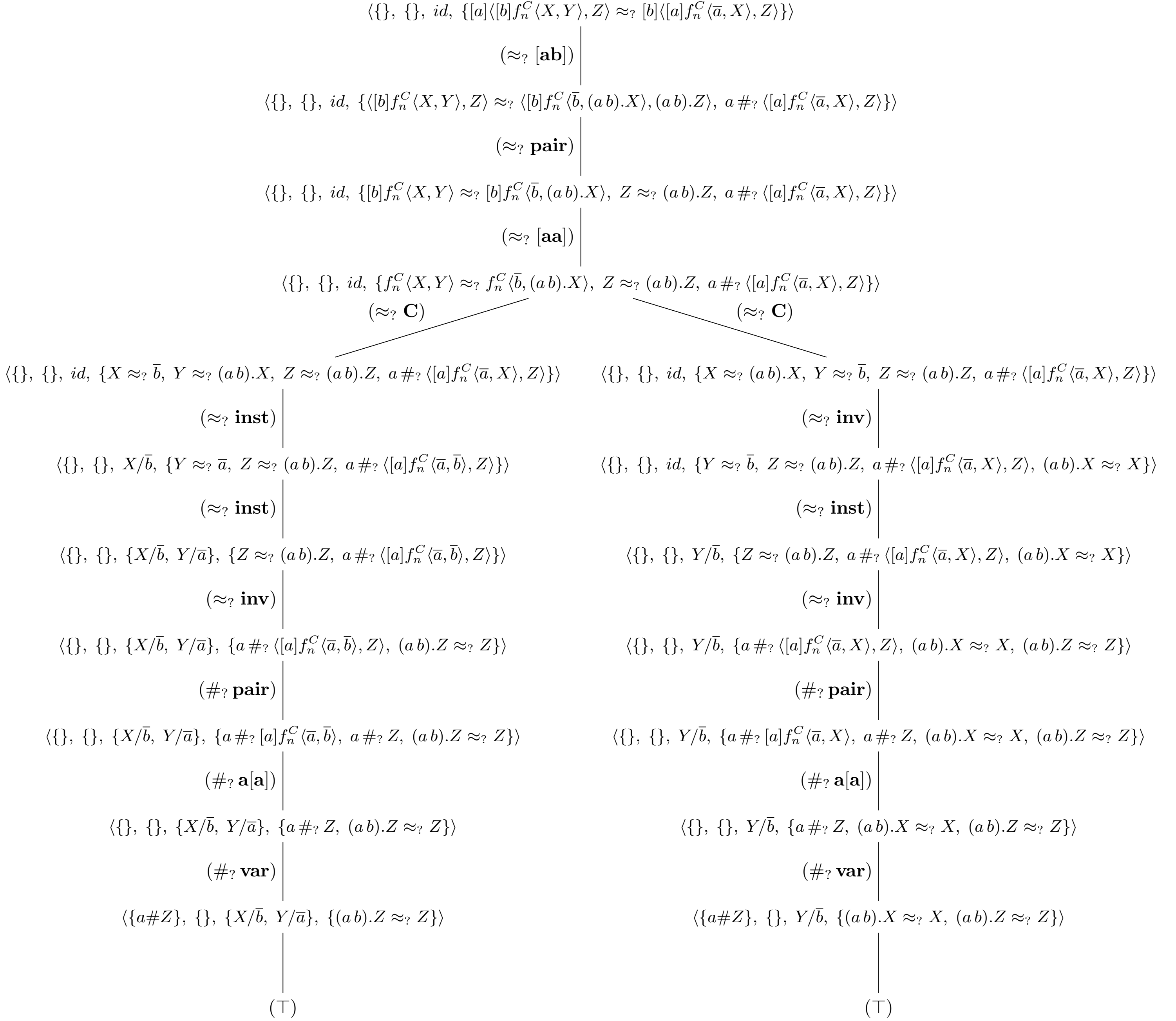

Figure 5.11: Derivation tree of Ex. 5.4.

solutions of $\langle\{a\#Z\},\emptyset,\{X/\overline{b},Y/(a\,b).X\},\{(a\,b).Z\approx_? Z\}\rangle$ *is unitary whereas the complete set of solutions of* $\langle\{a\#Z\},\emptyset,\{Y/\overline{b}\},\{(a\,b).X\approx_? X,\ (a\,b).Z\approx_? Z\}\rangle$ *is infinite.*

$$\mathcal{P} = \langle\emptyset,\emptyset,id,\{\ [a]f\langle[b](X*Y),Z\rangle\approx_?[b]f\langle[a](\overline{a}*X),Z\rangle\}\ \rangle$$

$$\Rightarrow_{(\approx_?[\mathbf{ab}])}\ \langle\emptyset,\emptyset,id,\{\ f\langle[b](X*Y),Z\rangle\approx_? f\langle[b](\overline{b}*(a\,b).X),(a\,b).Z\rangle,\ a\#_? f\langle[a](\overline{a}*X),Z\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{app})}\ \langle\emptyset,\emptyset,id,\{\ \langle[b](X*Y),Z\rangle\approx_?\langle[b](\overline{b}*(a\,b).X),(a\,b).Z\rangle,\ a\#_? f\langle[a](\overline{a}*X),Z\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{pair})}\ \langle\emptyset,\emptyset,id,\{\ [b](X*Y)\approx_?[b](\overline{b}*(a\,b).X),\ Z\approx_?(a\,b).Z,\ a\#_? f\langle[a](\overline{a}*X),Z\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?[\mathbf{aa}])}\ \langle\emptyset,\emptyset,id,\{\ X*Y\approx_?(\overline{b}*(a\,b).X),\ Z\approx_?(a\,b).Z,\ a\#_? f\langle[a](\overline{a}*X),Z\rangle\}\rangle$$

1.

$$\Rightarrow_{(\approx_?\mathbf{C})}\ \langle\emptyset,\emptyset,id,\{\ X\approx_?\overline{b},\ Y\approx_?(a\,b).X,\ Z\approx_?(a\,b).Z,\ a\#_? f\langle[a](\overline{a}*X),Z\rangle\}\rangle$$
$$\Rightarrow_{(\approx_?\mathbf{inst})}\ \langle\emptyset,\emptyset,\{X/\overline{b}\},\{\ Y\approx_?\overline{a},\ Z\approx_?(a\,b).Z,\ a\#_? f\langle[a](\overline{a}*X),Z\rangle\}\rangle$$
$$\Rightarrow_{(\approx_?\mathbf{inst})}\ \langle\emptyset,\emptyset,\{X/\overline{b},\ Y/\overline{a}\},\{\ Z\approx_?(a\,b).Z,\ a\#_? f\langle[a](\overline{a}*X),Z\rangle\}\rangle$$
$$\Rightarrow_{(\approx_?\mathbf{inv})}\ \langle\emptyset,\emptyset,\{X/\overline{b},\ Y/\overline{a}\},\{(a\,b).Z\approx_? Z,\ a\#_? f\langle[a](\overline{a}*X),Z\rangle\}\rangle$$
$$\Rightarrow_{(\#_?\,\mathbf{app})}\ \langle\emptyset,\emptyset,\{X/\overline{b},\ Y/\overline{a}\},\{(a\,b).Z\approx_? Z,\ a\#_?\langle[a](\overline{a}*X),Z\rangle\}\rangle$$
$$\Rightarrow_{(\#_?\,\mathbf{pair})}\ \langle\emptyset,\emptyset,\{X/\overline{b},\ Y/\overline{a}\},\{(a\,b).Z\approx_? Z,\ a\#_?[a](\overline{a}*X),\ a\#_? Z\}\rangle$$
$$\Rightarrow_{(\#_?\,\mathbf{a[a]})}\ \langle\emptyset,\emptyset,\{X/\overline{b},\ Y/\overline{a}\},\{(a\,b).Z\approx_? Z,\ a\#_? Z\}\rangle$$
$$\Rightarrow_{(\#_?\,\mathbf{var})}\ \langle\{a\#Z\},\ \emptyset,\{X/\overline{b},Y/\overline{a}\},\{(a\,b).Z\approx_? Z\}\rangle$$

2.

$$\Rightarrow_{(\approx_?\mathbf{C})}\ \langle\emptyset,\emptyset,id,\{\ X\approx_?(a\,b).X,\ Y\approx_?\overline{b},\ Z\approx_?(a\,b).Z,\ a\#_? f\langle[a](\overline{a}*X),Z\rangle\}\rangle$$
$$\Rightarrow_{(\approx_?\mathbf{inv})}\ \langle\emptyset,\emptyset,id,\{(a\,b).X\approx_? X,\ Y\approx_?\overline{b},\ Z\approx_?(a\,b).Z,\ a\#_? f\langle[a](\overline{a}*X),Z\rangle\}\rangle$$
$$\Rightarrow_{(\approx_?\mathbf{inst})}\ \langle\emptyset,\emptyset,\{Y/\overline{b}\},\{(a\,b).X\approx_? X,\ Z\approx_?(a\,b).Z,\ a\#_? f\langle[a](\overline{a}*X),Z\rangle\}\rangle$$
$$\Rightarrow_{(\approx_?\mathbf{inv})}\ \langle\emptyset,\emptyset,\{Y/\overline{b}\},\{(a\,b).X\approx_? X,\ (a\,b).Z\approx_? Z,\ a\#_? f\langle[a](\overline{a}*X),Z\rangle\}\rangle$$
$$\Rightarrow_{(\#_?\,\mathbf{app})}\ \langle\emptyset,\emptyset,\{Y/\overline{b}\},\{(a\,b).X\approx_? X,\ (a\,b).Z\approx_? Z,\ a\#_?\langle[a](\overline{a}*X),Z\rangle\}\rangle$$
$$\Rightarrow_{(\#_?\,\mathbf{pair})}\ \langle\emptyset,\emptyset,\{Y/\overline{b}\},\{(a\,b).X\approx_? X,\ (a\,b).Z\approx_? Z,\ a\#_?[a](\overline{a}*X),\ a\#_? Z\}\rangle$$
$$\Rightarrow_{(\#_?\,\mathbf{a[a]})}\ \langle\emptyset,\emptyset,\{Y/\overline{b}\},\{(a\,b).X\approx_? X,\ (a\,b).Z\approx_? Z,\ a\#_? Z\}\rangle$$
$$\Rightarrow_{(\#_?\,\mathbf{var})}\ \langle\{a\#Z\},\emptyset,\{Y/\overline{b}\},\{(a\,b).X\approx_? X,\ (a\,b).Z\approx_? Z\}\rangle$$

Remark 5.6 (Nominal C-unification is NP-complete)**.** *Observe that the proof that first-order C-unification is NP-complete (see Rmk. 2.5) is easily adapted to the context of nominal C-unification. Constants* c_0*,* c_1 *are replaced by atom terms* $\overline{a}$ *and* $\overline{b}$*, and the*

polynomial measure is changed to that used in Lem. 5.2. Remembering that this measure assures that the time of checking non-deterministicaly a solution for a unification problem is polynomial. The nominal C-unification problem for $\mathcal{C}$ *is given by* $\mathcal{P}_\mathcal{C} = \langle \emptyset, \{E_i | 1 \leq i \leq n\}\rangle$. *Simplifying* $\mathcal{P}_\mathcal{C}$ *would not add additional freshness constraints since the problem does not include abstractions. Thus, to conclude it is only necessary to check that* $\langle \emptyset, \sigma\rangle$ *is a solution for* $\mathcal{P}_\mathcal{C}$ *if and only if* σ *instantiates exactly one of the variables* X_{p_i}, X_{q_i} *and* X_{r_i} *in each equation with* $\overline{a}$ *and the other two with* $\overline{b}$*, which means that* $\mathcal{C}$ *has a solution.*

5.2 Nominal C-matching

In this section the attention is restricted to *nominal C-matching problems*: a nominal C-unification problem whose solutions should be applied only to the *lhs* of the nominal equations.

A nominal C-matching algorithm was specified through the *matching-step* rules presented in Fig. 5.12. Its specification is given in Fig. 5.13 and is in file `C_Matching.v`. These rules basically apply the rules in Figs. 5.1, 5.2 and 5.8, but now the set $\mathcal{X}$ of protected variables plays an important role and should be defined as the variables occurring in the *rhs* of the set of equational constraints P, in the input problem.

5.2.1 Basic notions on nominal C-matching and auxiliary formalised properties

This subsection shortly describes the main auxiliary lemmas related with nominal C-matching notions.

Definition 5.8 (Protected variables and nominal C-matching problems)**.** *The set of protected variables for a matching problem* $\langle \nabla, P\rangle$ *is the set of rhs* variables *of the equations in* P*, denoted by* $\mathtt{Rvar}(P)$*, i.e.,* $\mathtt{Rvar}(P) = \{X \mid s \approx_? t \in P \text{ and } X \in \mathtt{var}(t)\}$. *The quadruple associated with the nominal C-matching problem* $\langle \nabla, P\rangle$ *is given by* $\langle \nabla, \mathtt{Rvar}(P), id, P\rangle$.

$$(\mu_v)\ \frac{\mathcal{P} \Rightarrow_v \mathcal{Q}}{\mathcal{P} \Rightarrow_\mu \mathcal{Q}} \qquad (\mu_{\mathbf{fp}})\ \frac{\langle \nabla, \mathcal{X}, \sigma, P \uplus \{\pi.X \approx_? X\}\rangle}{\langle \nabla \cup \mathtt{dom}(\pi)\#X,\ \mathcal{X},\ \sigma,\ P\rangle}, P = P_{fp_\approx}$$

Figure 5.12: Matching-step

Remark 5.7. *Notice that, the standard nominal unification algorithm presented in Sec. 2.2 (of Chap. 2) can be seen as a particular case of the algorithm of Sec. 5.1, removing the*

```
Inductive match_step : Triple → Triple → Prop :=
  | match_to_unif_step : ∀ T T', unif_step (rhvars_Probl (snd T)) T T' →
    match_step T T'
  | match_fixpoint : ∀ C C' S P pi X,
    fixpoint_Problem P →
    pi ≠ [] →
    set_In (pi|.X ~? []|.X)) P →
    C' = set_union Context_eqdec C (fresh_context (dom_perm pi) X) →
    match_step (C, S, P) (C', S, P\(pi|.X ~? []|.X))) .
```

Figure 5.13: Specification of match_step

restriction of the protected variables of rule ($\approx_?$ **inst**) *and rule* ($\approx_?$ **C**)*, and also adding rule* ($\mu_{\mathbf{fp}}$) *of Fig. 5.12 to the equational system of Fig. 5.1.*

Derivation with rules of Fig. 5.12 is denoted by $\Rightarrow_\mu$. A quadruple $\mathcal{Q}$ is called a *matching leaf* if $\mathcal{Q}$ is a normal form w.r.t. $\Rightarrow_\mu$. A *matching path* is a reduction $\mathcal{P} \Rightarrow^*_\mu \mathcal{Q}$, where $\mathcal{Q}$ is a matching leaf. The specifications of matching leaf and matching path are in file `C_Matching.v` and are presented in Fig. 5.14.

```
Definition match_leaf (T : Triple) := NF _ match_step T .

Definition match_path (T T' : Triple) :=
  tr_clos _ match_step T T' ∧ match_leaf T' .
```

Figure 5.14: Specification of match_leaf and match_path

When solving a problem according to the rules in Fig. 5.12 using the protected variables given in Def. 5.8, the domain of a substitution that is a solution is disjoint from the set of *rhs* variables of the problem.

Definition 5.9 (Solution for a nominal C-matching problem)**.** *A* nominal C-matching solution *for a quadruple* $\mathcal{P}$ *of the form* $\langle\Delta, \mathtt{Rvar}(P), \delta, P\rangle$ *is a pair* $\langle\nabla, \sigma\rangle$*, where* $\mathtt{dom}(\sigma) \cap \mathtt{Rvar}(P) = \emptyset$*, and the following conditions are satisfied:*

i) $\nabla \vdash \Delta\sigma$*;*

ii) $\nabla \vdash a \# t\sigma$*, if* $a \#_? t \in P$*;*

iii) $\nabla \vdash s\sigma \approx_{\alpha,C} t$*, if* $s \approx_? t \in P$*;*

iv) there is a substitution λ *such that* $\nabla \vdash \delta\lambda \approx_{\alpha,C} \sigma$*.*

A nominal C-matching solution *for the problem* $\langle\Delta, \mathcal{X}, P\rangle$ *is a solution for* $\langle\Delta, \mathtt{Rvar}(P), id, P\rangle$*, its associated nominal C-matching problem. The* solution set *for a matching problem* $\mathcal{P}$ *is denoted by* $\mathcal{M}_C(\mathcal{P})$*.*

Lemma 5.7 ($\mathcal{U}_C$ and $\mathcal{M}_C$ equivalence)**.** *Let $\mathcal{P} = \langle \Delta, \texttt{Rvar}(P), \delta, P\rangle$ be a quadruple. Then, $\langle \nabla, \sigma\rangle \in \mathcal{U}_C(\mathcal{P})$ if and only if $\langle \nabla, \sigma\rangle \in \mathcal{M}_C(\mathcal{P})$.*

Proof. The formalisation follows straightforwardly from the definitions of $\mathcal{U}_C(\mathcal{P})$ and $\mathcal{M}_C(\mathcal{P})$, and is given in file `C_Matching.v`, Lem. `unif_match_sol_equiv`. □

Remark 5.8. *Despite the fact that the reduction rules (Figs. 5.2, 5.1, and 5.12) preserve the set $\mathcal{X}$ of protected variables given as input, in the following formalised results, for quadruples of the form $\mathcal{P} = \langle \Delta,\ \mathcal{X}, \delta, P\rangle$ and $\mathcal{Q} = \langle \Delta',\ \mathcal{X}, \delta', Q\rangle$, where $\mathcal{P} \Rightarrow_\mu \mathcal{Q}$, the sets $\texttt{Rvar}(P)$ and $\texttt{Rvar}(Q)$ will be considered. These sets change after reduction steps using rules such as ($\approx_?$ **refl**), ($\approx_?$ **inst**) and ($\mu_{\mathbf{fp}}$), but as Lem. 5.8 shows, $\texttt{Rvar}(Q) \subseteq \texttt{Rvar}(P)$, therefore if $\texttt{Rvar}(P) \subseteq \mathcal{X}$ then $\texttt{Rvar}(Q) \subseteq \mathcal{X}$. Since in the matching algorithm $\mathcal{X}$ is the set of* rhs *variables of the input problem, for each quadruple in a derivation $\mathcal{P}_0 \Rightarrow_\mu \ldots \Rightarrow_\mu \mathcal{P}_n$ the* rhs *variables are in the protected set.*

Lemma 5.8 (Preservation of `Rvar` by $\Rightarrow_\mu$)**.** *Let $\mathcal{P} = \langle \Delta, \mathcal{X}, \delta, P\rangle$ and $\mathcal{Q} = \langle \Delta', \mathcal{X}, \delta', Q\rangle$ such that $\mathcal{P} \Rightarrow_\mu \mathcal{Q}$. Then $\texttt{Rvar}(Q) \subseteq \texttt{Rvar}(P)$.*

Proof. The formalisation follows by case analysis on the $\Rightarrow_\mu$ reduction and it presented in file `C_Matching.v`, Lem. `match_step_rh_vars`. □

Corollary 5.2 (Intersection emptyness preservation with *rhs* variables by $\Rightarrow_\mu$)**.** *Let $\mathcal{P}$ and $\mathcal{Q}$ be two quadruples, $\langle \Delta, \mathcal{X}, \delta, P\rangle$ and $\langle \Delta', \mathcal{X}, \delta', Q\rangle$, respectively, and $\mathcal{Y}$ be an arbitrary set of variables. If $\texttt{Rvar}(P) \cap \mathcal{Y} = \emptyset$ and $\mathcal{P} \Rightarrow_\mu \mathcal{Q}$, then $\texttt{Rvar}(Q) \cap \mathcal{Y} = \emptyset$.*

Proof. This is indeed an easy set theoretically based corollary of Lem. 5.8, which is specified in file `C_Matching.v`, Cor. `match_step_rh_inter_empty`. □

Corollary 5.3 (Preservation of valid quadruples by $\Rightarrow_\mu$)**.** *If $\mathcal{P} \Rightarrow_\mu \mathcal{Q}$ and $\mathcal{P}$ is valid then $\mathcal{Q}$ is also valid.*

Proof. The formalisation follows from Lem. 5.1 and is given in file `C_Matching.v`,
Lem. `match_step_valid_preserv`. □

Lemma 5.9 (Decidability of $\Rightarrow_\mu$)**.** *For every quadruple $\mathcal{P}$ it is possible to decide whether there exists $\mathcal{Q}$ such that $\mathcal{P} \Rightarrow_\mu \mathcal{Q}$. Thus, also it is possible to decide if $\mathcal{P}$ is a leaf.*

Proof. The formalisation is obtained by decidability of the relation $\Rightarrow_v$ (item (i) of Thm. 5.3), which is given in file `C_Matching.v`, Lem. `match_step_NF_dec`. □

5.2.2 Main formalised results for nominal C-matching

Theorem 5.4 (Termination of $\Rightarrow_\mu$)**.** *The relation $\Rightarrow_\mu$ is terminating.*

Proof. The proof is by case analysis on the derivation rules of the relation $\Rightarrow_\mu$, and uses a lexicographic measure over sets of equation and freshness constraints. It is contained in file `C_Matching.v`, Lem. `match_step_termination`. The measure is given by

$$\left\langle\ |\mathtt{var}(P)|,\ \sum_{s \approx_? t \in P} |s| + |t|,\ |P_{\approx}/P_{fp_{\approx}}|,\ \sum_{a \#_? s \in P} |s|\ \right\rangle$$

Let $\mathcal{P} = \langle \Delta, \mathcal{X}, \delta, P\rangle$ and $\mathcal{Q} = \langle \nabla, \mathcal{X}, \sigma, Q\rangle$ such that $\mathcal{P} \Rightarrow_\mu \mathcal{Q}$.

For the case of rule (μ_v), Thm. 5.3 item (ii) is applied.

For the case of an application of rule $(\mu_{\mathbf{fp}})$, one observes that:

1. $|\mathtt{var}(Q)| \leq |\mathtt{var}(P)|$,
2. $\sum_{s \approx_? t \in Q} |s| + |t| < \sum_{s \approx_? t \in P} |s| + |t|$,
3. $|Q_{\approx}/Q_{fp_{\approx}}| = |P_{\approx}/P_{fp_{\approx}}|$ and
4. $\sum_{a \#_? s \in Q} |s| = \sum_{a \#_? s \in P} |s|$.

Therefore the measure also decreases in this case, which concludes the proof. □

Lemma 5.10 (Preservation of solutions by $\Rightarrow_\mu$)**.** *Let $\mathcal{P} = \langle \Delta, \mathcal{X}, \delta, P\rangle$ a valid quadruple and $\mathcal{Q} = \langle \Delta', \mathcal{X}, \delta', Q\rangle$, if $\mathtt{Rvar}(P) \cap \mathtt{dom}(\sigma) = \emptyset$, $\mathcal{P} \Rightarrow_\mu \mathcal{Q}$ and $\langle \nabla, \sigma\rangle \in \mathcal{M}_C(\mathcal{Q})$, then $\langle \nabla, \sigma\rangle \in \mathcal{M}_C(\mathcal{P})$.*

Proof. The proof is by case analysis on the derivation rules of $\Rightarrow_\mu$, and is formalised in file `C_Matching.v`, Lem. `match_step_preserv`. Notice that hypothesis $\mathtt{Rvar}(P) \cap \mathtt{dom}(\sigma) = \emptyset$ is in fact necessary, since from $\langle \nabla, \sigma\rangle \in \mathcal{M}_C(\mathcal{Q})$ it is impossible to derive such statement which is a necessary condition to show that $\langle \nabla, \sigma\rangle \in \mathcal{M}_C(\mathcal{P})$. According Def. 5.9, one has $\mathtt{Rvar}(Q) \cap \mathtt{dom}(\sigma) = \emptyset$. From this, the hypothesis $\mathtt{Rvar}(P) \cap \mathtt{dom}(\sigma) = \emptyset$ and using Lems. 5.3 (item (iii)) and 5.7 one concludes the case of rule (μ_v). For the case of rule $(\mu_{\mathbf{fp}})$, one needs to conclude the conditions of Def. 5.9 for the pair $\langle \nabla, \sigma\rangle$ w.r.t. $\mathcal{P}$. Condition (iv) is trivially satisfied. The first condition is proved just observing that every constraint $a\#X$ in Δ is also in $\Delta \cup \mathtt{dom}(\pi)$. The second condition is easily proved from the fact that if $a \#_? s \in P \uplus \{\pi.X \approx_? X\}$ then $a \#_? s \in P$. Then, one applies the hypothesis $\langle \nabla, \sigma\rangle \in \mathcal{M}_C(\mathcal{Q})$ using Def. 5.9, item (ii), to conclude. The third condition is proved by analysis of two cases. The first case is when $s \approx_? t \in P \uplus \{\pi.X \approx_? X\}$ being the equation $s \approx_? t$ equal to $\pi.X \approx_? X$. In this case, one starts proving the statement $X \cap \mathtt{dom}(\sigma) = \emptyset$ using the hypothesis $\mathtt{Rvar}(P \uplus \{\pi.X \approx_? X\}) \cap \mathtt{dom}(\sigma) = \emptyset$.

From this, $(\pi.X)\sigma$ can be replaced by $\pi.X$ in the objective $\nabla \vdash \pi X\sigma \approx_{\alpha,C} X$, remaining to prove that $\nabla \vdash \pi.X \approx_{\alpha,C} X$. Then, using the condition (i) of Def. 5.9 of hypothesis $\langle\nabla,\sigma\rangle \in \mathcal{M}_C(\mathcal{Q})$ one has that $\nabla \vdash (\Delta \cup \mathtt{dom}(\pi)\#X)\sigma$. Since $X \notin \mathtt{dom}(\sigma)$, one concludes that $\mathtt{dom}(\pi)\#X \subseteq \nabla$ and then the objective is proved using the definition of $\approx_{\{\alpha,C\}}$ for the case of suspensions. The second case is when $s \approx_? t \in P$. This case is trivial, and uses hypothesis $\langle\nabla,\sigma\rangle \in \mathcal{M}_C(\mathcal{Q})$ with Def. 5.9, item (iii). □

Theorem 5.5 (Completeness of $\Rightarrow_\mu$). *Let $\mathcal{P} = \langle\Delta,\mathcal{X},\delta,P\rangle$ a valid quadruple that is not a matching leaf, if* $\mathtt{Rvar}(P) \cap \mathtt{dom}(\sigma) = \emptyset$*, then $\langle\nabla,\sigma\rangle \in \mathcal{M}_C(\mathcal{P})$ if and only if there exists $\mathcal{Q}$ such that $\mathcal{P} \Rightarrow_\mu \mathcal{Q}$ and $\langle\nabla,\sigma\rangle \in \mathcal{M}_C(\mathcal{Q})$.*

Proof. This lemma is formalised in file `C_Matching.v`, Lem. `match_step_compl`. Necessity is proved by case analysis on the derivation rules of $\Rightarrow_\mu$. Lem. 5.9 is applied to the premise that $\mathcal{P}$ is not a matching leaf to obtain that there exists $\mathcal{Q}'$ such that $\mathcal{P} \Rightarrow_\mu \mathcal{Q}'$. Then for the case of rule (μ_v), using Lems. 5.3 (item (iv)) and 5.7 it is proved the assertion that there exists $\mathcal{Q}''$ such that $\mathcal{P} \Rightarrow_v \mathcal{Q}''$ and $\langle\nabla,\sigma\rangle \in \mathcal{U}_\mathcal{C}(\mathcal{Q}')$. From this, using again Lem. 5.7, applying rule (μ_v) and using Cor. 5.2 one concludes. For the case of rule $(\mu_{\mathbf{fp}})$, $\mathcal{P} = \langle\Delta,\mathcal{X},\delta,P' \uplus \{\pi.X \approx_? X\}\rangle$ with $P' = P'_{fp_\approx}$. The quadruple $\mathcal{Q} = \langle\Delta \cup \mathtt{dom}(\pi)\#X,\mathcal{X},\delta,P\rangle$ will be a witness. Thus, $\mathcal{P} \Rightarrow_\mu \mathcal{Q}$ follows by an application of rule $(\mu_{\mathbf{fp}})$. To prove that $\langle\nabla,\sigma\rangle \in \mathcal{M}_C(\mathcal{Q})$, one has to show that the conditions of Def. 5.9 are satisfied, having as hypothesis that $\langle\nabla,\sigma\rangle \in \mathcal{M}_C(\mathcal{P})$. Conditions (ii), (iii) and (iv) are trivially verified and intersection emptyness is proved using Cor. 5.2. For condition (i), a constraint $a\#X$ is chosen that is in $\Delta \cup \mathtt{dom}(\pi)\#X$ to analyse if it is either in Δ or $\mathtt{dom}(\pi)\#X$. If $a\#X$ is in Δ the proof is trivial, otherwise one first proves the assertion that $\{X\} \cap \mathtt{dom}(\sigma) = \emptyset$ from the hypotheses that $\mathcal{P}$ is valid and $\mathtt{Rvar}(P) \cap \mathtt{dom}(\sigma) = \emptyset$. This allows to replace every $X\sigma$ and every $(\pi.X)\sigma$, respectively, just by X and $\pi.X$, because $X \notin \mathtt{dom}(\sigma)$. Since $\pi.X \approx_? X$ is in $P \uplus \{\pi.X \approx_? X\}$ and $\langle\nabla,\sigma\rangle \in \mathcal{M}_C(\mathcal{P})$, one has that $\nabla \vdash (\pi.X)\sigma \approx_{\alpha,C} X$, therefore $\nabla \vdash \pi.X \approx_{\alpha,C} X$ and then $\mathtt{dom}(\pi)\#X \subseteq \nabla$. On the other hand, having $a \in \mathtt{dom}(\pi)$ as hypothesis, one has to prove that $\nabla \vdash a\#X\sigma$, which is the same as $\nabla \vdash a\#X$. Using the fact that $\mathtt{dom}(\pi)\#X \subseteq \nabla$, one concludes.

Sufficiency is formalised as a direct consequence of Lem. 5.10. □

Theorem 5.6 (Soundness of $\Rightarrow_\mu^*$). *Let $\mathcal{P} = \langle\Delta,\mathcal{X},\delta,P\rangle$ be a valid quadruple and $\mathcal{P} \Rightarrow_\mu^* \mathcal{Q}$. If $\mathcal{Q}$ is a matching leaf and $\langle\nabla,\sigma\rangle \in \mathcal{M}_C(\mathcal{Q})$ such that* $\mathtt{Rvar}(P) \cap \mathtt{dom}(\sigma) = \emptyset$ *then $\langle\nabla,\sigma\rangle \in \mathcal{M}_C(\mathcal{P})$.*

Proof. The proof uses Cors. 5.3 and 5.2 and Lem. 5.10, and is done by induction on the number of steps of $\Rightarrow_\mu$. It is given in file `C_Matching.v`, Thm. `match_path_soundness`. If $\mathcal{P} = \mathcal{Q}$ the proof is trivial. In the case where $\mathcal{P} \Rightarrow_\mu \mathcal{Q}$, Lem. 5.10 is applied to

conclude. When $\mathcal{P} \Rightarrow_\mu \mathcal{R}$ and $\mathcal{R} \Rightarrow_\mu^* \mathcal{Q}$, one uses Lem. 5.10, IH and Lems. 5.3 and 5.2 to conclude. □

Theorem 5.7 (Completeness of $\Rightarrow_\mu^*$)**.** *Let $\mathcal{P} = \langle \Delta, \mathcal{X}, \delta, P \rangle$ be a valid quadruple and $\langle \nabla, \sigma \rangle \in \mathcal{M}_C(\mathcal{P})$. Then there exists a matching leaf $\mathcal{Q}$ such that $\mathcal{P} \Rightarrow_\mu^* \mathcal{Q}$ and $\langle \nabla, \sigma \rangle \in \mathcal{M}_C(\mathcal{Q})$.*

Proof. The formalisation follows by well-founded induction on the number of applications of $\Rightarrow_\mu$. It is formalised in file `C_Matching.v`, Thm. `match_path_compl`. Also, Lem. 5.9 is applied in the analysis of the cases where either $\mathcal{P}$ is a matching leaf or there exists Q' such that $\mathcal{P} \Rightarrow_\mu \mathcal{Q}'$. If $\mathcal{P}$ is a matching leaf then $\mathcal{P} = \mathcal{Q}$ and the proof is completed. If there exists Q' such that $\mathcal{P} \Rightarrow_\mu \mathcal{Q}'$, one applies Lem. 5.9 to obtain that $\mathcal{P}$ is not a matching leaf. Lem. 5.5 is applied to the premise that $\mathcal{P}$ is not a matching leaf. From this and the hypothesis $\langle \nabla, \sigma \rangle \in \mathcal{M}_C(\mathcal{P})$ one obtains that there exists $\mathcal{Q}'$ such that $\mathcal{P} \Rightarrow_\mu \mathcal{Q}'$.

The IH is established as the following statement: $\forall \mathcal{R}$ valid, if $\mathcal{P} \Rightarrow_\mu \mathcal{R}$, $\texttt{Rvar}(\mathcal{R}) \cap \texttt{dom}(\sigma)$ and $\langle \nabla, \sigma \rangle \in \mathcal{M}_C(\mathcal{R})$, then there exists $\mathcal{S}$, such that $\mathcal{R} \Rightarrow_\mu^* \mathcal{S}$ and $\langle \nabla, \sigma \rangle \in \mathcal{M}_C(\mathcal{S})$.

This is applied to the hypothesis $\mathcal{P} \Rightarrow_\mu \mathcal{Q}'$ to conclude that there exists $\mathcal{Q}$, such that $\mathcal{Q}' \Rightarrow_\mu^* \mathcal{Q}$ and $\langle \nabla, \sigma \rangle \in \mathcal{M}_C(\mathcal{Q}')$. The other premises of IH are achieved with the auxiliary results given by Lem. 5.3 and Cor. 5.2. Finally, by case analysis on the statement $\mathcal{Q}' \Rightarrow_\mu^* \mathcal{Q}$, one concludes. □

Remark 5.9. *Let $\mathcal{P} = \langle \Delta, \texttt{Rvar}(P), id, P \rangle$ and $\mathcal{Q} = \langle \nabla, \texttt{Rvar}(P), \sigma, Q \rangle$ be, respectively, a matching problem and a quadruple, so that $\mathcal{P} \Rightarrow_\mu^* \mathcal{Q}$. If $\pi.X \approx_? X \in Q$, then the nominal C-matching algorithm eliminates this FP equation by a further application of rule $(\mu_{\mathbf{fp}})$. From this and Lem. 5.5, one concludes that successful matching leaves must have an empty set of nominal constraints, and obviously it does not contain any FP equation (Lem. 5.8). This is the reason why nominal C-matching problems are considered to have only finite sets of solutions.*

Theorem 5.8 (Characterisation of successful matching leaves)**.** *Let $\mathcal{Q} = \langle \Delta, \mathcal{X}, \delta, Q \rangle$ a matching leaf, if $\mathcal{Q}$ is a proper problem and there exists $\langle \nabla, \sigma \rangle \in \mathcal{M}_C(\mathcal{Q})$, then $Q = \emptyset$.*

Proof. This result is formalised in file `C_Matching.v`, Thm. `match_successfull_leaves_ch`. First one proves the assertion that Q is a FP problem. This statement is proved using Lems. 5.5 and 5.7, and rule (μ_v) of the definition of matching-step (Fig. 5.12). Therefore, if Q is a fixed problem it must be equal to the empty set, otherwise $\mathcal{Q}$ could be reduced by an application of rule $(\mu_{\mathbf{fp}})$ of Fig. 5.12, which contradicts the fact that $\mathcal{Q}$ is a matching leaf. □

$$\langle\{\},\ \{X,Z\},\ id,\ \{[a]\langle[b]f_n^C\langle X,Y\rangle,Z\rangle \approx_? [b]\langle[a]f_n^C\langle\overline{a},X\rangle,Z\rangle\}\rangle$$

$(\approx_?\ [\mathbf{ab}])$

$$\langle\{\},\ \{X,Z\},\ id,\ \{\langle[b]f_n^C\langle X,Y\rangle,Z\rangle \approx_? \langle[b]f_n^C\langle\overline{b},(a\,b).X\rangle,(a\,b).Z\rangle,\ a\#_?\langle[a]f_n^C\langle\overline{a},X\rangle,Z\rangle\}\rangle$$

$(\approx_?\ \mathbf{pair})$

$$\langle\{\},\ \{X,Z\},\ id,\ \{[b]f_n^C\langle X,Y\rangle \approx_? [b]f_n^C\langle\overline{b},(a\,b).X\rangle,\ Z\approx_?(a\,b).Z,\ a\#_?\langle[a]f_n^C\langle\overline{a},X\rangle,Z\rangle\}\rangle$$

$(\approx_?\ [\mathbf{aa}])$

$$\langle\{\},\ \{X,Z\},\ id,\ \{f_n^C\langle X,Y\rangle \approx_? f_n^C\langle\overline{b},(a\,b).X\rangle,\ Z\approx_?(a\,b).Z,\ a\#_?\langle[a]f_n^C\langle\overline{a},X\rangle,Z\rangle\}\rangle$$

Left branch $(\approx_?\ \mathbf{C})$:

$$\langle\{\},\ \{X,Z\},\ id,\ \{X\approx_?\overline{b},\ Y\approx_?(a\,b).X,\ Z\approx_?(a\,b).Z,\ a\#_?\langle[a]f_n^C\langle\overline{a},X\rangle,Z\rangle\}\rangle$$

$(\bot)$

Right branch $(\approx_?\ \mathbf{C})$:

$$\langle\{\},\ \{X,Z\},\ id,\ \{X\approx_?(a\,b).X,\ Y\approx_?\overline{b},\ Z\approx_?(a\,b).Z,\ a\#_?\langle[a]f_n^C\langle\overline{a},X\rangle,Z\rangle\}\rangle$$

$(\approx_?\ \mathbf{inv})$

$$\langle\{\},\ \{X,Z\},\ id,\ \{Y\approx_?\overline{b},\ Z\approx_?(a\,b).Z,\ a\#_?\langle[a]f_n^C\langle\overline{a},X\rangle,Z\rangle,\ (a\,b).X\approx_? X\}\rangle$$

$(\approx_?\ \mathbf{inst})$

$$\langle\{\},\ \{X,Z\},\ Y/\overline{b},\ \{Z\approx_?(a\,b).Z,\ a\#_?\langle[a]f_n^C\langle\overline{a},X\rangle,Z\rangle,\ (a\,b).X\approx_? X\}\rangle$$

$(\approx_?\ \mathbf{inv})$

$$\langle\{\},\ \{X,Z\},\ Y/\overline{b},\ \{a\#_?\langle[a]f_n^C\langle\overline{a},X\rangle,Z\rangle,\ (a\,b).X\approx_? X,\ (a\,b).Z\approx_? Z\}\rangle$$

$(\#_?\ \mathbf{pair})$

$$\langle\{\},\ \{X,Z\},\ Y/\overline{b},\ \{a\#_?[a]f_n^C\langle\overline{a},X\rangle,\ a\#_? Z,\ (a\,b).X\approx_? X,\ (a\,b).Z\approx_? Z\}\rangle$$

$(\#_?\ \mathbf{a[a]})$

$$\langle\{\},\ \{X,Z\},\ Y/\overline{b},\ \{a\#_? Z,\ (a\,b).X\approx_? X,\ (a\,b).Z\approx_? Z\}\rangle$$

$(\#_?\ \mathbf{var})$

$$\langle\{a\#Z\},\ \{X,Z\},\ Y/\overline{b},\ \{(a\,b).X\approx_? X,\ (a\,b).Z\approx_? Z\}\rangle$$

$(\mu_{\mathbf{fp}})$

$$\langle\{a\#X,\ b\#X,\ a\#Z\},\ \{X,Z\},\ Y/\overline{b},\ \{(a\,b).Z\approx_? Z\}\rangle$$

$(\mu_{\mathbf{fp}})$

$$\langle\{b\#Z,\ a\#X,\ b\#X,\ a\#Z\},\ \{X,Z\},\ Y/\overline{b},\ \{\}\rangle$$

$(\top)$

Figure 5.15: Derivation tree of Ex. 5.5

Example 5.5 (Nominal C-matching). *This example is similar to Ex. 5.4, but now the set of protected variables is equal to the* rhs *variables of the initial problem, that is* $\{X, Z\}$*. This results in the execution of the nominal C-matching algorithm that provides the matching leaves*

$$\mathcal{Q}_1 = \langle \emptyset, \{X, Z\}, id, \{Y/(a\,b).X\}\rangle\{X \approx_? \overline{b},\ (a\,b).Z \approx_? Z,\ a \#_? f\langle [a](\overline{a} * X), Z\rangle\},$$

$$\textit{and, } \mathcal{Q}_2 = \langle \{a\#X, b\#X, a\#Z, b\#Z\}, \{X, Z\}, \{Y/\overline{b}\}, \emptyset\rangle.$$

Since X *is a protect variable,* $\mathcal{Q}_1$ *has no solution due the fact that the equation* $X \approx_? \overline{b}$ *cannot be solved since* X *cannot be instantiated. Notice that, in the generated tree, by the OCaml implementation (see Fig. 5.15), the* lhs *branch does not reach a leaf. Indeed, once the constraint* $X \approx_? \overline{b}$ *is detected, that branch fails.*

$\mathcal{Q}_2$ *has just one solution given by* $\langle \{a\#X, b\#X, a\#Z, b\#Z\}, \{Y/\overline{b}\}\rangle$*. Thms. 5.6 and 5.7 show that this solution is the unique C-matching solution for the initial problem.*

$$\mathcal{P} = \langle \emptyset,\ \{X, Z\}, id, \{\ [a]f\langle [b](X * Y), Z\rangle \approx_? [b]f\langle [a]((\overline{a} * X)), Z\rangle\ \}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{[ab]})} \langle \emptyset,\ \{X, Z\}, id, \{\ f\langle [b](X * Y), Z\rangle \approx_? f\langle [b](\overline{b} * (a\,b).X), (a\,b).Z\rangle\ ,\ a \#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{app})} \langle \emptyset,\ \{X, Z\}, id, \{\ \langle [b](X * Y), Z\rangle \approx_? \langle [b](\overline{b} * (a\,b).X), (a\,b).Z\rangle\ ,\ a \#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{pair})} \langle \emptyset,\ \{X, Z\}, id, \{\ [b](X * Y) \approx_? [b](\overline{b} * (a\,b).X)\ ,\ Z \approx_? (a\,b).Z,\ a \#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{[aa]})} \langle \emptyset,\ \{X, Z\}, id, \{\ X * Y \approx_? (\overline{b} * (a\,b).X)\ ,\ Z \approx_? (a\,b).Z,\ a \#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle$$

1.

$$\Rightarrow_{(\approx_?\mathbf{C})} \langle \emptyset,\ \{X, Z\}, id, \{X \approx_? \overline{b},\ \ Y \approx_? (a\,b).X\ ,\ Z \approx_? (a\,b).Z,\ a \#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{inst})} \langle \emptyset,\ \{X, Z\}, \{Y/(a\,b).X\}, \{X \approx_? \overline{b},\ \ Z \approx_? (a\,b).Z\ ,\ a \#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{inv})} \langle \emptyset,\ \{X, Z\}, \{Y/(a\,b).X\}, \{X \approx_? \overline{b},\ (a\,b).Z \approx_? Z,\ a \#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle$$

2.

$$\Rightarrow_{(\approx_?\mathbf{C})} \langle \emptyset,\ \{X, Z\}, id, \{\ X \approx_? (a\,b).X\ ,\ Y \approx_? \overline{b},\ Z \approx_? (a\,b).Z,\ a \#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{inv})} \langle \emptyset,\ \{X, Z\}, id, \{(a\,b).X \approx_? X,\ \ Y \approx_? \overline{b}\ ,\ Z \approx_? (a\,b).Z,\ a \#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{inst})} \langle \emptyset,\ \{X, Z\}, \{Y/\overline{b}\}, \{(a\,b).X \approx_? X,\ \ Z \approx_? (a\,b).Z\ ,\ a \#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{inv})} \langle \emptyset,\ \{X, Z\}, \{Y/\overline{b}\}, \{(a\,b).X \approx_? X,\ (a\,b).Z \approx_? Z,\ \ a \#_? f\langle [a](\overline{a} * X), Z\rangle\ \}\rangle$$

$$\Rightarrow_{(\#_?\,\mathbf{app})} \langle \emptyset,\ \{X, Z\}, \{Y/\overline{b}\}, \{(a\,b).X \approx_? X,\ (a\,b).Z \approx_? Z,\ \ a \#_? \langle [a](\overline{a} * X), Z\rangle\ \}\rangle$$

$$\Rightarrow_{(\#_? \mathbf{pair})} \langle \emptyset,\ \{X,Z\},\{Y/\overline{b}\},\{(a\,b).X \approx_? X,\ (a\,b).Z \approx_? Z,\ \boxed{a \#_? [a](\overline{a} * X),\ a \#_? Z}\}\rangle$$

$$\Rightarrow_{(\#_? \mathbf{a[a]})} \langle \emptyset,\ \{X,Z\},\{Y/\overline{b}\},\{(a\,b).X \approx_? X,\ (a\,b).Z \approx_? Z,\ \boxed{a \#_? Z}\}\rangle$$

$$\Rightarrow_{(\#_? \mathbf{var})} \langle \{a\#Z\},\ \{X,Z\},\{Y/\overline{b}\},\{\boxed{(a\,b).X \approx_? X},\ (a\,b).Z \approx_? Z\}\rangle$$

$$\Rightarrow_{(\mu_{\mathbf{fp}})} \langle \{a\#X, b\#X, a\#Z\},\ \{X,Z\},\{Y/\overline{b}\},\{\boxed{(a\,b).Z \approx_? Z}\}\rangle$$

$$\Rightarrow_{(\mu_{\mathbf{fp}})} \langle \{a\#X, b\#X, a\#Z, b\#Z\},\ \{X,Z\},\{Y/\overline{b}\},\emptyset\rangle$$

Example 5.6 (Nominal C equality-checking)**.** *This example presents the execution of the nominal C-unification algorithm applied to nominal C-equality checking. In item (a), the set of protected variables, $\{X,Y,Z\}$, consists now of all variables in the input problem. The algorithm generates two leaves (see Fig. 5.16)*

$$\langle \emptyset,\ \{X,Y,Z\}, id, \{X \approx_? \overline{b},\ Y \approx_? (a\,b).X, (a\,b).Z \approx_? Z,\ a \#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle$$

$$\textit{and, } \langle \emptyset,\ \{X,Y,Z\}, id, \{(a\,b).X \approx_? X,\ Y \approx_? \overline{b},\ (a\,b).Z \approx_? Z,\ a \#_? f\langle [a](\overline{a} * X), Z\rangle\}\rangle.$$

Both are quadruples that have equations without solutions. In the former, X cannot be instantiated to solve $X \approx_? \overline{b}$, and in the latter, Y cannot be instantiated to solve $Y \approx_? \overline{b}$.

In item (b), the set of protected variables, $\{X,Y\}$, consists also of all variables in the input problem, but the generated leaves (see Fig. 5.17) are

$$\langle \emptyset,\ \{X,Y\}, id, \{X \approx_? \overline{b},\ \overline{b} \approx_? (a\,b).X, (a\,b).Y \approx_? Y,\ a \#_? f\langle [a](\overline{a} * X), Y\rangle\}\rangle$$

$$\textit{and, } \langle \{a\#X, b\#X, a\#Y, b\#Y\},\ \{X,Y\}, id, \emptyset\rangle$$

The first leaf has also equations with the protected variable X. Namely, in equations $X \approx_? \overline{b}$ and $\overline{b} \approx_? (a\,b).X$ X cannot be instantiated. Thus, neither equation has solutions. On the other branch, the second leaf provides a solution given by the freshness context $\{a\#X, b\#X, a\#Y, b\#Y\}$.

In item (c), the set of protected variables, $\{X\}$, consists also of all variables in the input problem and the generated leaves (see Fig. 5.18) are

$$\langle \{a\#X, c\#X, d\#X\}, \{X\}, id, \emptyset\rangle \textit{ and } \langle \{a\#X, b\#X, c\#X\}, \{X\}, id, \emptyset\rangle$$

Observe that these two successful leaves have different freshness contexts with the same number of elements: $\{a\#X, c\#X, d\#X\}$ and $\{a\#X, b\#X, c\#X\}$.

(a) $\langle \emptyset,\ \{X,Y,Z\}, id, \{\boxed{[a]f\langle [b](X * Y), Z\rangle \approx_? [b]f\langle [a](\overline{a} * X), Z\rangle}\}\rangle$

$\Rightarrow_{(\approx_?[\mathbf{ab}])}$ $\langle\emptyset,\ \{X,Y,Z\}, id, \{\ f\langle[b](X*Y), Z\rangle \approx_? f\langle[b](\overline{b}*(a\,b).X),(a\,b).Z\rangle\ ,\ a\,\#_?\,f\langle[a](\overline{a}*X), Z\rangle\}\rangle$

$\Rightarrow_{(\approx_?\mathbf{app})}$ $\langle\emptyset,\ \{X,Y,Z\}, id, \{\ \langle[b](X*Y), Z\rangle \approx_? \langle[b](\overline{b}*(a\,b).X),(a\,b).Z\rangle\ ,\ a\,\#_?\,f\langle[a](\overline{a}*X), Z\rangle\}\rangle$

$\Rightarrow_{(\approx_?\mathbf{pair})}$ $\langle\emptyset,\ \{X,Y,Z\}, id, \{\ [b](X*Y) \approx_? [b](\overline{b}*(a\,b).X)\ ,\ Z \approx_? (a\,b).Z,\ a\,\#_?\,f\langle[a](\overline{a}*X), Z\rangle\}\rangle$

$\Rightarrow_{(\approx_?[\mathbf{aa}])}$ $\langle\emptyset,\ \{X,Y,Z\}, id, \{\ X*Y \approx_? (\overline{b}*(a\,b).X)\ ,\ Z \approx_? (a\,b).Z,\ a\,\#_?\,f\langle[a](\overline{a}*X), Z\rangle\}\rangle$

1.

$\Rightarrow_{(\approx_?\mathbf{C})}$ $\langle\emptyset,\ \{X,Y,Z\}, id, \{X \approx_? \overline{b},\ Y \approx_? (a\,b).X,\ \ Z \approx_? (a\,b).Z\ ,\ a\,\#_?\,f\langle[a](\overline{a}*X), Z\rangle\}\rangle$

$\Rightarrow_{(\approx_?\mathbf{inv})}$ $\langle\emptyset,\ \{X,Y,Z\}, id, \{X \approx_? \overline{b},\ Y \approx_? (a\,b).X, (a\,b).Z \approx_? Z,\ a\,\#_?\,f\langle[a](\overline{a}*X), Z\rangle\}\rangle$

2.

$\Rightarrow_{(\approx_?\mathbf{C})}$ $\langle\emptyset,\ \{X,Y,Z\}, id, \{\ X \approx_? (a\,b).X\ ,\ Y \approx_? \overline{b},\ Z \approx_? (a\,b).Z,\ a\,\#_?\,f\langle[a](\overline{a}*X), Z\rangle\}\rangle$

$\Rightarrow_{(\approx_?\mathbf{inv})}$ $\langle\emptyset,\ \{X,Y,Z\}, id, \{(a\,b).X \approx_? X,\ Y \approx_? \overline{b},\ \ Z \approx_? (a\,b).Z\ ,\ a\,\#_?\,f\langle[a](\overline{a}*X), Z\rangle\}\rangle$

$\Rightarrow_{(\approx_?\mathbf{inv})}$ $\langle\emptyset,\ \{X,Y,Z\}, id, \{(a\,b).X \approx_? X,\ Y \approx_? \overline{b},\ (a\,b).Z \approx_? Z,\ a\,\#_?\,f\langle[a](\overline{a}*X), Z\rangle\}\rangle$

(b) $\langle\emptyset,\ \{X,Y\}, id, \{[a]f\langle[b](X*\overline{b}), Y\rangle \approx_? [b]f\langle[a](\overline{a}*X), Y\rangle\}\rangle$

$\Rightarrow_{(\approx_?[\mathbf{ab}])}$ $\langle\emptyset,\ \{X,Y\}, id, \{\ f\langle[b](X*\overline{b}), Y\rangle \approx_? f\langle[b](\overline{b}*(a\,b).X),(a\,b).Y\rangle\ ,\ a\,\#_?\,f\langle[a](\overline{a}*X), Y\rangle\}\rangle$

$\Rightarrow_{(\approx_?\mathbf{app})}$ $\langle\emptyset,\ \{X,Y\}, id, \{\ \langle[b](X*\overline{b}), Y\rangle \approx_? \langle[b](\overline{b}*(a\,b).X),(a\,b).Y\rangle\ ,\ a\,\#_?\,f\langle[a](\overline{a}*X), Y\rangle\}\rangle$

$\Rightarrow_{(\approx_?\mathbf{pair})}$ $\langle\emptyset,\ \{X,Y\}, id, \{[b](\ X*\overline{b}) \approx_? [b](\overline{b}*(a\,b).X)\ ,\ Y \approx_? (a\,b).Y,\ a\,\#_?\,f\langle[a](\overline{a}*X), Y\rangle\}\rangle$

$\Rightarrow_{(\approx_?[\mathbf{aa}])}$ $\langle\emptyset,\ \{X,Y\}, id, \{\ X*\overline{b} \approx_? (\overline{b}*(a\,b).X)\ ,\ Y \approx_? (a\,b).Y,\ a\,\#_?\,f\langle[a](\overline{a}*X), Y\rangle\}\rangle$

1.

$\Rightarrow_{(\approx_?\mathbf{C})}$ $\langle\emptyset,\ \{X,Y\}, id, \{X \approx_? \overline{b},\ \overline{b} \approx_? (a\,b).X,\ \ Y \approx_? (a\,b).Y\ ,\ a\,\#_?\,f\langle[a](\overline{a}*X), Y\rangle\}\rangle$

$\Rightarrow_{(\approx_?\mathbf{inv})}$ $\langle\emptyset,\ \{X,Y\}, id, \{X \approx_? \overline{b},\ \overline{b} \approx_? (a\,b).X, (a\,b).Y \approx_? Y,\ a\,\#_?\,f\langle[a](\overline{a}*X), Y\rangle\}\rangle$

2.

$\Rightarrow_{(\approx_?\mathbf{C})}$ $\langle\emptyset,\ \{X,Y\}, id, \{\ X \approx_? (a\,b).X\ ,\ \overline{b} \approx_? \overline{b},\ Y \approx_? (a\,b).Y,\ a\,\#_?\,f\langle[a](\overline{a}*X), Y\rangle\}\rangle$

$\Rightarrow_{(\approx_?\mathbf{inv})}$ $\langle\emptyset,\ \{X,Y\}, id, \{(a\,b).X \approx_? X,\ \ \overline{b} \approx_? \overline{b}\ ,\ Y \approx_? (a\,b).Y,\ a\,\#_?\,f\langle[a](\overline{a}*X), Y\rangle\}\rangle$

$\Rightarrow_{(\approx_?\mathbf{refl})}$ $\langle\emptyset,\ \{X,Y\}, id, \{(a\,b).X \approx_? X,\ \ Y \approx_? (a\,b).Y\ ,\ a\,\#_?\,f\langle[a](\overline{a}*X), Y\rangle\}\rangle$

$\Rightarrow_{(\approx_?\mathbf{inv})}$ $\langle\emptyset,\ \{X,Y\}, id, \{(a\,b).X \approx_? X,\ (a\,b).Y \approx_? Y,\ \ a\,\#_?\,f\langle[a](\overline{a}*X), Y\rangle\ \}\rangle$

$\Rightarrow_{(\#_?\,\mathbf{app})}$ $\langle\emptyset,\ \{X,Y\}, id, \{(a\,b).X \approx_? X,\ (a\,b).Y \approx_? Y,\ \ a\,\#_?\,\langle[a](\overline{a}*X), Y\rangle\ \}\rangle$

$$\Rightarrow_{(\#_? \mathbf{pair})} \langle \emptyset,\ \{X, Y\}, id, \{(a\,b).X \approx_? X,\ (a\,b).Y \approx_? Y,\ \boxed{a \#_? [a](\overline{a} * X),\ a \#_? Y}\}\rangle$$

$$\Rightarrow_{(\#_? \mathbf{a[a]})} \langle \emptyset,\ \{X, Y\}, id, \{(a\,b).X \approx_? X,\ (a\,b).Y \approx_? Y,\ \boxed{a \#_? Y}\}\rangle$$

$$\Rightarrow_{(\#_? \mathbf{var})} \langle \{a\#Y\},\ \{X, Y\}, id, \{\boxed{(a\,b).X \approx_? X},\ (a\,b).Y \approx_? Y\}\rangle$$

$$\Rightarrow_{(\mu_{\mathbf{fp}})} \langle \{a\#X, b\#X, a\#Y\},\ \{X, Y\}, id, \{\boxed{(a\,b).Y \approx_? Y}\}\rangle$$

$$\Rightarrow_{(\mu_{\mathbf{fp}})} \langle \{a\#X, b\#X, a\#Y, b\#Y\},\ \{X, Y\}, id, \emptyset\rangle$$

(c) $\langle \emptyset, \{X\}, (a\,b).X * (c\,d).X \approx_? (c\,d)(b\,d)(a\,d).X * X\rangle$

1.

$$\Rightarrow_{(\approx_? \mathbf{C})} \langle \emptyset, \{X\}, id, \boxed{(a\,b).X \approx_? (c\,d)(b\,d)(a\,d).X},\ (c\,d).X \approx_? X\rangle$$

$$\Rightarrow_{(\approx_? \mathbf{inv})} \langle \emptyset, \{X\}, id, \boxed{(a\,b)(a\,d)(b\,d)(c\,d).X \approx_? X},\ (c\,d).X \approx_? X\rangle$$

$$\Rightarrow_{(\mu_{\mathbf{fp}})} \langle \{a\#X, c\#X, d\#X\}, \{X\}, id, \boxed{(c\,d).X \approx_? X}\rangle$$

$$\Rightarrow_{(\mu_{\mathbf{fp}})} \langle \{a\#X, c\#X, d\#X\}, \{X\}, id, \emptyset\rangle$$

2.

$$\Rightarrow_{(\approx_? \mathbf{C})} \langle \emptyset, \{X\}, id, (a\,b).X \approx_? X,\ \boxed{(c\,d).X \approx_? (c\,d)(b\,d)(a\,d).X}\rangle$$

$$\Rightarrow_{(\approx_? \mathbf{inv})} \langle \emptyset, \{X\}, id, \boxed{(a\,b).X \approx_? X},\ (c\,d)(a\,d)(b\,d)(c\,d).X \approx_? X\rangle$$

$$\Rightarrow_{(\mu_{\mathbf{fp}})} \langle \{a\#X, b\#X\}, \{X\}, id, \boxed{(c\,d)(a\,d)(b\,d)(c\,d).X \approx_? X}\rangle$$

$$\Rightarrow_{(\mu_{\mathbf{fp}})} \langle \{a\#X, b\#X, c\#X\}, \{X\}, id, \emptyset\rangle$$

Remark 5.10. *Notice that, the α, C equality-checking of Ex. 5.6, item (b), differs from that of Ex. 4.8. In the former, the algorithm provides the freshness context $\{a\#X, b\#X, a\#Y, b\#Y\}$, while in the latter it returns only $\top$. This exemplifies the different behaviours between these two equality-checking algorithms. One outputs only $\top$ or $\bot$ if the two compared terms are or not $\approx_{\alpha,C}$-equivalent under a given freshness context, and the other provides either a set o freshness contexts or returns* fail *when the two compared terms are not $\approx_{\alpha,C}$-equivalent in any context.*

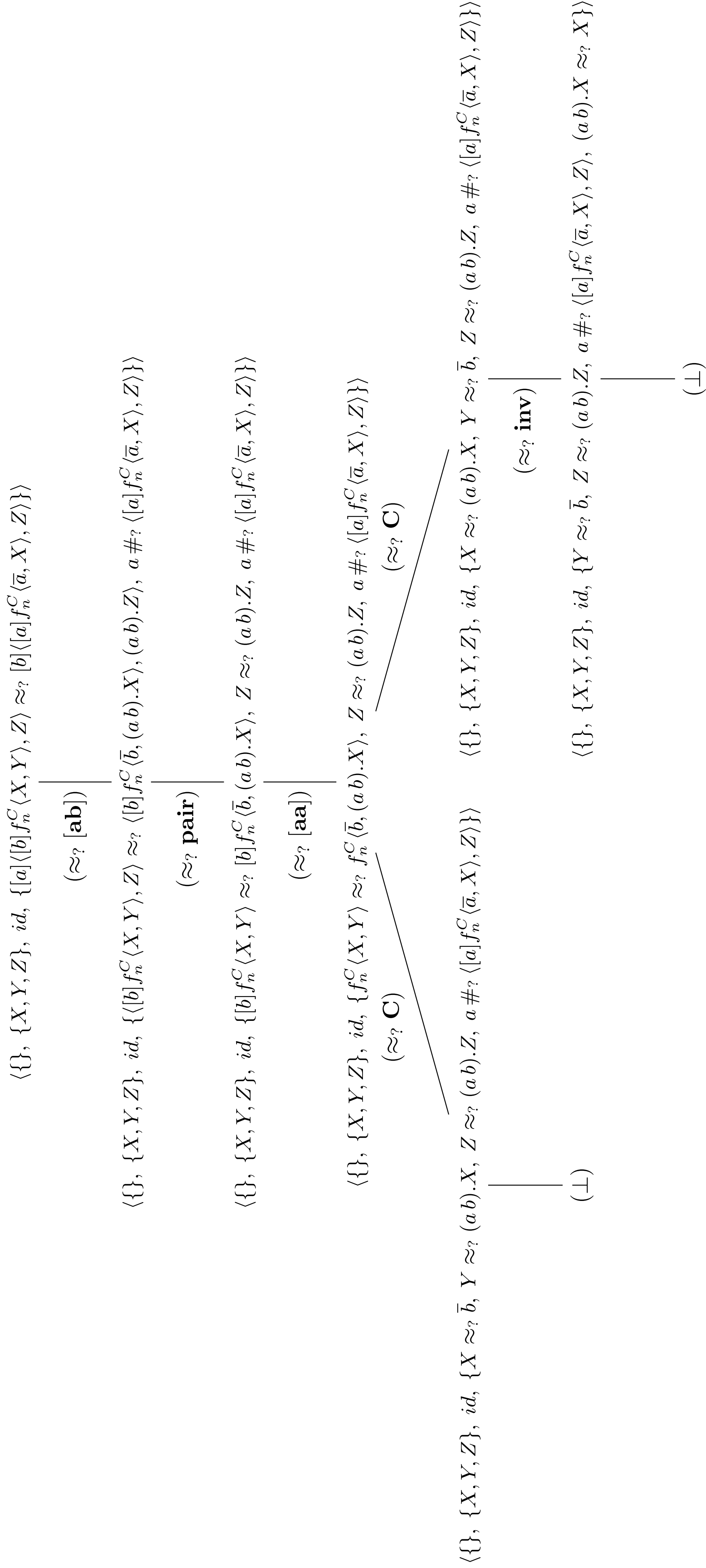

Figure 5.16: Derivation tree of Ex. 5.6 (a)

$$\langle\{\},\ \{X,Y\},\ id,\ \{[a]\langle[b]f_n^C\langle X,\bar{b}\rangle,Y\rangle \approx_? [b]\langle[a]f_n^C\langle\bar{a},X\rangle,Y\rangle\}\rangle$$

$(\approx_?\ \mathbf{[ab]})$

$$\langle\{\},\ \{X,Y\},\ id,\ \{\langle[b]f_n^C\langle X,\bar{b}\rangle,Y\rangle \approx_? \langle[b]f_n^C\langle\bar{b},(a\,b).X\rangle,(a\,b).Y\rangle,\ a\,\#_?\,\langle[a]f_n^C\langle\bar{a},X\rangle,Y\rangle\}\rangle$$

$(\approx_?\ \mathbf{pair})$

$$\langle\{\},\ \{X,Y\},\ id,\ \{[b]f_n^C\langle X,\bar{b}\rangle \approx_? [b]f_n^C\langle\bar{b},(a\,b).X\rangle,\ Y\approx_?(a\,b).Y,\ a\,\#_?\,\langle[a]f_n^C\langle\bar{a},X\rangle,Y\rangle\}\rangle$$

$(\approx_?\ \mathbf{[aa]})$

$$\langle\{\},\ \{X,Y\},\ id,\ \{f_n^C\langle X,\bar{b}\rangle \approx_? f_n^C\langle\bar{b},(a\,b).X\rangle,\ Y\approx_?(a\,b).Y,\ a\,\#_?\,\langle[a]f_n^C\langle\bar{a},X\rangle,Y\rangle\}\rangle$$

Left branch $(\approx_?\ \mathbf{C})$:

$$\langle\{\},\ \{X,Y\},\ id,\ \{X\approx_?\bar{b},\ \bar{b}\approx_?(a\,b).X,\ Y\approx_?(a\,b).Y,\ a\,\#_?\,\langle[a]f_n^C\langle\bar{a},X\rangle,Y\rangle\}\rangle$$

$(\bot)$

Right branch $(\approx_?\ \mathbf{C})$:

$$\langle\{\},\ \{X,Y\},\ id,\ \{X\approx_?(a\,b).X,\ \bar{b}\approx_?\bar{b},\ Y\approx_?(a\,b).Y,\ a\,\#_?\,\langle[a]f_n^C\langle\bar{a},X\rangle,Y\rangle\}\rangle$$

$(\approx_?\ \mathbf{inv})$

$$\langle\{\},\ \{X,Y\},\ id,\ \{\bar{b}\approx_?\bar{b},\ Y\approx_?(a\,b).Y,\ a\,\#_?\,\langle[a]f_n^C\langle\bar{a},X\rangle,Y\rangle,\ (a\,b).X\approx_? X\}\rangle$$

$(\approx_?\ \mathbf{refl})$

$$\langle\{\},\ \{X,Y\},\ id,\ \{Y\approx_?(a\,b).Y,\ a\,\#_?\,\langle[a]f_n^C\langle\bar{a},X\rangle,Y\rangle,\ (a\,b).X\approx_? X\}\rangle$$

$(\approx_?\ \mathbf{inv})$

$$\langle\{\},\ \{X,Y\},\ id,\ \{a\,\#_?\,\langle[a]f_n^C\langle\bar{a},X\rangle,Y\rangle,\ (a\,b).X\approx_? X,\ (a\,b).Y\approx_? Y\}\rangle$$

$(\#_?\ \mathbf{pair})$

$$\langle\{\},\ \{X,Y\},\ id,\ \{a\,\#_?\,[a]f_n^C\langle\bar{a},X\rangle,\ a\,\#_?\,Y,\ (a\,b).X\approx_? X,\ (a\,b).Y\approx_? Y\}\rangle$$

$(\#_?\ \mathbf{a[a]})$

$$\langle\{\},\ \{X,Y\},\ id,\ \{a\,\#_?\,Y,\ (a\,b).X\approx_? X,\ (a\,b).Y\approx_? Y\}\rangle$$

$(\#_?\ \mathbf{var})$

$$\langle\{a\#Y\},\ \{X,Y\},\ id,\ \{(a\,b).X\approx_? X,\ (a\,b).Y\approx_? Y\}\rangle$$

$(\mu_{\mathbf{fp}})$

$$\langle\{a\#X,\ b\#X,\ a\#Y\},\ \{X,Y\},\ id,\ \{(a\,b).Y\approx_? Y\}\rangle$$

$(\mu_{\mathbf{fp}})$

$$\langle\{b\#Y,\ a\#X,\ b\#X,\ a\#Y\},\ \{X,Y\},\ id,\ \{\}\rangle$$

$(\top)$

Figure 5.17: Derivation tree of Ex. 5.6 (b)

$$\langle \{\},\ \{X\},\ id,\ \{f_n^C\langle (a\,b).X, (c\,d).X\rangle \approx_? f_n^C\langle (c\,d)(b\,d)(a\,d).X, X\rangle\}\rangle$$

Left branch $(\approx_? \mathbf{C})$:

$$\langle \{\},\ \{X\},\ id,\ \{(a\,b).X \approx_? (c\,d)(b\,d)(a\,d).X,\ (c\,d).X \approx_? X\}\rangle$$

$(\approx_? \mathbf{inv})$

$$\langle \{\},\ \{X\},\ id,\ \{(c\,d).X \approx_? X,\ (a\,b)(a\,d)(b\,d)(c\,d).X \approx_? X\}\rangle$$

$(\mu_{\mathbf{fp}})$

$$\langle \{c\#X,\ d\#X\},\ \{X\},\ id,\ \{(a\,b)(a\,d)(b\,d)(c\,d).X \approx_? X\}\rangle$$

$(\mu_{\mathbf{fp}})$

$$\langle \{a\#X,\ c\#X,\ d\#X\},\ \{X\},\ id,\ \{\}\rangle$$

$(\top)$

Right branch $(\approx_? \mathbf{C})$:

$$\langle \{\},\ \{X\},\ id,\ \{(c\,d).X \approx_? (c\,d)(b\,d)(a\,d).X,\ (a\,b).X \approx_? X\}\rangle$$

$(\approx_? \mathbf{inv})$

$$\langle \{\},\ \{X\},\ id,\ \{(a\,b).X \approx_? X,\ (c\,d)(a\,d)(b\,d)(c\,d).X \approx_? X\}\rangle$$

$(\mu_{\mathbf{fp}})$

$$\langle \{a\#X,\ b\#X\},\ \{X\},\ id,\ \{(c\,d)(a\,d)(b\,d)(c\,d).X \approx_? X\}\rangle$$

$(\mu_{\mathbf{fp}})$

$$\langle \{c\#X,\ a\#X,\ b\#X\},\ \{X\},\ id,\ \{\}\rangle$$

$(\top)$

Figure 5.18: Derivation tree of Ex. 5.6 (c)

Chapter 6

Nominal fixed point problems

This chapter is devoted to the description of a generator of the possible infinite solutions of a nominal C-unification problem of the form $\langle\Delta, \mathcal{X}, \sigma, \cup_{i,j}\{\pi_{ij}.X_i \approx_? X_i\}\rangle$, of $i = 0..k$ and $j = 0..k_i$, denominated nominal C FP problems. Sec. 6.1 uses an algebraic approach to determine when is possible to build an infinite, but not complete yet, set of solutions for nominal C problems. Sec. 6.2 extends the generator of solutions, obtaining a sound and complete set for nominal C FP problems. The results of the current chapter were published in [3] and [4].

6.1 Generating combinatorial solutions via *pseudo-cycles*

To build solutions for a successful leaf $\mathcal{P} = \langle\nabla, \mathcal{X}, \sigma, P\rangle$ in the derivation tree of a given unification problem, one selects and combine solutions generated for C FP equations $\pi.X \approx_? X$, for each $X \in Var(P)$ such that $X \notin \mathcal{X}$. The notion of *pseudo-cycle of a permutation* is used in order to provide precise conditions to build terms t by combining the atoms in $dom(\pi)$, such that $\nabla \vdash \pi \cdot t \approx_{\alpha,C} t$. For convenience, the algebraic cycle representation of permutations is used. Thus, instead of presenting permutations as lists of swappings, they are presented as products of disjoint cycles [62].

Example 6.1. *The permutations*

$$\begin{pmatrix} a & \mapsto & b \\ b & \mapsto & c \\ c & \mapsto & d \\ d & \mapsto & a \\ e & \mapsto & f \\ f & \mapsto & e \end{pmatrix} \quad \textit{and} \quad \begin{pmatrix} a & \mapsto & b \\ b & \mapsto & c \\ c & \mapsto & d \\ d & \mapsto & a \end{pmatrix}$$

can be respectively represented by the lists of swappings $(a\,b)(e\,f)(c\,d)(a\,c)$ *and* $(a\,b)(c\,d)(a\,c)$*, but are also represented as the product of permutation cycles* $(a\,b\,c\,d)(e\,f)$ *and* $(a\,b\,c\,d)$*.*

Permutation cycles of length one are omitted. In general the cyclic representation of a permutation consists of the product of all its cycles.

Remark 6.1. *Let* π *be a permutation with* $|dom(\pi)| = n$*.*

- *Given* $a \in dom(\pi)$ *the elements of the sequence* $a, \pi(a), \pi^2(a), \ldots$ *cannot be all distinct;*
- *Taking the first* $k \leq n$*, such that* $\pi^k(a) = a$*, one has the* k*-cycle* $(a\ \pi(a)\ \ldots \pi^{k-1}(a))$*, where* $\pi^{j+1}(a)$ *is the* successor *of* $\pi^j(a)$*. For the 4-cycle in the permutation* $(a\,b\,c\,d)\,(e\,f)$*, the 4-cycles generated by* a, b, c *and* d *are the same:* $(a\,b\,c\,d) = (b\,c\,d\,a) = (c\,d\,a\,b) = (d\,a\,b\,c)$*.*

Def. 6.1 establishes the notion of a *pseudo-cycle w.r.t. a k-cycle* κ. Intuitively, given a k-cycle κ and a commutative function symbol $*$, a pseudo-cycle w.r.t κ, $(A_0 \ldots A_l)$, is a cycle whose elements are either atom terms built from the atoms in κ or terms of the form $A'_i * A'_j$, for A'_i, A'_j elements of a pseudo-cycle w.r.t κ.

Definition 6.1 (Pseudo-cycle)**.** *Let* $\kappa = (a_0\ a_1\ \ldots\ a_{k-1})$ *be a* k*-cycle of a permutation* π*. A* pseudo-cycle *w.r.t.* κ *is inductively defined as follows:*

1. $\overline{\kappa} = (\overline{a_0} \cdots \overline{a_{k-1}})$ *is a* pseudo-cycle *w.r.t.* κ*, called* trivial pseudo-cycle *of* κ*.*
2. $\kappa' = (A_0 \ldots A_{k'-1})$ *is a* pseudo-cycle *w.r.t.* κ*, if the following conditions are simultaneously satisfied:*
   (a) *each element of* κ' *is of the form* $B_i * B_j$*, where* $*$ *is a commutative function symbol in the signature, and* B_i, B_j *are different elements of* κ''*, a* pseudo-cycle *w.r.t.* κ*.* κ' *will be called a* first-instance pseudo-cycle *of* κ'' *w.r.t.* κ*.*
   (b) $A_i \not\approx_{\alpha,C} A_j$ *for* $i \neq j$*,* $0 \leq i, j \leq k' - 1$*;*

(c) for each $0 \leq i < k' - 1$, $\kappa \cdot A_i \approx_{\alpha,C} A_{(i+1) mod\, k'}$.

Notation 6.1. *The* length *of the pseudo-cycle* κ*, denoted by* $|\kappa|$*, consists of the number of elements in* κ*. A pseudo-cycle of length one will be called* unitary.

Example 6.2.

1. *(Continuing Ex. 5.1) The unitary pseudo-cycles of* $\kappa = (a\,b)$ *are of the form* $(\overline{a} * \overline{b})$ *for* $*$ *any commutative symbol in the signature. These pseudo-cycles are the basis for a more elaborated construction used to build infinite independent solutions for the leaf* $\langle \emptyset, id, \{X \approx_? (a\,b).X\}\rangle$*. Exs. of these solutions are:* $\langle \emptyset, \{X/\overline{a} * \overline{b}\}\rangle$, $\langle \emptyset, \{X/(\overline{a} * \overline{a}) * (\overline{b} * \overline{b})\}\rangle$, $\langle \emptyset, \{X/(\overline{a} * \overline{b}) * (\overline{a} * \overline{b})\}\rangle$, $\langle \emptyset, \{X/((\overline{a} * \overline{a}) * \overline{a}) * ((\overline{b} * \overline{b}) * \overline{b})\}\rangle$, $\langle \emptyset, \{X/(\overline{a} * (\overline{a} * \overline{a})) * (\overline{b} * (\overline{b} * \overline{b}))\}\rangle$*, etc.*

2. *(Continuing Exs. 5.3 and 6.1) In* $\mathcal{Q}_1$ *and* $\mathcal{Q}_2$ *one has the occurrences of the 4-cycle* $\kappa = (a\,b\,c\,d)$*. Suppose* $*, \oplus, +$ *are commutative operators in the signature. The following are pseudo-cycles w.r.t.* κ*:* $\overline{\kappa} = (\overline{a}\ \overline{b}\ \overline{c}\ \overline{d})$*;* $\kappa_1 = ((\overline{a}*\overline{b})\ (\overline{b}*\overline{c})\ (\overline{c}*\overline{d})\ (\overline{d}*\overline{a}))$*;* $\kappa_2 = ((\overline{a} \oplus \overline{c})\ (\overline{b} \oplus \overline{d}))$*;* $\kappa_{11} = (((\overline{a}*\overline{b}) + (\overline{b}*\overline{c}))\ ((\overline{b}*\overline{c}) + (\overline{c}*\overline{d}))((\overline{c}*\overline{d}) + (\overline{d}*\overline{a}))\ ((\overline{d}*\overline{a}) + (\overline{a}*\overline{b})))$*;* $\kappa_{12} = (((\overline{a}*\overline{b}) * (\overline{c}*\overline{d}))\ ((\overline{b}*\overline{c}) * (\overline{d}*\overline{a})))$*;* $\kappa_{21} = (((\overline{a} \oplus \overline{c}) * (\overline{b} \oplus \overline{d})))$*;* $\kappa_{121} = (((\overline{a}*\overline{b}) * (\overline{c}*\overline{d})) * ((\overline{b}*\overline{c}) * (\overline{d}*\overline{a})))$*.* κ_1 *and* κ_2 *are first-instance pseudo-cycles of* $\overline{\kappa}$*, and* κ_{11} *and* κ_{12} *of* κ_1 *and* κ_{21} *of* κ_2*. Notice that,* $|\overline{\kappa}| = |\kappa_1| = |\kappa_{11}| = 4$, $|\kappa_{12}| = 2$*, and* $|\kappa_{21}| = |\kappa_{121}| = 1$*. Also,* κ_1 *corresponds to* $((\overline{a}*\overline{d})\ (\overline{b}*\overline{a})\ (\overline{c}*\overline{b})\ (\overline{d}*\overline{c}))$*, a first-instance pseudo-cycle of* $\overline{\kappa}$*.*

   Finally, observe that for the elements of the unitary pseudo-cycles κ_{21} *and* κ_{121}*, say* $s = (\overline{a} \oplus \overline{c}) * (\overline{b} \oplus \overline{d})$ *and* $t = ((\overline{a}*\overline{b}) * (\overline{c}*\overline{d})) * ((\overline{b}*\overline{c}) * (\overline{d}*\overline{a}))$*,* $\{X/s\}$ *and* $\{X/t\}$ *(resp.* $\{Y/s\}$ *and* $\{Y/t\}$*) are solutions of the C FP equation* $(a\ b\ c\ d)(e\ f).X \approx_? X$ *(resp.* $(a\ b\ c\ d).Y \approx_? Y$*).*

Pseudo-cycles will be considered *modulo commutativity.* This notion of equivalence between pseudo-cycles is formally expressed by Def. 6.2.

Definition 6.2 (Pseudo-cycle C equivalence)**.** *Let* κ_1 *and* κ_2 *be two pseudo-cycles w.r.t. the pseudo-cycle* κ*. They are said to be* equivalent modulo commutativity of $*$*, denoted by* $\kappa_1 \approx_{(C,*)} \kappa_2$*, if each element* A *of* κ_1 *is equivalent modulo commutativity of* $*$ *to one element* B *of* κ_2*, i.e.,* $A \approx_{(C,*)} B$ *and they have the same successor, that is,* $\kappa \cdot A \approx_{(C,*)} \kappa \cdot B$*.*

Example 6.3. $((A * B)\ (C * D)\ (E * F)) = \kappa_1 \approx_{(C,*)} \kappa_2 = ((F * E)\ (A * B)\ (D * C))$.

Notation 6.2. $[\kappa]_{(C,*)}$ *denotes the equivalence class modulo commutativity of* $*$ *of the pseudo-cycle* κ*, that is,* $[\kappa]_{(C,*)} = \{\kappa' \mid \kappa \approx_C \kappa'\}$*.*

When the operator $*$ is clear from the context, it will be denoted by $\approx_C$ and the congruence class of the pseudo-cycle κ by $[\kappa]_C$. So, pseudo-cycle equivalence modulo C can be determined by a single element.

Lemma 6.1. *If κ_1 and κ_2 are two pseudo-cycles w.r.t. the cycle κ, and there exists $t \in \kappa_1$ such that $t \approx_C t'$, for some $t' \in \kappa_2$, then $\kappa_1 \approx_C \kappa_2$.*

Proof. The proof follows directly from the fact that $\kappa.t \approx_C \kappa.t'$. □

Remark 6.2. *The elements of pseudo-cycles can be represented by the coefficients of κ. Thus, if one chooses an element A_0 of κ, the pseudo-cycle $\kappa' = ((\kappa^0 \cdot A_0) \cdots (\kappa^{k-1} \cdot A_0))$ can be represented by $(\overline{0} \cdots \overline{k-1})$, the choice of A_0 only reorders the pseudo-cycle. Then, one can define an operation $+$ between elements of $(\overline{0} \cdots \overline{k-1})$ where $\overline{i} + \overline{j} = \overline{i+j}$ and $\overline{i+k} = \overline{i}$.*

Def. 6.3 is devoted to organise the first-instance pseudo-cycles of a *pseudo-cycle* κ in a matrix with dimensions $(k-1) \times k$, denominated *first-instance pseudo-cycle matrix*. Each first-instance pseudo-cycle of κ is also a pseudo-cycle that have size less or equal to $|\kappa|$. As showed in the following lemmas, a chain of subsequent first-instance pseudo-cycle matrices may give rise to a set of pseudo-cycles κ_i, where $|\kappa_i| = 1$. Such pseudo-cycles configures solutions for a given C FP problem.

Definition 6.3. *The* first-instance pseudo-cycle matrix $\mathcal{M}_{(k-1)\times k}$ of a pseudo-cycle $\kappa = (\overline{0} \cdots \overline{k-1})$ for a commutative function symbol $*$, *is defined as the $(k-1) \times k$-matrix with components $a_{ij} := \overline{j-1} * \overline{i+j-1}$.*

$$\mathcal{M}_{(k-1)\times k} = \begin{bmatrix} \overline{0} * \overline{1} & \overline{1} * \overline{2} & \cdots & \overline{k-1} * \overline{0} \\ \overline{0} * \overline{2} & \overline{1} * \overline{3} & \cdots & \overline{k-1} * \overline{1} \\ & & & \\ \ddots & \ddots & \ddots & \ddots \\ & & & \\ \overline{0} * \overline{k-2} & \overline{1} * \overline{k-1} & \cdots & \overline{k-1} * \overline{k-3} \\ \overline{0} * \overline{k-1} & \overline{1} * \overline{0} & \cdots & \overline{k-1} * \overline{k-2} \end{bmatrix}_{(k-1)\times k}$$

When the function symbol $*$ is clear from the context $\mathcal{M}_{(k-1)\times k}$ will be called simply first-instance pseudo-cycle matrix *of κ.*

Remark 6.3. *One wants to establish a relationship between the rows of the matrix $\mathcal{M}_{(k-1)\times k}$ related to κ and the first-instance pseudo-cycles of κ: for each i, $1 \leq i \leq k-1$, the i-th row $[a_{i1}\ a_{i2} \ldots a_{ik}]$ of $\mathcal{M}_{(k-1)\times k}$ is mapped to the first-instance pseudo-cycle: $\kappa_i = (\overline{0} * \overline{i} \quad \overline{1} * \overline{i+1} \quad \ldots \ \overline{k-1} * \overline{i-1})$.*

Example 6.4. *The first-instance pseudo-cycle matrix of* $\kappa = (\overline{0}\,\overline{1}\,\overline{2}\,\overline{3})$ *is*

$$\mathcal{M}_{3\times 4} = \begin{bmatrix} \overline{0}*\overline{1} & \overline{1}*\overline{2} & \overline{2}*\overline{3} & \overline{3}*\overline{0} \\ \overline{0}*\overline{2} & \overline{1}*\overline{3} & \overline{2}*\overline{0} & \overline{3}*\overline{1} \\ \overline{0}*\overline{3} & \overline{1}*\overline{0} & \overline{2}*\overline{1} & \overline{3}*\overline{2} \end{bmatrix}$$

Not all the rows of $\mathcal{M}_{3\times 4}$ *are pseudo-cycles of* κ*: the second row does not, since it contradicts condition (2.b) of the Def. 6.1; however, it contains two first-instance pseudo-cycles of* κ*, both with length 2. Also notice that the first and third rows are equivalent modulo* C*.*

Example 6.5. *For the* 4*-cycle* $\kappa = (a\ b\ c\ d)$*, if one considers its pseudo-cycle* $\overline{\kappa} = (\overline{a}\ \overline{b}\ \overline{c}\ \overline{d})$*, the representation* $\kappa = (\overline{0}\quad \overline{1}\quad \overline{2}\quad \overline{3})$ *of* κ *via coefficients, does not depend on the choice of* A_0*.*

- *if* $A_0 = \overline{a}$*, then* $(\overline{0}\ \overline{1}\ \overline{2}\ \overline{3})$ *corresponds to* $((\kappa^0 \cdot \overline{a})\quad (\kappa^1 \cdot \overline{a})\quad (\kappa^2 \cdot \overline{a})\quad (\kappa^3 \cdot \overline{a})) = (\overline{a}\ \overline{b}\ \overline{c}\ \overline{d})$*.*
- *if* $A_0 = \overline{b}$*, then* $(\overline{0}\ \overline{1}\ \overline{2}\ \overline{3})$ *corresponds to* $((\kappa^0 \cdot \overline{b})\quad (\kappa^1 \cdot \overline{b})\quad (\kappa^2 \cdot \overline{b})\quad (\kappa^3 \cdot \overline{b})) = (\overline{b}\ \overline{c}\ \overline{d}\ \overline{a})$*.*

and so on. If one chooses the pseudo-cycle $k_1 = ((a*c)\ (b*d))$ *of* κ*, one can still represent it via coefficients* $(\overline{0}\ \overline{1})$*.*

- *if* $A_0 = (a*c)$ *then* $(\overline{0}\ \overline{1})$ *corresponds to* κ_1*, itself.*
- *if* $A_0 = (b*d)$ *then* $(\overline{0}\ \overline{1})$ *corresponds to* $((\kappa_1^0 \cdot (b*d))\quad (\kappa_1^1 \cdot (b*d))) = ((b*d)\ (a*c)))$*.*

Lemma 6.2. *Let* $\mathcal{M} = (a_{ij})_{k-1\times k}$ *be a first-instance pseudo-cycle matrix for a pseudo-cycle* κ*. The following properties are valid in* $\mathcal{M}$*:*

1. $a_{i(j+1)} = \kappa \cdot a_{ij}$*, for* $j < k$ *.*
2. $\kappa \cdot a_{ik} = a_{i1}$*;*
3. *The element* a_{ij} *is equivalent modulo commutativity of* $*$ *to the element* $a_{(k-i)(i+j)}$*, i.e.,* $a_{ij} \approx_C a_{(k-i)(i+j)}$*, for* $1 \leq i \leq \lfloor \frac{k-1}{2} \rfloor$*.*
4. *Suppose* $k = 2n$ *for some positive integer* n*.*
   (a) $a_{ni} \approx_C a_{n(n+i)}$*, for* $1 \leq i \leq k$*.*
   (b) *If* $\kappa_{n_1} = (a_{n1}\ a_{n2} \ldots\ a_{nn})$ *and* $\kappa_{n_2} = (a_{n(n+1)}\ a_{n(n+2)} \ldots\ a_{nk})$ *then* $\kappa_{n_1} \approx_C \kappa_{n_2}$*.*

   That is, when k *is even, the* $\frac{k}{2}$*-th row of the matrix , has two equivalent modulo* C *pseudo-cycles with relation to* κ*, both with length* $\frac{k}{2}$*.*

Proof. The proof of all items follows by algebraic manipulation, using the commutativity of $*$ and Def. 6.3:

1. $$\begin{aligned} a_{i(j+1)} &= (\overline{j+1-1}) * (\overline{i+j+1-1}) = \overline{(j-1)+1} * \overline{(i+j-1)+1} \\ &= (\kappa \cdot \overline{j-1}) * (\kappa \cdot \overline{i+j-1}) = \kappa \cdot a_{ij} \end{aligned}$$

2. $$\begin{aligned} \kappa \cdot a_{ik} &= (\kappa \cdot \overline{k-1}) * (\kappa \cdot \overline{i+k-1}) = \overline{k-1+1} * \overline{i+k-1+1} \\ &= \overline{(1-1)+k} * \overline{(i+1-1)+k} = \overline{1-1} * \overline{i+1-1} = a_{i1} \end{aligned}$$

3. $$\begin{aligned} a_{(k-i)(i+j)} &= \overline{i+j-1} * \overline{k-i+i+j-1} \approx_C \overline{k-i+i+j-1} * \overline{i+j-1} \\ &= \overline{(j-1)+k} * \overline{i+j-1} = \overline{j-1} * \overline{i+j-1} = a_{ij} \end{aligned}$$

4. (a) By Def. 6.3, it follows that
   $a_{n(n+i)} = \overline{n+i-1} * \overline{n+n+i-1} = \overline{n+i-1} * \overline{(i-1)+k} \approx_C \overline{i-1} * \overline{n+i-1} = a_{ni}$

   (b) The proof follows directly from Def. 6.2 and the mapping from the rows of $M_{(k-1)\times k}$ to cycles.

$\square$

Example 6.6. *Let $\rho = (a\ b\ c\ d\ e)$ be a 5-cycle, its pseudo-cycle representation via coefficients is $\kappa = (\overline{0}\,\overline{1}\,\overline{2}\,\overline{3}\,\overline{4})$, and the corresponding first-instance pseudo-cycle matrix is*

$$\mathcal{M}_{3\times 4} = \begin{bmatrix} \overline{0}*\overline{1} & \overline{1}*\overline{2} & \overline{2}*\overline{3} & \overline{3}*\overline{4} & \overline{4}*\overline{0} \\ \overline{0}*\overline{2} & \overline{1}*\overline{3} & \overline{2}*\overline{4} & \overline{3}*\overline{0} & \overline{4}*\overline{1} \\ \overline{0}*\overline{3} & \overline{1}*\overline{4} & \overline{2}*\overline{0} & \overline{3}*\overline{1} & \overline{4}*\overline{2} \\ \overline{0}*\overline{4} & \overline{1}*\overline{0} & \overline{2}*\overline{1} & \overline{3}*\overline{2} & \overline{4}*\overline{3} \end{bmatrix}$$

Notice that, for instance, $a_{12} \approx_C a_{43}$ which implies that $\kappa_1 \approx_C \kappa_4$. Similarly, $a_{23} \approx_C a_{35}$ implies $\kappa_2 \approx_C \kappa_3$. Every row of $\mathcal{M}_{3\times 4}$ is a first-instance pseudo-cycle of κ, with length 5.

Remark 6.4. *The next results establish the properties and conditions that a row of $\mathcal{M}_{(k-1)\times k}$ must satisfy to be (or contain) a pseudo-cycle of κ.*

Lemma 6.3. *Let $\mathcal{M} = (a_{ij})_{k-1\times k}$ be a first-instance pseudo-cycle matrix for a pseudo-cycle κ . If there exists a positive integer $n \leq k$ such that $a_{ni} \approx_C a_{ni'}$, for some $i \neq i'$, then $k = 2n$.*

Proof. Suppose that $a_{ni} \approx_C a_{ni'}$ with $i \neq i'$. Then, $\overline{i-1} * \overline{i+n-1} \approx_C \overline{i'-1} * \overline{i'+n-1}$. Since $i \neq i'$, it follows that $\overline{i-1} = \overline{i'+n-1}$ and $\overline{i'-1} = \overline{i+n-1}$. Therefore, $\overline{i} = \overline{i'+n}$ and $\overline{i'} = \overline{i+n}$, which imply, $\overline{i+i'} = \overline{(i+i')+2n}$. Hence, $\overline{0} = \overline{2n}$ and $k = 2n$. $\square$

Theorem 6.1. *Let κ be a pseudo-cycle with k elements and $\mathcal{M}$ its first-instance matrix, with k an odd number. The $k-1$ rows of $\mathcal{M}$ are first-instance pseudo-cycles w.r.t. κ, with k elements.*

Proof. For each row $r_i = [a_{i1} \quad a_{i2} \quad \dots \quad a_{ik}]$ of $\mathcal{M}$, for $1 \leq i \leq k$, one uses the mapping from Rmk. 6.3, to obtain the candidate pseudo-cycle of κ :

$$\kappa_i = (\overline{0} * \overline{i} \quad \overline{1} * \overline{i+1} \quad \dots \ \overline{k-1} * \overline{i-1}),$$

whose elements clearly satisfy the conditions **b.** and **c.** of Def. 6.1. Also, since k is odd, it follows from Lem. 6.2, that the elements of κ_i satisfy condition **a.** of the Def. 6.1. □

Lemma 6.4. *Let $\mathcal{M}_{(k-1)\times k}$ be first-instance matrix of the pseudo-cycle κ, with $k = 2n+1$ for some positive integer n. If κ_i is the pseudo-cycle in the i-th row of $\mathcal{M}$, for $1 \leq i \leq k-1$, then $\kappa_i \approx_C \kappa_{k-i}$.*

Proof. By Lem. 6.2, $a_{(k-i)(i+j)} \approx_C a_{ij}$, for each $1 \leq i \leq k-1$, therefore, by Lem. 6.1, $\kappa_i \approx_C \kappa_{k-i}$. □

The next theorem says that the first-instance pseudo-cycle matrix of κ contains all possible first-instance pseudo-cycles of κ.

Theorem 6.2. *κ' is a first-instance pseudo-cycle of κ iff it is equivalent to a pseudo-cycle that is in a row of the first-instance pseudo-cycle matrix $\mathcal{M}_{(k-1)\times k}$ of κ.*

Proof. Let $A_0 \in \kappa'$, by definition, $A_0 = B_1 * B_2$ for some $B_1, B_2 \in \kappa$. If $B_1 \neq B_2$ then $A_0 = \overline{m} * \overline{n}$ with $m \neq n$ and $0 \leq m, n \leq k-1$. But for all m, n there exist i, j such that $\overline{m} * \overline{n} \approx_C a_{ij} \in M_{(k-1)\times k}$. On one hand, if k is odd and $\kappa'' = (A_0 \quad \kappa A_0 \quad \dots \quad \kappa^{(k-1)} A_0)$ then, by Thm. 6.1, κ'' is a pseudo-cycle in a row of $\mathcal{M}_{(k-1)\times k}$, with k elements. Besides, by Lem. 6.1, $\kappa' \approx_C \kappa''$. On the other hand, if k is even and $\kappa'' = (A_0 \quad \kappa A_0 \quad \dots \quad \kappa^{(k-1)} A_0)$, by Lem. 6.2, either κ'' is equivalent to a row i for $1 \leq i \leq \lfloor \frac{k-1}{2} \rfloor$, and therefore, κ'' is equivalent to a first-instance pseudo-cycle of κ with k elements, or κ'' can be split into two pseudo-cycles κ''_1 and κ''_2 of length $\frac{k}{2}$, both containing an element equivalent to A_0, by Lem. 6.1, $\kappa' \approx_C \kappa''_i$ $(i = 1, 2)$, and therefore, κ' is in a row of $M_{(k-1)\times k}$. □

Theorem 6.3. *Let κ be a pseudo-cycle with k elements and $\mathcal{M}$ be its first-instance pseudo-cycle matrix. The following properties hold*

i) if k is even, then κ has exactly $\lfloor \frac{k-1}{2} \rfloor$ first-instance pseudo-cycles with k elements, and one with $\frac{k}{2}$ elements.

ii) if k is odd, then κ has exactly $\frac{k-1}{2}$ first-instance pseudo-cycles with k elements.

Proof.

i) Suppose $k = 2n$ for some positive integer n. By Lem. 6.2, it follows that rows 1 to $\lfloor \frac{k-1}{2} \rfloor$ of $\mathcal{M}$ are first-instance pseudo-cycles of κ with k elements and $\kappa_i \approx_C \kappa_{k-i}$, for $1 \leq i \leq \lfloor \frac{2n-1}{2} \rfloor$. According to Lems. 6.2 and 6.3, the n-th row of $\mathcal{M}$ contains two equivalent first-instance pseudo-cycles of κ both with $\frac{k}{2}$ elements.

ii) Suppose $k = 2n + 1$, for some non-negative integer n. By Thm. 6.1 the $k - 1$ rows of $\mathcal{M}$ are first-instance pseudo-cycles of κ with length k. From Lem. 6.2, it follows that $\kappa_i \approx_C \kappa_{k-i}$, for $1 \leq i \leq \lfloor \frac{k-1}{2} \rfloor$ and the result follows.

□

Remark 6.5. *Let κ be a pseudo-cycle. Notice that only item 2 of Def. 6.1 may build a first-instance pseudo-cycle κ' w.r.t. κ with fewer elements. If $|\kappa'| < |\kappa|$ then, due to algebraic properties of cycles and commutativity of the operator applied ($*$), one must have that $|\kappa'| = |\kappa|/2$. Thus, unitary pseudo-cycles can only be generated from cycles of length a power of two. This is the intuition behind the next theorem, proved by induction on the size of the cycle κ.*

Theorem 6.4. *A pseudo-cycle κ contains a unitary pseudo-cycle iff $|\kappa|$ is a power of two.*

Proof. Let κ be of the form $\kappa = (a_0 \quad a_1 \quad \dots \quad a_{k-1})$.
($\Leftarrow$) The proof is by induction on n.

- **Base Case.** $n = 1$

  In this case, $k = 2$ and the first-instance pseudo-cycle matrix w.r.t. κ and $*$ is

  $$\mathcal{M}_{12\times} = \left[\begin{array}{cc} \overline{0} * \overline{1} & \overline{1} * \overline{0} \end{array} \right]$$

  Notice that $\overline{0} * \overline{1} \approx_C \overline{1} * \overline{0}$, and this single row of $\mathcal{M}_{12\times}$ contains two equivalent first-instance unitary pseudo-cycles $\kappa_1 = (\overline{0} * \overline{1})$ and $\kappa_2 = (\overline{1} * \overline{0})$.

- **Induction Step.** Suppose that the result holds for $k = 2^n$. It is proved the result holds for $k = 2^{n+1}$.

  Let κ_l be a pseudo-cycle of length $l = 2^{n+1} = 2.(2^n)$. By Thm. 6.3, κ_l has a first instance pseudo-cycle of length $\frac{l}{2} = 2^n$, and by IH, the result follows.

($\Rightarrow$) Let κ_1 and κ_2 be pseudo-cycles w.r.t. a pseudo-cycle κ such that κ_2 is a first-instance pseudo-cycle of κ_1. By Lem. 6.2 , $|\kappa_2| < |\kappa_1|$ only if $|\kappa_1| = 2.|\kappa_2|$.

Notice that $\overline{\kappa} = (\overline{a_0} \cdots \overline{a_{k-1}})$ is an immediate first-instance pseudo-cycle of κ with k elements.

- If $k = 1 = 2^0$, the result follows.
- Suppose that $k > 1$. Then, there exists a pseudo-cycle κ_p regarding to κ, with $|\kappa_p| = 1$ only if one has a chain of pseudo-cycles $\kappa, \kappa_1, \cdots, \kappa_{p-1}, \kappa_p$, where κ_1 is a first-instance pseudo-cycle of κ, and κ_{i+1} is a first-instance pseudo-cycle of κ_i, for all $i = 1, \cdots, (p-1)$. Besides,

  $$|\kappa| = 2.|\kappa_1|,\ |\kappa_1| = 2.|\kappa_2|,\ \cdots,\ |\kappa_{p-1}| = 2.|\kappa_p| \text{ and } |\kappa_p| = 1.$$

  So, k must be equal to 2^p, and the result follows.

□

Notice that, according to item 2.c of Def. 6.1, if $\kappa' = (A_0 \ldots A_{k'-1})$ is a pseudo-cycle w.r.t. π then $\pi \cdot A_{k'-1} \approx_{\alpha,C} A_0$; particularly, if $k' = 1$ then $\pi \cdot A_0 \approx_{\alpha,C} A_0$. Below, given $\mathcal{P} = \langle \emptyset, \mathcal{X}, \{\pi.X \approx_? X\}\rangle$ a C FP problem, if $X \notin \mathcal{X}$ by a *combinatorial solution* of $\mathcal{P}$ one understands a substitution $\{X/t\}$, such that $\pi \cdot t \approx_C t$, and t contains only atoms from π and commutative function symbols, built as unary pseudo-cycles w.r.t. κ a cycle in π.

Theorem 6.5. *Let $\mathcal{P} = \langle \emptyset, \mathcal{X}, \{\pi.X \approx_? X\}\rangle$ be a C FP problem, where $X \notin \mathcal{X}$. $\mathcal{P}$ has a combinatorial solution iff there exists a unitary pseudo-cycle κ w.r.t. π.*

Proof. ($\Leftarrow$) Suppose that π has a unitary pseudo-cycle, say $\kappa = (\ t\ )$, then $\kappa \cdot t \approx_C t$, by definition of pseudo-cycles, and $\pi \cdot t \approx_C t$. Therefore, $\{X/t\}$ is a solution for $\mathcal{P}$.
($\Rightarrow$) Suppose that π does not have a unitary pseudo-cycle, then by Thm. 6.4, every pseudo-cycle κ w.r.t. π has length $k = 2^n(2r+1)$, for some positive integer r.

Suppose, by contradiction, that there is a combinatorial solution for $\mathcal{P}$, say $\{X/t\}$. If any atom from $dom(\kappa)$ is in t, then all atoms of κ have to occur in t, otherwise one would not have $\pi \cdot t \approx_C t$. Since one works modulo *commutativity*, terms may change their positions pairwise, inside t, respecting the parentheses. Therefore, one should be able to arrange in pairs the atoms in κ, and permute them around the commutative symbols. To do so, one has to take the k elements and organise them in pairs, interactively. But if k has an odd factor, different from 1, it is not possible. □

Remark 6.6. *Since one can generate infinitely many unitary pseudo-cycles from a given 2^n-cycle κ in π, $n \in \mathbb{N}$, if $X \notin \mathcal{X}$ there exist infinite independent solutions for the C FP problem $\langle \emptyset, \mathcal{X}, \{\pi.X \approx_? X\}\rangle$.*

Remark 6.7. *Since the set of solutions of a nominal C-unification problem $\mathcal{P} = \langle \nabla, \mathcal{X}, P\rangle$ is the union of the solutions of the successful leaves of the derivation tree $\mathcal{T}_{\langle \nabla, \mathcal{X}, P\rangle}$ (Thm. 5.2). And this successful leaves are C FP problems (Lem. 5.5). One concludes that the nominal C-unification problems may have infinite sets of independent solutions.*

Remark 6.8. *Proving that nominal C FP problems have potentially infinite independent solutions does not provide the warranty that these problems are in the infinitary class. In fact, to show that a problem has type ∞, it is necessary to prove the existence of infinite minimal complete sets of solutions. At this point, nominal C FP was proved* at least infinitary*, which means that these kind of problems have either type ∞ or type* 0*.*

6.2 General solutions for C FP problems.

Pseudo-cycles are built just from atom terms in $dom(\pi)$ and commutative function symbols. In this section, an extension of the Def. 6.1 provides a broader class of solutions involving all function symbols and constructors of the signature.

Remark 6.9 (More general solution and complete set of solutions)**.** *The notion of* more general than *(denoted by $\preceq$) and* complete set of solutions *given by Def. 2.26 are extended to be used in the context of the $\approx_{\alpha,C}$ relation. The only change that must be done in Def. 2.26 is to replace $\mathcal{U}(\mathcal{P})$ by $\mathcal{U}_C(\mathcal{P})$.*

Example 6.7. *Given the nominal C-unification problem*

$$\mathcal{P} = \langle \emptyset, \emptyset, id, \{[a']\langle (a\,c).X \star (a\,c)(b\,c).Y,\ (a\,d)(b\,d)(c\,d).X\rangle \approx_? [b']\langle X \star Y,\ X\rangle\}\rangle$$

the algorithm in Sec. 5.1 transforms it into the following C FP problems (see Fig. 6.1).

$$\mathcal{Q}_1 = \langle \{a'\#X, a'\#Y\}, \emptyset, id, \{\pi_1.X \approx_? X,\ \pi_2.Y \approx_? Y,\ \pi_3.X \approx_? X\}\rangle$$

and

$$\mathcal{Q}_2 = \langle \{a'\#Y, b'\#Y\}, \emptyset, \{X/\pi_1^{-1}.Y\}, \{\pi_4.Y \approx_? Y,\ \pi_5.Y \approx_? Y\}\rangle.$$

Where:

- $\star$ *is a commutative symbol.*
- $\pi_1 = (a\,c)(a'\,b')$
- $\pi_2 = (a\,c)(b\,c)(a'\,b')$
- $\pi_3 = (a\,d)(b\,d)(c\,d)(a'\,b')$
- $\pi_4 = (a\,c)(b\,c)(a'\,b')(a\,c)(a'\,b')$
- $\pi_5 = (a'\,b')(a\,c)(a\,d)(b\,d)(c\,d)(a'\,b')(a\,c)(a'\,b')$

In the k-cycle permutation representation π_1, π_2, π_3, π_4 *and* π_5 *are, respectively given by:* $(a\,c)(a'\,b')$, $(a\,b\,c)(a'\,b')$, $(a\,b\,c\,d)(a'\,b')$, $(a\,b)$ *and* $(a\,d\,c\,b)(a'\,b')$. *Then, in this representation:*

$$\mathcal{Q}_1 = \langle \{a'\#X, a'\#Y\}, \emptyset, id, \{(a\,c)(a'\,b').X \approx_? X,\ (a\,b\,c)(a'\,b').Y \approx_? Y,\ (a\,b\,c\,d)(a'\,b').X \approx_? X\}\rangle$$

and

$$\mathcal{Q}_2 = \langle \{a'\#Y, b'\#Y\}, \emptyset, \{X/(a'\,b')(a\,c).Y\}, \{(a\,b).Y \approx_? Y,\ (a\,d\,c\,b)(a'\,b').Y \approx_? Y\}\rangle.$$

Def. 6.4 extends Def. 6.1 adding new cases in the construction of the *cycles*. *Extended pseudo-cycles*, or just `epc`'s, considers all nominal syntactic elements including new variables, and also non commutative function symbols. As in the previous section, the unitary extended pseudo-cycles give rise to solutions for a C FP problem, but now the generated set is proved complete. The definition of extended pseudo-cycle given below is parametric on a set $\mathcal{Z}$ of variables.

Definition 6.4 (Extended Pseudo-cycle)**.** *Let* $\pi.X \approx_? X$ *and* $\mathcal{Z}$ *a set of variables. The* extended pseudo-cycles *(for short,* `epc`*)* κ *for* π *relative to* $\mathcal{Z}$ *are inductively defined from the permutation cycles of* π *as follows:*

1. $\kappa = (Y)$, *for any variable not occurring in* $\mathcal{Z}$, *is an* `epc` *for* π;
2. $\kappa = (\overline{a_0} \cdots \overline{a_{k-1}})$ *is an* `epc` *for* $(a_0\ \cdots a_{k-1})$ *a permutation cycle in* π *such that* $k = 2^l$, *for* $l > 0$, *called* trivial extended pseudo-cycle *of* π.
3. $\kappa = (A_0 \ldots A_{k-1})$, *for a length* $k \geq 1$, *is an* `epc` *for* π, *if the following conditions are simultaneously satisfied:*
   (a) i. *each element of* κ *is of the form* $B_i \star B_j$, *where* $\star$ *is a commutative function symbol in the signature, and* B_i, B_j *are different elements of* κ', *an* `epc` *for* π; *in this case,* κ *will be called a* first-instance extended pseudo-cycle *of* κ' *for* π; *or*
   ii. *each element of* κ *is of the form* $B_i \star C_j$ *for any commutative symbol* $\star$, *where* B_i *and* C_j *are elements of* κ' *and* κ'' `epc`*'s for* π, *which might both be the same, but* κ *is not a first-instance* `epc` *for* π; *or*
   iii. *each element of* κ *is of the form* $\langle B_i, C_j\rangle$, *where* B_i *and* C_j *are elements of* κ' *and* κ'' `epc`*'s for* π, *which might both be the same; or*
   iv. *either each element of* κ *is of the form* $g\,B_i$ *or each element is of the form* $[e]\,B_i$, *where* g *is a non commutative function symbol in the signature and* $e \notin dom(\pi)$, *and each* B_i *is an element of* κ' *an* `epc` *for* π; *or*

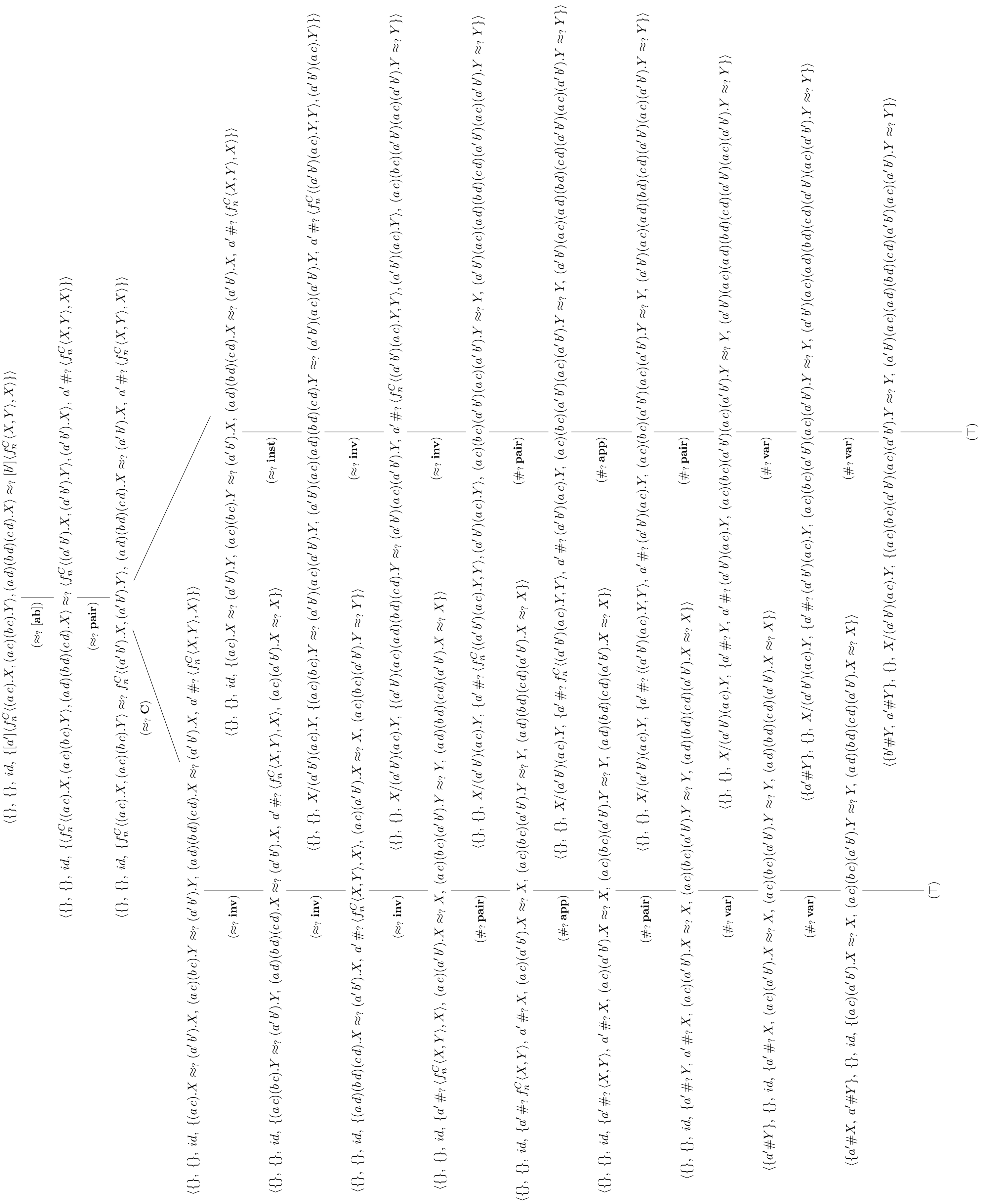

Figure 6.1: Derivation tree of Ex. 6.7

v. each element of κ is of the form $[a_j]B_i$, where a_j are atoms in $\kappa' = (\overline{a_0} \cdots \overline{a_{k'-1}})$ a trivial epc *for π, and B_i elements of κ'' an* epc *for π; and*

(b) for $\nabla' = \cup_{Y \in Var(\kappa)}\{dom(\pi)\#Y\}$,

i. it does not hold that $\nabla' \vdash A_i \approx_{\alpha,C} A_j$ for $i \neq j$, $0 \leq i,j \leq k-1$; and

ii. for each $0 \leq i \leq k-1$ one has that $\nabla' \vdash \pi(A_i) \approx_{\alpha,C} A_{(i+1)\,mod\,k}$.

Remark 6.10. *Notice that* epc*'s built using only items 2 and 3.a.i and 3.b are exactly those pseudo-cycles that are built by Def. 6.1. A relevant aspect is that, as in Def. 6.1, item 2.a, only case 3.a.i of Def. 6.4 allows generating* epc*'s that might be shorter than the* epc*'s to which this case is applied. When this is the case, the length of the generated* epc *is half of the original one. Extended pseudo-cycles of length $k = 1$ are called* unitary.

Example 6.8. *(Continuing Ex. 6.7) Given the C FP equation $(a\,b\,c\,d)(a'\,b').X \approx_? X$. Let $\kappa = (a\,b\,c\,d)$ and $\mathcal{Z} = \{X, Y\}$. Assume, $\star$ and $\oplus$ are commutative symbols, f and g non commutative symbols. The following are pseudo-cycles relative to $\mathcal{Z}$: $(\overline{a}\star\overline{d}\ \ \overline{b}\star\overline{a}\ \ \overline{c}\star\overline{b}\ \ \overline{d}\star\overline{c})$, $(\overline{a}\star\overline{c}\oplus\overline{b}\star\overline{d})$, etc. The following are* epc*'s relative to $\mathcal{Z}$: $(f\langle\overline{a},\overline{b}\rangle\ \ f\langle\overline{b},\overline{c}\rangle\ \ f\langle\overline{c},\overline{d}\rangle\ \ f\langle\overline{d},\overline{a}\rangle)$, $([e]\overline{a}\star\overline{c}\ \ [e]\overline{b}\star\overline{d})$, $(g\langle f\overline{a},[e]\overline{a}\rangle\ \ g\langle f\overline{b},[e]\overline{b}\rangle\ \ g\langle f\overline{c},[e]\overline{c}\rangle\ \ g\langle f\overline{d},[e]\overline{d}\rangle)$, $(\langle t, f\langle g\langle f\overline{a},[e]\overline{b}\rangle, Z\rangle \oplus f\langle g\langle f\overline{c},[e]\overline{d}\rangle, Z\rangle\rangle \star \langle t, f\langle g\langle f\overline{b},[e]\overline{c}\rangle, Z\rangle \oplus f\langle g\langle f\overline{d},[e]\overline{a}\rangle, Z\rangle\rangle\rangle)$, etc.*

Example 6.9 (Continuing Exs. 6.7 and 6.8)**.** *For the C FP $(a\,b\,c)(a'\,b').Y \approx_? Y$ it is not possible to obtain an* epc *w.r.t. $\kappa' = (a\,b\,c)$, due the fact that the length of this cycle is not a power of two. On the other hand, applying case 3.a.i to the trivial* epc *$(\overline{a}\ \overline{b}\ \overline{c}\ \overline{d})$ w.r.t. $\kappa = (a\,b\,c\,d)$, one obtains the* epc *$(\overline{a}\star\overline{c}\ \ \overline{b}\star\overline{d})$ that has half length of the trivial one.*

Since unitary epc's can only be obtained from permutation cycles of length a power of two, when a unitary epc is being generated, the last application of 3.a.i transforms a length two epc of the form $(A_0\ \ A_1)$ into $(A_0 \star A_1)$. By condition 3.b.ii, $\nabla \vdash \pi(A_0) \approx_{\alpha,C} A_1$ and $\nabla \vdash \pi(A_1) \approx_{\alpha,C} A_0$. Therefore, $\nabla \vdash \pi(A_0 \star A_1) \approx_{\alpha,C} A_0 \star A_1$.

Although the relation $\approx_{\alpha,C}$ is being considered by the class of terms involved in the generation of epc's, only $\approx_C$ would be necessary, except for considerations related with the freshness constraints (on new variables); hence, the invariant 3.b.ii can be seen as $\pi(A_i) \approx_C A_{i+1}$, where $i+1$ is read modulo the length of the epc.

6.2.1 Soundness and completeness of the generator

Definition 6.5 (Generated solutions of singleton C FP problems)**.** *For $\mathcal{Q}$ and $\pi.X \approx_? X \in Q$, the set of* generated solutions *for $\langle \Delta, \mathcal{X}, \{\pi.X \approx_? X\}\rangle$, denoted as $\langle \Delta, \mathcal{X}, \{\pi.X \approx_? X\}\rangle_{Sol_G}$, consists of pairs of the form $\langle \nabla, \{X/s\}\rangle$ where (s) is a unitary* epc *for π such that $\nabla \vdash dom(\Delta|_X)\,\#\,s$, where $\nabla = \Delta \cup_{Y \in Var(s)} (dom(\Delta|_X)\#Y \cup dom(\pi)\#Y)$.*

Example 6.10 (Continuing Exs. 6.7 and 6.8)**.**
The set $\langle\{a'\#X, a'\#Y\}, \emptyset, \{(a\,b\,c\,d)(a'\,b').X \approx_? X\}\rangle_{Sol_G}$ *contains only generated solutions* $\langle\nabla, \{X/\mathtt{e}\}\rangle$ *where* $(\mathtt{e})$ *is a* `epc` *w.r.t.* $(a\,b\,c\,d)$*. Because* $a'\#X$ *is in the freshness context, the condition* $\nabla \vdash dom(\Delta|_X)\,\#\,s$ *imposes that pairs* $\langle\Delta, \{X/s\}\rangle$*, being* (s) *an* `epc` *regarding* $(a'\,b')$*, are not in the set of generated solutions.*

Theorem 6.6 (Soundness of solutions of singleton C FP problems)**.**
Each $\langle\nabla, \{X/s\}\rangle$ *in* $\langle\Delta, \mathcal{X}, \{\pi.X \approx_? X\}\rangle_{Sol_G}$ *is a solution of* $\langle\Delta, \mathcal{X}, \{\pi.X \approx_? X\}\rangle$*.*

Proof. If $X \in \mathcal{X}$ the proof is trivial, because

$$\langle\Delta, \mathcal{X}, \{\pi.X \approx_? X\}\rangle_{Sol_G} = \{\langle\Delta \cup dom(\pi)\#X, id\rangle\}$$

with $id = X/X$. Supposing $X \notin \mathcal{X}$, the proof follows the lines of reasoning used for non unitary `epc`'s. By construction, the invariant that the elements of an `epc` of length l, $\kappa' = (e_0 \ldots e_{l-1})$, satisfy the property $\nabla' \vdash \pi(e_i) \approx_{\alpha,C} e_{i+1}$, where $i+1$ abbreviates $i+1$ modulo l, and $\nabla' = \cup_{Y\in Var(\kappa')} dom(\pi)\#Y$, holds. The only case in which the length of an `epc` decreases is 3.a.i. Thus, when this case applies to a binary `epc`, say $(s_0\ s_1)$, an unitary `epc` (s) is built, being this of the form $(s_0 \oplus s_1)$ for a commutative function symbol $\oplus$. Since by the invariant one has that $\nabla' \vdash \pi(s_i) \approx_{\alpha,C} s_{i+1}$, for $i = 0, 1$, one has that $\nabla' \vdash \pi(s_0 \oplus s_1) \approx_{\alpha,C} s_0 \oplus s_1$; thus, one has that $\nabla' \vdash \pi(s) \approx_{\alpha,C} s$. In further steps in the construction of `epc`'s, new unitary `epc`'s (t') might be built from unitary `epc`'s (t) applying cases 3.a.ii, iii, iv and v, that, can easily be checked, preserve the property $\nabla' \vdash \pi(t') \approx_{\alpha,C} t'$, for $\nabla' = \cup_{Y\in Var(t')} dom(\pi)\#Y$, if $\nabla' \vdash \pi(t) \approx_{\alpha,C} t$, for $\nabla' = \cup_{Y\in Var(t)} dom(\pi)\#Y$. Therefore all unitary non-trivial `epc`'s give a correct solution of the form $\langle\nabla', \{X/s\}\rangle$ of the C FP problem $\langle\emptyset, \mathcal{X}, \pi.X \approx_? X\rangle$. Hence, if in addition, one has that $\nabla' \cup \Delta \vdash dom(\Delta|_X)\,\#\,s$, then for $\nabla := \nabla' \cup \Delta$, the pair $\langle\nabla, \{X/s\}\rangle \in \langle\Delta, \mathcal{X}, \{\pi.X \approx_? X\}\rangle_{Sol_G}$ is a solution of $\langle\Delta, \mathcal{X}, \{\pi.X \approx_? X\}\rangle$. □

Remark 6.11. *Assuming the symbols in the signature are denumerable, it is possible to enumerate the unitary* `epc`*'s and thus the* generated solutions*. This can be done as usual, enumerating first all possible unitary* `epc`*'s with an element of length bounded by a small natural, say twice the length of* π*, and using only the first* $|\pi|$ *symbols in the signature and atoms in* $dom(\pi)$*; then, this length is increased generating all extended unitary* `epc`*'s with elements of length* $|\pi| + 1$ *and using only the first* $|\pi + 1|$ *symbols in the signature and atoms in* $dom(\pi)$ *and so on.*

Remark 6.12. *The following result, proved by induction in the construction of the* `epc`*'s, is used in the proof of completeness of generated solutions for C FP problems.*

Lemma 6.5 (Extended pseudo-cycle correspondence for π and π^2)**.** *For $k \geq 1$, $(A_0 \cdots A_{2^k-1})$ is an* `epc` *for π if, and only if, there exist $(B_0 \cdots B_{2^{k-1}-1})$ and $(C_0 \cdots C_{2^{k-1}-1})$* `epc`*'s for π^2 with a substitution σ such that atoms in its image belong to $dom(\pi)\backslash dom(\pi^2)$, and for $0 \leq j \leq 2^{k-1} - 1$ one has $B_j\sigma \approx_{\alpha,C} A_{2j}$ and $C_j\sigma \approx_{\alpha,C} A_{2j+1}$.*

Proof. The proof is by induction in the construction of the extended pseudo-cycles.
Base Case.

1. Case 1 of Def. 6.4 does not apply since $k \geq 1$.

2. Let $(\overline{a}\ \overline{b})$ be a trivial `epc`for π, and let $(Y\ Y')$ be an `epc`for π^2. Notice that $dom(\pi^2) = dom(\pi)\backslash\{a \mid a \in \text{2-cycle of } \pi\}$.

Induction Step.

1. By Def. 6.4(3.a.i), from an `epc`for π, $(A_0 \cdots A_{2^k-1})$ one can build another `epc`in steps A and/or B. By i.h., there exist `epc`'s for π^2: $(B_0 \cdots B_{2^{k-1}-1})$ and $(C_0 \cdots C_{2^{k-1}-1})$ and a substitution σ such that $B_j\sigma \approx_{\alpha,C} A_{2j}$ and $C_j\sigma \approx_{\alpha,C} A_{2j+1}$. We consider two cases, depending on whether the `epc`obtained has length equal to 2^k or 2^{k-1}:

   (a) $(A_0 \star A_j \cdots A_{2^k-1} \star A_{2^{k-1}+j})$ with $j \neq 2^{k-1}$, then the length of the new `epc`does not change. From the two `epc`'s for π^2 we build the `epc`'s in the following way:

   - j is even: $(B_0 \star B_{\frac{j}{2}} \cdots B_{2^{k-1}-1} \star B_{2^{k-1}+\frac{j}{2}-1})$ and $(C_0 \star C_{\frac{j}{2}} \cdots C_{2^{k-1}-1} \star C_{2^{k-1}+\frac{j}{2}-1})$. Notice that the conditions of σ are preserved, for instance, $(B_0 \star B_{\frac{j}{2}})\sigma = A_0 \star A_j$.
   - j is odd: $(B_0 \star C_{\frac{j-1}{2}} \cdots B_{2^{k-1}-1} \star C_{2^{k-1}+\frac{j-1}{2}-1})$ and $(C_0 \star B_{\frac{j+1}{2}} \cdots C_{2^{k-1}-1} \star B_{2^{k-1}+\frac{j+1}{2}-1})$. Notice that $(B_0 \star C_{\frac{j-1}{2}})\sigma = B_0\sigma \star C_{\frac{j-1}{2}}\sigma = A_0 \star A_j$, similarly, one can check that the two `epc`'s satisfy the conditions on σ.

   (b) $(A_0 \star A_{2^{k-1}} \cdots A_{2^{k-1}-1} \star A_{2^k-1})$ with $j = 2^{k-1}$. From the `epc`'s for π^2 we build $(B_0 \star B_{2^{k-2}} \cdots B_{2^{k-2}-1} \star B_{2^{k-1}-1})$ and $(C_0 \star C_{2^{k-2}} \cdots C_{2^{k-2}-1} \star C_{2^{k-1}-1})$.

2. By Def. 6.4(3.a.ii), from `epc`'s for π, $(A_0 \cdots A_{2^k-1})$ and $(A'_0 \cdots A'_{2^{k'}-1})$, we build an `epc`for π, $(A_0 \star A'_j \cdots A_{2^k-1} \star A'_{2^{k'}-1})$, for some $0 \leq j \leq 2^k - 1$ (the case $0 \leq j \leq 2^{k'} - 1$ is analogous).

   By i.h., there exist `epc`'s for π^2, $(B_0 \cdots B_{2^{k-1}-1})$ and $(C_0 \cdots C_{2^{k-1}-1})$ for $(A_0 \cdots A_{2^k-1})$, and $(B'_0 \cdots B'_{2^{k'-1}-1})$ and $(C'_0 \cdots C'_{2^{k'-1}-1})$ for $(A'_0 \cdots A'_{2^{k'}-1})$, satisfying the conditions for σ, σ' respectively. We can choose B_i, C_i, B'_j, C'_j such that $var(B_i, C_i) \cap var(B'_j, C'_j) = \emptyset$. Then we have consider two cases:

- for a j even: take $(B_0 \star B'_{\frac{j}{2}} \cdots B_{2^{k-1}-1} \star B'_{2^{k'-1}+\frac{j}{2}-1})$ and $(C_0 \star C'_{\frac{j}{2}} \cdots C_{2^{k-1}-1} \star C_{2^{k'-1}+\frac{j}{2}-1})$.
- for a j odd: $(B_0 \star C'_{\frac{j-1}{2}} \cdots B_{2^{k-1}-1} \star C'_{2^{k'-1}+\frac{j-1}{2}-1})$ and $(C_0 \star B'_{\frac{j-1}{2}}) \cdots C_{2^{k-1}-1} \star B'_{2^{k'-1}+\frac{j-1}{2}-1})$.

Its straightforward to check that the conditions for these $\texttt{epc}$of π^2 hold for $\sigma \cup \sigma'$.

3. By Def. 6.4(3.a.iii), from $\texttt{epc}$'s for π, $(A_0 \cdots A_{2^k-1})$ and $(A'_0 \cdots A'_{2^{k'}-1})$, we build the $\texttt{epc}$of π: $(\langle A_0, A'_j\rangle \cdots \langle A_{2^k-1}, A'_{2^{k'}+j-1}\rangle)$.

   By i.h., there exist $\texttt{epc}$'s for π^2, $(B_0 \cdots B_{2^{k-1}-1})$ and $(C_0 \cdots C_{2^{k-1}-1})$ for $(A_0 \cdots A_{2^k-1})$, and $(B'_0 \cdots B'_{2^{k'-1}-1})$ and $(C'_0 \cdots C'_{2^{k'-1}-1})$ for $(A'_0 \cdots A'_{2^{k'}-1})$, satisfying the conditions for σ, σ' resp. Similarly to the previous case, one can check that the result holds, depending on whether j is even or odd.

4. By Def. 6.4(3.a.iv), from $(A_0 \cdots A_{2^k-1})$ an $\texttt{epc}$for π, one can build either $([e]A_0 \cdots [e]A_{2^k-1})$ or $(g\,A_0 \cdots g\,A_{2^k-1})$. By i.h., there exist $\texttt{epc}$'s for π^2: $(B_0 \cdots B_{2^{k-1}-1})$ and $(C_0 \cdots C_{2^{k-1}-1})$ and a substitution σ satisfying the requirements. Its clear that $([e]B_0 \cdots [e]B_{2^{k-1}-1})$ and $([e]C_0 \cdots [e]C_{2^{k-1}-1})$ are $\texttt{epc}$'s in π^2, for $([e]A_0 \cdots [e]A_{2^k-1})$, and similarly, $(g\,B_0 \cdots g\,B_{2^{k-1}-1})$ and $(g\,C_0 \cdots g\,C_{2^{k-1}-1})$ are $\texttt{epc}$'s in π^2, for $(g\,A_0 \cdots g\,A_{2^k-1})$.

5. By Def. 6.4(3.a.v), from $(A_0 \cdots A_{2^k-1})$ and $(a_0 \cdots a_{2^l-1})$, resp. an $\texttt{epc}$and a permutation cycle of π, we build $([a_j]A_0 \cdots [a_{2^l+j-1}]A_{2^k-1})$, another $\texttt{epc}$for π. By i.h., we have for π^2 $\texttt{epc}$'s $(B_0 \cdots B_{2^{k-1}-1})$ and $(C_0 \cdots C_{2^{k-1}-1})$ and a substitution σ satisfying the necessary conditions.

   - Case $l = 1$, then $(a_0 \cdots a_{2^l-1}) = (a\,b)$, take $([a]B_0 \cdots [a]B_{2^{k-1}-1})$ and $([b]C_0 \cdots [b]C_{2^{k-1}-1})$.
   - Otherwise, $(a_0\,a_2 \cdots a_{2^l-2})$ and $(a_1\,a_3 \cdots a_{2^l-1})$ are permutation cycles of π^2, and the we take $([a_j]B_0 \cdots [a_{j-2}]B_{2^{k-1}-1})$ and $([a_{j+1}]C_0 \cdots [a_{j-1}]C_{2^{k-1}-1})$.

   In both cases σ satisfy the necessary conditions.

□

Example 6.11. *For $(a\,b)$ and $(c\,d\,e\,f)$, permutation cycles of π, one has that (a), (b), $(c\,e)$ and $(d\,f)$ are permutation cycles of π^2, and also, $a, b \in dom(\pi) \backslash dom(\pi^2)$. Therefore, supposing that '$+$', '$*$' and '$\star$' are commutative function symbols, $((\overline{c} * \overline{e}) + \overline{a}) \star ((\overline{d} * \overline{f}) + \overline{b})$ and $((\overline{c} * \overline{e}) + Y) \star ((\overline{d} * \overline{f}) + Y')$ are respectively unitary $\texttt{epc}$'s of π and π^2. Then:*

- $\langle \Delta, \{X/((\overline{c} * \overline{e}) + \overline{a}) \star ((\overline{d} * \overline{f}) + \overline{b})\}\rangle \in \langle \Delta, \emptyset, \pi.X \approx_? X\rangle_{Sol_G}$ *iff*

- $\langle \Delta', \{X/((\overline{c} * \overline{e}) + Y) \star ((\overline{d} * \overline{f}) + Y')\}\rangle \in \langle \Delta, \emptyset, \pi^2.X \approx_? X\rangle_{Sol_G}$,

where $\Delta' = \Delta \cup dom(\pi^2)\#Y, Y' \cup dom(\Delta|_X)\#Y, Y'$. *So the* σ *of Lem. 6.5 is* $\{Y/\overline{a}, Y'/\overline{b}\}$, *so that* $((\overline{c} * \overline{e}) + \overline{a} \ \ (\overline{d} * \overline{f}) + \overline{b})$ *is an* epc *of* π, $((\overline{c} * \overline{e}) + Y)$ *and* $((\overline{d} * \overline{f}) + Y')$ *are* epc*'s of* π^2, *with* $((\overline{c} * \overline{e}) + Y)\sigma = (\overline{c} * \overline{e}) + \overline{a}$ *and* $((\overline{d} * \overline{f}) + Y')\sigma = (\overline{d} * \overline{f}) + \overline{b}$.

Example 6.12. *Let* $\pi = (a\ b\ c\ d\ e\ f\ g\ h)$ *then* $\pi^2 = (a\ c\ e\ g)(b\ d\ f\ h)$. *There are solutions of* $\langle \emptyset, \emptyset, \pi^2 \cdot X \approx_? X\rangle$ *that are not solutions of* $\langle \emptyset, \emptyset, \pi.X \approx_? X\rangle$*:*

- $\langle \emptyset, X/(\overline{a} \oplus \overline{e}) \oplus (\overline{c} \oplus \overline{g})\rangle, \langle \emptyset, X/(\overline{b} \star \overline{f}) \oplus (\overline{d} \star \overline{h})\rangle \in \langle \emptyset, \emptyset, \pi^2 \cdot X \approx_? X\rangle_{Sol_G}$;
- $\langle \emptyset, X/((\overline{a} \oplus \overline{e}) \oplus (\overline{c} \oplus \overline{g})) \oplus ((\overline{b} \star \overline{f}) \oplus (\overline{d} \star \overline{h}))\rangle \in \langle \emptyset, \emptyset, \pi^2 \cdot X \approx_? X\rangle_{Sol_G}$

but none of them is a solution for $\langle \emptyset, \emptyset, \pi.X \approx_? X\rangle$.

However there exist solutions in the intersection of both problems, for instance,

$$\langle \emptyset, X/((\overline{a} \oplus \overline{e}) \oplus (\overline{c} \oplus \overline{g})) * (X/(\overline{b} \oplus \overline{f}) \oplus (\overline{d} \oplus \overline{h}))\rangle.$$

Theorem 6.7 (Completeness of solutions for singleton C FP problems)**.**
Let $\langle \Delta, \mathcal{X}, \{\pi.X \approx_? X\}\rangle$ *be a singleton C FP problem with a solution* $\langle \nabla, \{X/s\}\rangle$. *Then there exists* $\langle \nabla', \{X/t\}\rangle \in \langle \Delta, \mathcal{X}, \{\pi.X \approx_? X\}\rangle_{Sol_G}$ *such that* $\langle \nabla', \{X/t\}\rangle \preceq \langle \nabla, \{X/s\}\rangle$.

Proof. Since $\langle \nabla, \{X/s\}\rangle$ is a solution of the problem, it follows that $\nabla \vdash \Delta\{X/s\}$ and $\nabla \vdash \pi(s) \approx_{\alpha,C} s$. The proof is done by induction on the structure of s.
Base Case. This case will be split in two parts.

1. $s = \overline{a}$.

   The pair $\langle \nabla, \{X/\overline{a}\}\rangle$ is a solution only if $a \notin dom(\Delta|_X) \cup dom(\pi)$, then $\emptyset \vdash \pi \cdot \overline{a} = \overline{a}$. Let Y be a new variable and $\nabla' = dom(\Delta|_X)\#Y \cup dom(\pi)\#Y$, then $\langle \nabla', \{X/Y\}\rangle$ is a generated solution. Let $\sigma = \{Y/\overline{a}\}$, notice that $\nabla \vdash \nabla'\sigma$ and $Y\sigma = \overline{a}$. Therefore, $\langle \nabla', \{X/Y\}\rangle \preceq \langle \nabla, \{X/\overline{a}\}\rangle$.

2. $s = \pi'.Y$ and $dom(\pi)\#\pi'.Y$.

   Notice that $\langle \nabla, \{X/\pi'.Y\}\rangle \in \langle \Delta, \mathcal{X}, \pi.X \approx_? X\rangle_{Sol_C}$ only if $\nabla \vdash dom(\Delta|_X) \# \pi'.Y, dom(\pi) \# \pi'.Y$, that is, if $\nabla \vdash (\pi')^{-1} \cdot dom(\Delta|_X) \# Y$ and $\nabla \vdash (\pi')^{-1} \cdot dom(\pi) \# Y$, so that $\Delta \cup ((\pi')^{-1} \cdot dom(\Delta|_X) \cup (\pi')^{-1} \cdot dom(\pi)) \# Y \subset \nabla$.

   Let $\langle \nabla', \{X/Z\}\rangle \in \langle \Delta, \mathcal{X}, \pi.X \approx_? X\rangle_{Sol_G}$ with $\nabla' = \Delta \cup dom(\pi) \cup dom(\Delta|_X)\#Z$, Consider the substitution $\sigma = \{Z/\pi'.Y\}$, then $\nabla \vdash Z\sigma \approx_{\alpha,C} \pi'.Y$ and $\nabla'\sigma = \Delta\sigma \cup (dom(\pi) \cup dom(\Delta|_X))\#Z\sigma = \Delta \cup (\pi')^{-1} \cdot dom(\pi)\#Y \cup (\pi')^{-1} \cdot dom(\Delta|_X)\#Y$, so $\nabla \vdash \nabla'\sigma$. Therefore, $\langle \nabla', \{X/Z\}\rangle \preceq \langle \nabla, \{X/\pi'.Y\}\rangle$.

Induction Step.

1. $s = \langle s_1, s_2 \rangle$

   In this case $\nabla \vdash \pi(\langle s_1, s_2 \rangle) \approx_{\alpha,C} \langle s_1, s_2 \rangle$, that is, $\nabla \vdash \langle \pi(s_1), \pi(s_2) \rangle \approx_{\alpha,C} \langle s_1, s_2 \rangle$, which implies in $\nabla \vdash \pi(s_i) \approx_{\alpha,C} s_i$, for $i = 1, 2$.

   By IH and Defs. 6.4 and 6.5, there exist $\langle \nabla'_1, \{X/t_1\} \rangle, \langle \nabla'_2, \{X/t_2\} \rangle \in \langle \Delta, \mathcal{X}, \pi.X \approx_? X \rangle_{Sol_G}$ such that (t_1), (t_2) and $(\langle t_1, t_2 \rangle)$ are unitary `epc`'s w.r.t. π. Furthermore $\langle \nabla'_i, \{X/t_i\} \rangle \preceq \langle \nabla, \{X/s_i\} \rangle$, i.e., there exist substitutions λ_i such that $\nabla \vdash \nabla_i \lambda_i$ and $\nabla \vdash t_i \lambda_i \approx s_i$, for $i = 1, 2$. One can choose (t_1) and (t_2) such that $Var(t_1) \cap Var(t_2) = \emptyset$ and $dom(\lambda_i) \cap Var(s_j) = \emptyset$, for $i, j = 1, 2$. Then, $\nabla \vdash \langle t_1, t_2 \rangle \lambda_1 \lambda_2 \approx_{\alpha,C} \langle s_1, s_2 \rangle$, and $\nabla \vdash (\nabla_1 \cup \nabla_2) \lambda_1 \lambda_2$, that is, $\langle \nabla_1 \cup \nabla_2, \{X/\langle t_1, t_2 \rangle\} \rangle \preceq \langle \nabla, \{X/\langle s_1, s_2 \rangle\} \rangle$.

2. $s = fs'$

   Since $\nabla \vdash \pi \cdot fs' \approx_{\alpha,C} fs'$, it follows that $\nabla \vdash f(\pi(s')) \approx_{\alpha,C} fs'$ and therefore, $\nabla \vdash \pi(s') \approx_{\alpha,C} s'$. By IH and Defs. 6.4 and 6.5, there exist $\langle \nabla', \{X/t'\} \rangle \in \langle \Delta, \mathcal{X}, \pi.X \approx_? X \rangle_{Sol_G}$ such that (t') and (ft') are unitary `epc`'s w.r.t. π. Furthermore $\langle \nabla', \{X/t'\} \rangle \preceq \langle \nabla, \{X/s'\} \rangle$, that is, there exist a substitution σ such that $\nabla \vdash \nabla' \sigma$ and $\nabla \vdash t' \sigma \approx_{\alpha,C} s'$, and since $\nabla \vdash ft' \sigma \approx_{\alpha,C} f(t' \sigma) \approx_{\alpha,C} fs'$ and adding f at the top of t' does not change the variables of t', therefore, $\langle \nabla', \{X/ft'\} \rangle \in \langle \Delta, \mathcal{X}, \pi.X \approx_? X \rangle_{Sol_G}$ and $\langle \nabla', \{X/ft'\} \rangle \preceq \langle \nabla, \{X/fs'\} \rangle$.

3. $s = [e]s'$.

   (a) $e \notin dom(\pi)$

   Since $\nabla \vdash \pi([e]s') \approx_{\alpha,C} [e]s'$, it follows that $\nabla \vdash \pi(s') \approx_{\alpha,C} s'$, i.e., $\langle \nabla, X/s' \rangle$ is a solution for $\langle \Delta, \mathcal{X}, \pi.X \approx_? X \rangle$. By IH and Defs. 6.4 and 6.5, there exist $\langle \nabla', \{X/t'\} \rangle \in \langle \Delta, \mathcal{X}, \pi.X \approx_? X \rangle_{Sol_G}$ such that (t') and $([e]t')$ are unitary `epc`'s w.r.t. π. Furthermore $\langle \nabla', \{X/t'\} \rangle \preceq \langle \nabla, \{X/s'\} \rangle$, i.e., there exist a substitution σ such that $\nabla \vdash \nabla' \sigma$ and $\nabla \vdash t' \sigma \approx_{\alpha,C} s'$, therefore, $\langle \nabla', \{X/[e]t'\} \rangle \in \langle \Delta, \mathcal{X}, \pi.X \approx_? X \rangle_{Sol_G}$ and $\langle \nabla', \{X/[e]t'\} \rangle \preceq \langle \nabla, \{X/[e]s'\} \rangle$.

   (b) $e \in dom(\pi)$.

   By hypothesis, $\nabla \vdash \pi([e]s') \approx_{\alpha,C} [e]s'$, i.e., $\nabla \vdash [\pi \cdot e](\pi(s')) \approx_{\alpha,C} [e]s'$, and $\nabla \vdash \pi(s') \approx_{\alpha,C} (\pi \cdot e \ \ e)(s')$ only if $\nabla \vdash (\pi \cdot e) \# s'$. Notice that e occurs in s' iff $\pi \cdot e$ occurs in s'. Therefore, for $\nabla \vdash e \# s'$, it follows that $\nabla \vdash \pi(s') \approx_{\alpha,C} s'$ and the result follows by induction hypothesis.

4. $s = s_1 \oplus s_2$

   This case has two parts:

(a) $\nabla \vdash \pi(s_1) \approx_{\alpha,C} s_1$ and $\nabla \vdash \pi(s_2) \approx_{\alpha,C} s_2$.

By IH and Defs. 6.4 and 6.5, there exist $\langle \nabla'_1, \{X/t_1\}\rangle, \langle \nabla'_2, \{X/t_2\}\rangle \in \langle \Delta, \mathcal{X}, \pi.X \approx_? X\rangle_{Sol_G}$ such that (t_1), (t_2) and $(t_1 \oplus t_2)$ are unitary `epc`'s w.r.t. π. Furthermore $\langle \nabla'_i, \{X/t_i\}\rangle \preceq \langle \nabla, \{X/s_i\}\rangle$, i.e., there exist substitutions λ_i such that $\nabla \vdash \nabla_i \lambda_i$ and $\nabla \vdash t_i \lambda_i \approx s_i$, for $i = 1, 2$. One can choose (t_1) and (t_2) such that $Var(t_1) \cap Var(t_2) = \emptyset$ and $dom(\lambda_i) \cap Var(s_j) = \emptyset$, for $i, j = 1, 2$. Then, $\nabla \vdash (t_1 \oplus t_2)\lambda_1\lambda_2 \approx_{\alpha,C} (s_1 \oplus s_2)$, and $\nabla \vdash (\nabla_1 \cup \nabla_2)\lambda_1\lambda_2$, that is, $\langle \nabla_1 \cup \nabla_2, \{X/t_1 \oplus t_2\}\rangle \preceq \langle \nabla, \{X/s_1 \oplus s_2\}\rangle$.

(b) $\nabla \vdash \pi(s_1) \approx_{\alpha,C} s_2$ and $\nabla \vdash \pi(s_2) \approx_{\alpha,C} s_1$.

Notice that $\nabla \vdash \pi^2(s_1) \approx_{\alpha,C} \pi(s_2) \approx_{\alpha,C} s_1$ and $\nabla \vdash \pi^2(s_2) \approx_{\alpha,C} \pi(s_1) \approx_{\alpha,C} s_2$. Therefore, $\langle \nabla, \{X/s_1\}\rangle$ and $\langle \nabla, \{X/s_2\}\rangle$ are solutions of $\langle \Delta, \mathcal{X}, \pi^2.X \approx_? X\rangle$. By IH, there exist $\langle \nabla_1, \{X/t_1\}\rangle, \langle \nabla_2, \{X/t_2\}\rangle \in \langle \Delta, \mathcal{X}, \pi^2 \cdot X \approx_? X\rangle_{Sol_G}$ such that $\langle \nabla_i, \{X/t_i\}\rangle \preceq \langle \nabla, \{X/s_i\}\rangle$. Then there exist substitutions λ_i such that $\nabla \vdash \nabla_i \lambda_i$ and $\nabla \vdash t_i \lambda_i \approx_{\alpha,C} s_i$, for $i = 1, 2$.

One can choose (t_1) and (t_2) such that $Var(t_1) \cap Var(t_2) = \emptyset$ and $dom(\lambda_i) \cap Var(s_j) = \emptyset$, for $i, j = 1, 2$.

Therefore, $\langle \nabla_1 \cup \nabla_2, X/t_1 \oplus t_2\rangle \in \langle \Delta, \mathcal{X}, \pi^2.X \approx_? X\rangle_{Sol_G}$ and $\langle \nabla_1 \cup \nabla_2, X/t_1 \oplus t_2\rangle \preceq \langle \nabla, X/s_1 \oplus s_2\rangle$, via substitution $\lambda = \lambda_1\lambda_2$.

Notice that $\nabla \vdash \pi(t_1)\lambda \approx_{\alpha,C} \pi(s_1) \approx_{\alpha,C} s_2 \approx_{\alpha,C} t_2\lambda$ and analogously, $\nabla \vdash \pi(t_2)\lambda \approx_{\alpha,C} t_1\lambda$. Hence, λ is a solution for the C-unification problem $\{\pi(t_1) =_? t_2, \pi(t_2) =_? t_1\}$. Let $\langle \nabla', \lambda'\rangle$ be a solution more general than $\langle \nabla, \lambda\rangle$ such that the atoms in the image of λ' are in $dom(\pi)\backslash dom(\pi^2)$. Since (t_1) and (t_2) are unitary `epc`'s of π^2, it follows by Lem. 6.5, that $(t_1\lambda'\ t_2\lambda')$ is an `epc` for π. By Def. 6.4, $(t_1\lambda' \oplus t_2\lambda')$ is a unitary `epc`for π, such that $\langle \nabla', \{X/t_1\lambda' \oplus t_2\lambda'\}\rangle \in \langle \Delta, \mathcal{X}, \{\pi.X \approx_? X\}\rangle_{Sol_G}$ and $\langle \nabla', \{X/t_1\lambda' \oplus t_2\lambda'\}\rangle \preceq \langle \nabla, \{X/s_1 \oplus s_2\}\rangle$.

□

Remark 6.13. *The algorithm of syntactic commutative unification presented in Sec. 2.1.2 is used in the proof of Lem. 6.7 (case 4.b) and Def. 6.6.*

Definition 6.6 (General C-matchers)**.** *Let s_i, for $i = 1..k$, be terms. A* most general C-matcher *of these terms, if it exists, is a most general C-solution δ of the C-unification problem $\{s_i =_? Z\}_{i=1..k}$, where Z is a new variable for s_i, with $i = 1..k$.*

Remark 6.14. *Alternatively, Def. 6.4 could be restricted to ground terms (by removing the first case in the construction of* `epc`*'s), and then instead of computing C-matchers via C-unification, one could use an $\approx_{\alpha,C}$-equality checker (for example, the one specified in*

Sec. 4.3). This would also simplify case 3.a.iv) in Def. 6.4, since it would be sufficient to consider just one atom e' not in $dom(\pi)$.

Definition 6.7 (Generated solutions for a variable)**.** *Let the C FP problems for X in $\mathcal{P}$ be given by $\langle \nabla, \mathcal{X}, \pi_i.X \approx_? X \rangle$, for $\pi_i \in \Pi_X$, and such that $|\Pi_X| = k$. If there exist*

- *solutions $\langle \nabla_i, \mathcal{X}, \{X/t_i\} \rangle \in \langle \nabla, \mathcal{X}, \pi_i.X \approx_? X \rangle_{Sol_G}$ for each C FP problem and*
- *a most general C-matcher δ of the terms $\{t_i\}_{i=1..k}$ with X as new variable*

such that the problem $\langle \emptyset, \cup_{(a\#Y)\in\nabla''}\{a\#Y\delta\} \rangle$, where $\nabla'' := \cup_{i=1}^{k} \nabla_i$, has a solution $\langle \nabla', \emptyset \rangle$, then one says that $\langle \nabla', \{X/X\delta\} \rangle$ is a generated solution *for X. The set of all generated solutions is denoted by $[X]_{\mathcal{P}_G}$.*

Example 6.13. *Let $P_i := \pi_i.X \approx_? X$, for $i = 1..3$, be C FP equations for $\pi_1 = (a\ b\ c\ d)$, $\pi_2 = (a\ c)$ and $\pi_3 = (b\ d)$ and suppose that $\mathcal{P} := \langle \nabla, \mathcal{X}, P \rangle$ is a C FP problem where P_i for $i = 1..3$ are the C FP equations for X in P.*

1. $\langle \nabla \cup a, b, c, d\#Y, \delta_1 := \{X/((a * c) * (b * d)) \oplus Y\} \rangle \in \langle \nabla, \mathcal{X}, P_1 \rangle_{Sol_G}$;
2. $\langle \nabla \cup a, c\#Y', Y'', \delta_2 := \{X/((a * c) * Y') \oplus Y''\} \rangle \in \langle \nabla, \mathcal{X}, P_2 \rangle_{Sol_G}$*; and*
3. $\langle \nabla \cup b, d\#Y_1', Y_1'', \delta_3 := \{X/((b * d) * Y_1') \oplus Y_1''\} \rangle \in \langle \nabla, \mathcal{X}, P_3 \rangle_{Sol_G}$.

Remark 6.15. *Notice that, in Ex. 6.13, $\delta = \{X/((a*c)*(b*d)) \oplus Y'',\ Y'/(b*d),\ Y_1'/(a*c),\ Y/Y'', Y_1''/Y''\}$ is a most general C-solution of terms $\{t_i := X\delta_i\}$ with variable X.*

Remark 6.16. *Also in Ex. 6.13, according to the definition, the set of initial freshness constraints is given as $\nabla'' = \nabla \cup \{a, b, c, d\#Y,\ a, c\#Y', Y'',\ b, d\#Y_1', Y_1''\}$. Notice that $Y'' \in Var(im(\delta))$, have to satisfy the constraints on Y_1'', Y and X, that is, $a, b, c, d\#Y''$ is a new constraint on Y'', inherited from the constraints of the variables in the domain of δ. $\langle \nabla', \emptyset \rangle$ is the solution of $\langle \emptyset, \cup_{(a\#Y)\in\nabla''}\{a\#Y\delta\} \rangle$, and then it holds that $\nabla' \vdash dom(\nabla''|_Z) \# Z\delta$, for all $Z \in dom(\delta)$. Thus, $\langle \nabla', \{X/X\delta\} \rangle$ belongs to $[X]_{\mathcal{P}_G}$.*

Example 6.14. *(Continuing Ex. 6.8) Consider the singleton C FP problems on the variable X in $\mathcal{Q}_1$ of Ex. 6.7 relative to the variable set $\mathcal{Z} = \{X, Y\}$ and $X \notin \mathcal{X}$: $\langle \{a'\#X\}, \mathcal{X}, \{Eq_1 := (a\,c)(a'\,b').X \approx_? X\} \rangle$ and $\langle \{a'\#X\}, \mathcal{X}, \{Eq_2 := (a\,b\,c\,d)(a'\,b').X \approx_? X\} \rangle$. Since $a'\#X$ is in the freshness context, there is no combinatorial solution with occurrences of the atoms in the permutation cycle $(a'\,b')$. For the cycles $(a\,c)$ and $(a\,b\,c\,d)$, in equations Eq_1 and Eq_2, possible solutions include, respectively:*

- $\langle \nabla, \{X/(\overline{a} + \overline{c}) \star Z\} \rangle$, $\langle \nabla, \{X/(f\overline{a} + f\overline{c}) \star Z\} \rangle$, $\langle \nabla, \{X/([g]\overline{a} + [g]\overline{c}) \star Z\} \rangle$, *for* $\nabla = a, c, a', b'\#Z, a'\#X$;

- $\langle\nabla', \{X/(\overline{a}+\overline{c})\star(\overline{b}+\overline{d})\}\rangle$, $\langle\nabla', \{(f\overline{a}+f\overline{c})\star(f\overline{b}+f\overline{d})\}\rangle$, $\langle\nabla', \{([g]\overline{a}+[g]\overline{c})\star([g]\overline{b}+[g]\overline{d})\}\rangle$, *for* $\nabla' = a'\#X$.

Since the general C-matchers for pairs of these three solutions for Eq_1 *and* Eq_2 *are respectively* $\{Z/\overline{b}+\overline{d}\}$, $\{Z/f\overline{b}+f\overline{d}\}$ *and* $\{Z/[g]\overline{b}+[g]\overline{d}\}$, *the combined solutions for both singleton C FP problems are those given for* Eq_2.

Corollary 6.1 (Soundness and completeness of generated solutions for a variable). *Let* $\mathcal{P} = \langle\Delta, \mathcal{X}, P\rangle$ *be a C FP problem. Any solution in* $[X]_{\mathcal{P}_G}$ *is a solution of each C FP equation for* X *in* $\mathcal{P}$. *If* $\langle\nabla, \{X/s\}\rangle$ *is a solution for each C FP equation for* X *in* $\mathcal{P}$ *then there exists* $\langle\nabla', \{X/X\delta\}\rangle \in [X]_{\mathcal{P}_G}$ *such that* $\langle\nabla', \{X/X\delta\}\rangle \preceq \langle\nabla, \{X/s\}\rangle$

Proof. By Thm. 6.6 and Def. 6.5:

(*Soundness)* Each solution $\langle\nabla_i, \{X/t_i\}\rangle$ in $\langle\Delta, \mathcal{X}, \{\pi_i.X \approx_? X\}\rangle_{Sol_G}$ is a correct solution for $\langle\Delta, \mathcal{X}, \{\pi_i.X \approx_? X\}\rangle$, for $\pi_i \in \Pi_X$. Suppose $\langle\nabla', \{X/X\delta\}\rangle$ belongs to $[X]_{\mathcal{P}_G}$. Since δ is a C-solution of terms t_i with variable X, one has that $X\delta \approx_C t_i\delta$, and also that $\nabla_i \vdash \pi(t_i) \approx_{\alpha,C} t_i$. Thus, $\nabla' \vdash \pi(t_i)\delta \approx_{\alpha,C} t_i\delta$ since by definition one also has that $\nabla' \vdash dom(\nabla|_X)\#X\delta$, because by construction for all $Y \in Var(X\delta)$, ∇' includes the freshness constraints $dom(\nabla''|_X)\#Y$ and ∇'' is an extension of ∇.

(*Completeness)* For $|\Pi_X| = k$, $i = 1..k$, there are $\langle\nabla_i, \{X/t_i\}\rangle \in \langle\Delta, \mathcal{X}, \{\pi_i.X \approx_? X\}\rangle_{Sol_G}$, solution of $\langle\Delta, \mathcal{X}, \{\pi_i.X \approx_? X\}\rangle$, such that $\langle\nabla_i, \{X/t_i\}\rangle \preceq \langle\nabla, \{X/s\}\rangle$. Then, for each i, there exists a ∇_i such that $\nabla \vdash \nabla_i\lambda_i$ and $\nabla \vdash \{X/t_i\}\lambda_i \approx \{X/s\}$. One can choose each t_i in a way to satisfy $\cap_{i=1}^k Var(t_i) = \emptyset$, and then for $\lambda = \lambda_1 \cdots \lambda_k$ and $\nabla'' = \cup_{i=1}^k \nabla_i$, one also has $\nabla \vdash \nabla''\lambda$ and $\nabla \vdash \{X/t_i\}\lambda \approx \{X/s\}$.

Notice that $\langle\nabla, \lambda\rangle$ is a nominal C-solution for the problem $\langle\nabla_i, \mathcal{X}, \cup_{i=1}^k\{t_i \approx_? X\}\rangle$. Then, given δ, a most general C-solution for $\{t_i =_? X\}_{i=1..k}$, it holds that there exists λ' such that $\nabla \vdash \delta\lambda' \approx \lambda$.

Let $\langle\nabla', \emptyset\rangle$ be a solution of $\langle\emptyset, \mathcal{X}, \cup_{(a\#Y)\in\nabla''}\{a\#Y\delta\}\rangle$, then, by Def. 6.5, one has that $\langle\nabla', \{X/X\delta\}\rangle \in [X]_{\mathcal{P}_G}$, and so, since $\nabla \vdash \nabla''\lambda$, also that $\nabla \vdash \nabla''\delta\lambda'$, which is the same that $\nabla \vdash \nabla'\lambda'$. On the other hand, $X\delta \approx_C t_i\delta$ and then $\nabla \vdash s \approx_{\alpha,C} t_i\lambda \approx_{\alpha,C} t_i\delta\lambda' \approx_C X\delta\lambda'$, which implies $\nabla \vdash \{X/s\} \approx \{X/X\delta\}\lambda'$. Hence, $\langle\nabla', \{X/X\delta\}\rangle \preceq \langle\nabla, \{X/s\}\rangle$. □

Definition 6.8 (Generated Solutions for C FP problems). *Let* $\mathcal{P}$ *be a C FP problem. The set of generated solutions for* $\mathcal{P}$, *denoted as* $[\mathcal{P}]_{Sol_G}$, *is defined as the set that contains all solutions of the form*

$$\left\langle \bigcup_{X\in Var(P)} \nabla_X, \quad \bigcup_{X\in Var(P)} \{X/s_X\} \right\rangle,$$ *where each* $\langle\nabla_X, \{X/s_X\}\rangle \in [X]_{\mathcal{P}_G}$.

Example 6.15. *(Continuing Ex. 6.14) Consider the third singleton C FP problem on the variable* Y *in* $\mathcal{Q}_1$ *relative to* $\mathcal{Z} = \{X, Y\}$ *and* $X \notin \mathcal{X}$*:* $\langle\{a'\#Y\}, \mathcal{X}, \{(a\,b\,c)(a'\,b').Y \approx_? Y\}\rangle$*. There exists no possible combinatorial solution since* $a'\#Y$ *is in the freshness context and the length of permutation cycle* $(a\,b\,c)$ *is not a power of two. The only possible solution is given as* $\langle a, b, c, a', b'\#Y', \{Y/Y'\}\rangle$*. Hence, using the solutions in Ex. 6.14 for the C FP equations on* X*, one has the following solutions for the C FP problem* $\mathcal{Q}_1$*, where* $\Delta = a', b'\#X, a, b, c, a', b'\#Y'$*:*

- $\langle\Delta, \{X/(\overline{a} + \overline{c}) \star (\overline{b} + \overline{d}), Y/Y'\}\rangle$
- $\langle\Delta, \{X/(f\overline{a} + f\overline{c}) \star (f\overline{b} + f\overline{d}), Y/Y'\}\rangle$
- $\langle\Delta, \{X/([g]\overline{a} + [g]\overline{c}) \star ([g]\overline{b} + [g]\overline{d}), Y/Y'\}\rangle$

A similar analysis can be done for the C FP problem $\mathcal{Q}_2$ *in Ex. 6.7. Also, for the C FP equations* $(a\,b).Y \approx_? Y$ *and* $(a\,d\,c\,b)(a'\,b').Y \approx_? Y$*, the permutation cycle* $(a'\,b')$ *avoids any possible combinatorial solution with occurrences of the atoms* a' *or* b'*. Cycles* $(a\,b)$ *and* $(a\,d\,c\,b)$ *allow combinatorial solutions for each of these equations, but it will be seen (Ex. 6.20) that they cannot be combined.*

Corollary 6.2 (Soundness and completeness of generated solutions for C FP problems)**.** *Let* $\mathcal{P}$ *be a C FP problem. Any solution in the set of solutions* $[\mathcal{P}]_{Sol_G}$ *is a correct solution of* $\mathcal{P}$*. For any* $\langle\nabla, \delta\rangle$ *solution of* $\mathcal{P}$ *there exist a pair* $\langle\nabla', \sigma\rangle \in [\mathcal{P}]_{Sol_G}$ *such that* $\langle\nabla', \sigma\rangle \preceq \langle\nabla, \delta\rangle$*.*

Proof. By Def. 6.8 and Cor. 6.1:

(Soundness) A solution of $\mathcal{P}$ is of the form $\left\langle \bigcup_{X \in Var(P)} \nabla_X, \bigcup_{X \in Var(P)} \{X/s_X\} \right\rangle$, where each $\langle\nabla_X, \{X/s_X\}\rangle \in [X]_{\mathcal{P}_G}$ is a correct solution for all C FP equations in $\mathcal{P}$ for the variable X, this completes the soundness proof.

(Completeness) Let $\mathcal{P} = \{\langle\Delta, \mathcal{X}, \cup_{i=1}^{k}\{\pi_{i_1}.X_1 \approx_? X_1\}_{\pi_{i_1} \in \Pi_{X_1}}\}\rangle$ and $\langle\nabla, \delta\rangle$ be a solution of $\mathcal{P}$. There exist more general solutions $\langle\nabla_j, \{X_j/t_j\}\rangle \in [X_j]_{\mathcal{P}_G}$, for $j = 1, \ldots, k$; i.e., $\langle\nabla_j, \{X_j/t_j\}\rangle \preceq \langle\nabla, \delta\rangle$; hence, there is a solution for $\mathcal{P}$ of the form $\langle\bigcup_j \nabla_j, \bigcup_j\{X_j/t_j\}\rangle$ is in $[\mathcal{P}]_{Sol_G}$ and $\langle\bigcup_j \nabla_j, \bigcup_j\{X_j/t_j\}\rangle \preceq \langle\nabla, \delta\rangle$. □

Remark 6.17. *A greedy procedure for the generation of solutions in* $[X]_{\mathcal{P}}$ *proceeds as follows. Follow the construction of generated solutions in Def. 6.5 for each C FP problem* $\langle\nabla, \mathcal{X}, \pi_i.X \approx_? X\rangle$ *in* P*, where* $\pi_i \in \Pi_X$*, as given in Lem. 6.1; for each generated solution* $\langle\nabla', \{X/s\}\rangle$ *build the freshness context* $\nabla'' = \nabla' \cup \bigcup_{Y \in Var(s)} dom(\nabla|_X)\#Y \cup dom(\Pi_X)\#Y$ *and check whether* $\langle\nabla'', \{X/s\}\rangle$ *is a solution for all* $\langle\nabla, \mathcal{X}, \pi_i.X \approx_? X\rangle$*, for* $\pi_i \in \Pi_X$*. Here,* $dom(\Pi_X)\#Y$ *abbreviates* $\cup_{\pi_i \in \Pi_X} dom(\pi_i)\#Y$*.*

6.2.2 Improvements in the generation of solutions

The greedy procedure can be improved eliminating generation of solution of non interesting permutation cycles in Π_X, according to the observations below.

In first place, notice that according to the theory of pseudo-cycles, one is interested in building solutions with atoms that occur only in permutation cycles of length a power of two in all permutations $\pi \in \Pi_X$.

In second place, notice that if there exist permutation cycles of length a power of two $\kappa_i \in \pi_i$ and $\kappa_j \in \pi_j$, for $\pi_i, \pi_j \in \Pi_X$, such that $dom(\pi_i) \cap dom(\pi_j) \neq \emptyset$, $dom(\pi_i) \backslash dom(\pi_j) \neq \emptyset$ and $dom(\pi_j) \backslash dom(\pi_i) \neq \emptyset$, then there might not be possible solutions with occurrences of atom terms in the domain of π_i and/or π_j for the C FP equations related with permutations π_i and π_j. The simplest example is given by permutation cycles $(a\, b)$ and $(a\, c)$. The precise relation between permutation cycles that allows for construction of solutions for all permutations in Π_X is given in the next definition.

Definition 6.9 (Permutation factor)**.** *A permutation π is said to be an n-*factor *of a permutation π' whenever there exists n such that $\pi^n = \pi'$.*

Example 6.16. *Let $\pi = (a\, b\, c\, d\, e\, f\, g\, h)$. The odd powers of π, π^1, $\pi^3 = (a\, d\, g\, b\, e\, h\, c\, f)$, $\pi^5 = (a\, f\, c\, h\, e\, b\, g\, d)$ and $\pi^7 = (a\, h\, g\, f\, e\, d\, c\, b)$ are the only factors of π.*

Remark 6.18. *For a permutation cycle κ of length 2^k, the factors corresponding to permutation cycles of the same length are exactly the permutations cycles κ^p, for p odd such that $0 < p < 2^k$; also, if λ is a p-factor of κ then λ is the q-factor of κ, where q is the minimum odd number such that $0 < q < 2^k$ and $p \cdot q = 1$ modulo 2^k. For instance, if κ is a permutation cycle of length 2^4, κ^3, κ^5, κ^7, etc, are respectively the 11- 13- and 7-factors, etc, of κ.*

The key observation about permutation cycles κ and λ, of respective lengths 2^k and 2^l, for $k \geq l \geq 0$, such that, $\kappa^{2^{k-l}}$ contains a permutation cycle, say ν, that is a p-factor of λ, is that this happens if and only if regarding elements in $dom(\lambda)$, possible generated solutions from both permutation cycles coincide. Indeed, first, notice that either $l = 0$ and then $\nu = \lambda$ or $l > 0$ and $\lambda^{2^{l-1}}$ consists of 2^{l-1} permutation cycles of length two; second, observe that if $l > 0$, then $\lambda^{2^{l-1}} = \nu^{p \cdot 2^{l-1}} = \nu^{2^{l-1}}$, since p is an odd number (such that $0 < p < 2^l$). Moreover, notice that $\kappa^{2^{k-l}}|_{dom(\lambda)} = \nu$, that implies that $\kappa^{2^{k-1}}|_{dom(\lambda)} = \nu^{2^{l-1}}$. Thus, the permutation cycles of length two generated from κ and λ, restricted to $dom(\lambda)$ are the same, which implies that commutative combinations built (according to Def. 6.4) regarding to the elements in $dom(\lambda)$ are the same.

Example 6.17. *Consider $\kappa = (a\, b\, c\, d\, e\, f\, g\, h)$ and $\lambda = (a\, g\, e\, c)$. Notice that $\kappa^2 = (a\, c\, e\, g)(b\, d\, f\, h)$ and λ is a 3-factor of $\nu = (a\, c\, e\, g)$. Then $\lambda^2 = \nu^{3 \cdot 2} = \nu^2 = (a\, e)(c\, g)$. Also, notice that the unitary* `epc`*'s built from λ and ν are the same.*

Definition 6.10 (Permutation cycles in the top of Π_X). *Let Π_X be the set of permutations for C FP equations on the variable X in a C FP problem. A permutation cycle $\kappa \in \pi \in \Pi_X$ is in the top of Π_X, whenever for all atoms $a \in dom(\kappa)$ and all $\pi' \in \Pi_X$, if $a \in dom(\pi')$, and a is an element in a permutation cycle λ in π', then there exists a natural m such that the permutation cycle of the element a in π^{2^m}, say ν, is a factor of the permutation cycle λ.*

Example 6.18. *Consider the permutations $\pi_1 = (a\,b\,c\,d\,e\,f\,g\,h)$, $\pi_2 = (a\,g\,e\,c)(b\,f)$ and $\pi_3 = (a\,e)(c\,g)(d\,h)$. The permutation cycle π_1 is in the top of the set of permutations; indeed, notice that all permutation cycles in all permutations appear as a factor in powers of two of π_1: $\pi_1^0 = (a\,b\,c\,d\,e\,f\,g\,h)$; $\pi_1^2 = (a\,c\,e\,g)(b\,d\,f\,h)$; $\pi_1^4 = (a\,e)(c\,g)(b\,f)(d\,h)$; $\pi_1^8 = (a)(e)(c)(g)(b)(f)(d)(h)$.*

Theorem 6.8 (Atoms of interest in C FP problems on a variable). *Let Π_X be the set of permutations for C FP equations on the variable X in a C FP problem. Only the set of atoms in the domain of permutation cycles in the top of Π_X might occur in solutions of all C FP equations on X.*

Proof. Only atoms that are in permutation cycles of length a power of two in all permutations $\pi \in \Pi_X$ might occur in solutions of all C FP equations on X. Suppose a is an atom that only occurs in permutation cycles of length a power of two for all $\pi \in \Pi_X$ and let κ be a permutation cycle in Π_X of maximal length, say 2^k, with $a \in dom(\kappa)$. Suppose λ is a permutation cycle in ϕ, for some $\phi \in \Pi_X$, with $a \in dom(\lambda)$ and let 2^l be the length of λ. Only if λ is a factor of a permutation cycle in $\pi^{2^{k-l}}$, say ν such that $\nu^p = \lambda$, the `epc`'s built from λ (and from κ) will maintain the invariants required, restricted to the atoms in $dom(\lambda)$, that is for an `epc` built from λ of the form $(A_0\ \ldots\ A_{2^m-1})$, where $m \leq l$, $\phi(A_i) \approx_C A_{i+1}$ and $\phi^{2^{l-m}}(A_i) \approx_C A_i$, where $i+1$ reads modulo 2^m. This also holds for λ. Hence, since ν is a p-factor of λ (and also, $\pi^{2^{k-l}}|_{dom(\lambda)} = \nu$), one has that $\nu^p(A_i) \approx_C A_{i+1}$ and $\nu^{p\cdot 2^{l-m}}(A_i) \approx_C A_i$. If the `epc` is of length two, that is it is of the form $(A_0\,A_1)$, one has $m = 1$ and $\nu^{p\cdot 2^{l-1}}(A_i) \approx_C A_i$, for $i = 0, 1$, and since p is odd, this implies that $\nu^{2^{l-1}}(A_i) \approx_C A_i$, for $i = 0, 1$. This condition also holds for π, since $(\pi^{2^{k-l}}|_{dom(\nu)})^{2^{l-1}} = (\nu)^{2^{l-1}}$; hence, $\pi^{2^{k-1}}(A_i) = A_{i+1}$, for $i = 0, 1$. If κ is not a permutation cycle in the top of Π_X, then there exists some permutation cycle $\lambda \in \phi \in \Pi_X$, such that $a \in dom(\kappa) \cap dom(\lambda)$, 2^l is the length of λ, but the permutation cycle of length 2^l in $\kappa^{2^{k-l}}$, say ν, such that $a \in dom(\nu)$ is not a factor of λ. Thus, since $\nu^{2^{l-1}} \neq \lambda^{2^{l-1}}$ atoms in the domains of ν and λ cannot be combined uniformly to build common solutions for κ and λ (i.e., for π and ψ).

To finish it is showed how a common solution can be built when κ is in the top of Π_X. Suppose that (A) is a unitary `epc` built from λ by successive applications of case 3.a.i. of

Def. 6.4 halving in each step the length of the epc. One has that $\lambda(A) = A$. It is possible to generate an epc for κ of the form $(A\,\kappa(A)\,\kappa^2(A)\,\ldots\,\kappa^{2^{k-l}-1}(A))$. From this epc it is possible to build a unitary epc by successive applications of case 3.a.i. of Def. 6.4, first obtaining $(A\star_1\kappa^{2^{K-l-1}}(A)\;\kappa(A)\star_1\kappa^{2^{k-l-1}+1}(A)\ldots\;\kappa^{2^{k-l-1}-1}(A)\star_1\kappa^{2^{k-l}-1}(A))$, and so on until a unitary epc of the form $((\cdots((A\star_1 B_1)\star_2 B_2)\cdots)\star_{k-l} B_{k-l})$ is obtained where the B_i's, for $1 \leq i \leq k-l$ are adequate combinations of the terms $\kappa(A),\ldots\kappa^{2^{k-l}-1}(A)$ according to the constructions of epc's. From this epc one has the solution for $\pi.X \approx_? X$ of the form $\langle\emptyset, \{X/(\cdots((A\star_1 B_1)\star_2 B_2)\cdots)\star_{k-l} B_{k-l}\}\rangle$, where $\star_j$, for $j = 1,\ldots,l$ are commutative symbols. Using the unitary cycle (A) for λ and cases 1 and 3.a.ii of Def. 6.4 one can generate the unitary epc $((\cdots((A\star_1 Y_1)\star_2 Y_2)\cdots)\star_{k-l} Y_{k-l})$ which gives the solution $\langle\nabla, \{X/(\cdots((A\star_1 Y_1)\star_2 Y_2)\cdots)\star_{k-l} Y_{k-l}\}\rangle$ for λ, where $\nabla = \{dom(\lambda)\#Y_j | 1 \leq j \leq l\}$. The C-unification problem $\langle\nabla, \mathcal{X}, X \approx_? (\cdots((A\star_1 B_1)\star_2 B_2)\cdots)\star_{k-l} B_{k-l}, X \approx_? (\cdots((A\star_1 Y_1)\star_2 Y_2)\cdots)\star_{k-l} Y_{k-l}\}\rangle$ unifies with solution $\langle\emptyset, \{X/(\cdots((A\star_1 B_1)\star_2 B_2)\cdots)\star_{k-l} B_{k-l}\}\rangle$ which is a common solution for π and ϕ. □

Example 6.19. *(Continuing example 6.18) First, notice that the permutation cycle $\pi_1 = (a\,b\,c\,d\,e\,f\,g\,h)$ is not in the top of $(a\,d\,e\,b\,g\,h\,c\,f)$; also, π_1 is neither in the top of $(a\,b\,c\,d)$ nor in the top of $(a\,i)$. Since π_1 is not a factor of π_2, solutions generated from the* epc $(\bar{a}\,\bar{d}\,\bar{e}\,\bar{b}\,\bar{g}\,\bar{h}\,\bar{c}\,\bar{f})$ *might not be solutions built for π_1; for instance, consider the unitary* epc *built for π_2, $(((\bar{a}\star\bar{g})\diamond(\bar{e}\star\bar{c}))\oplus((\bar{d}\star\bar{h})\diamond(\bar{b}\star\bar{f}))$, which is not a solution for π_1, since not $\pi_1((\bar{a}\star\bar{g})\diamond(\bar{e}\star\bar{c}))\approx_C(\bar{d}\star\bar{h})\diamond(\bar{b}\star\bar{f})$. Also, for the* epc $(\bar{a}\,\bar{b}\,\bar{c}\,\bar{d})$: *the permutation cycles in π_1^2 are $(a\,c\,e\,g)$ and $(b\,d\,f\,h)$, which give different solutions. For $(\bar{a}\,\bar{i})$, the permutation cycle $(a\,e)$ in π_i^4 produces different solutions.*

Now consider solutions of C FP equations $\pi_i.X \approx_? X$, for $i = 1,2,3$, where Π_X consists of the permutations $\pi_1 = (a\,b\,c\,d\,e\,f\,g\,h)$, $\pi_2 = (a\,g\,e\,c)(b\,f)$ and $\pi_3 = (a\,e)(c\,g)(d\,h)$. In this case, it has been seen (Ex. 6.18) that π_1 is a permutation cycle in the top of Π_X. Among the solutions generated for $\pi_i.X \approx_? X$, for $i = 1,2,3$ through epc*'s one has, respectively:*
$\langle\nabla_1, \{X/s_1 = ((\bar{a}+\bar{e})\star(\bar{c}+\bar{g}))\oplus((\bar{b}+\bar{f})\star(\bar{d}+\bar{h})\}\rangle$,
$\langle\nabla_2, \{X/s_2 = ((\bar{a}+\bar{e})\star(\bar{c}+\bar{g}))\oplus((\bar{b}+\bar{f})\star Y\}\rangle$ *and*
$\langle\nabla_3, \{X/s_3 = ((\bar{a}+\bar{e})\star(\bar{c}+\bar{g}))\oplus(Z\star(\bar{d}+\bar{h})\}\rangle$,
where $\nabla_1 = \emptyset$, $\nabla_2 = \{a\#Y, b\#Y, c\#Y, e\#Y, f\#Y, g\#Y\}$ and
$\nabla_3 = \{a\#Z, c\#Z, d\#Z, e\#Z, g\#Z, h\#Z\}$*, and the symbols $\oplus, \star$ and $+$ are commutative. The C-unification problem $\langle\nabla_1\cup\nabla_2\cup\nabla_3, \{X \approx_? s_1, X \approx_? s_2, X \approx_? s_3\}\rangle$ has solution $\{X/s_1, Y/\bar{d}+\bar{h}, Z/\bar{b}+\bar{f}\}$ with the respective freshness constraints; thus, restricting this solution to the freshness constraints on X one has the common solution $\langle\emptyset, \{X/s_1\}\rangle$.*

Example 6.20. *(Continuing Ex. 6.15) As it is seen in Ex. 6.15, the C FP equations $(a\,b).Y \approx_? Y$ and $(a\,d\,c\,b)(a'\,b').Y \approx_? Y$ in the C FP problem $\mathcal{Q}_2$, have no possible combinatorial solution with occurrences of the atoms a' or b'. By Thm. 6.8, cycles $(a\,b)$*

and $(a\,d\,c\,b)$ *do not give rise to possible combinatorial solutions for both C FP equations. Hence, there is no feasible combinatorial solution for this C FP problem. Therefore, the unique possible solution for* $\mathcal{Q}_2$ *is* $\langle \{a', b' \# X, a, b, c, d, a', b' \# Y\}, \{X/(a\,c)(a'\,b').Y\} \rangle$.

Remark 6.19. *The greedy generation algorithm can then be improved by generating solutions only for the atoms in permutation cycles in the top of* Π_X.

Chapter 7

Nominal A, C and AC-unification and matching

This Chapter describes an extension of the nominal C-unification/matching algorithm (Chap. 5), by adding A and AC function symbols in the signature. A nominal C and AC-unification algorithm is obtained by extending the system of Fig. 5.1, with the addition of rule ($\approx_?$ **AC**) of Fig. 7.2. A nominal A, C and AC-matching algorithm is obtained by adding rule ($\approx_?$ **A**) of Fig. 7.1 to the derivation system, and instantiating the set $\mathcal{X}$ of protected variables with the set of *rhs* variables of the input problem.

In Sec. 7.1, it is presented the new rules ($\approx_?$ **A**) and ($\approx_?$ **AC**) (see Figs. 7.1 and 7.2) that simplifies equations with terms which are headed, respectively, by A and AC symbols. This rule-based approach is similar to the presented by Contejean [30], but differs from that strategy provided in Subsec. 2.1.2 for first-order AC-unification. In the former, a set of rules is applied to equations whose terms are headed by A, C and AC function symbols to obtain a finite set of new simpler equations, while the least translates the problem to the search of non-negative solutions for a set of Diophantine equations.

Additionally, in Sec. 7.2 a generator of solutions for FP problems with C and AC function symbols is described.

Definition 7.1 (Nominal A/C/AC-unification solution)**.** *A nominal A/C/AC-unification solution for a problem $\mathcal{P}$ is a pair $\langle \Delta, \delta \rangle$ such that the conditions of Def. 5.2, of Chap. 5 (Subsec. 5.1.1), are extended, replacing $\approx_{\alpha,C}$ by $\approx_{\{A,C,AC\}}$. The set of nominal A/C/AC-unification solutions for $\mathcal{P}$ is denoted by $\mathcal{U}_{AC}(\mathcal{P})$.*

7.1 Rules for nominal A, C and AC problems

In the nominal C and AC-unification and the nominal A, C and AC-matching algorithms presented in this section, the simplification rules ($\approx_?$ **A**) and ($\approx_?$ **AC**) use Def. 7.2 that

expresses an arbitrary way of bracketing and sorting of arguments w.r.t a given function symbol f. Properties of termination, preservation of solutions and completeness of these rules are presented by Lems. 7.2 to 7.4.

Definition 7.2. *Let s and f be, respectively, a term and a function symbol. If $n = \|s\|_f$, then $\langle\langle s_{(1)}, \ldots, s_{(n)}\rangle\rangle_f$ and $\langle\langle s_{(\overline{1})}, \ldots, s_{(\overline{n})}\rangle\rangle_f$ denote tuples that are constructed with the arguments of s w.r.t. f. In the former, the elements are in an arbitrary bracketing and sorted according to their occurrences in s, while in the latter the elements are in an arbitrary bracketing and sorting. If $n = 1$, both tuples are equal to $s_{(1)}$.*

Remark 7.1. *In Def. 7.2, the tuples $\langle\langle s_{(1)}, \ldots, s_{(n)}\rangle\rangle_f$ and $\langle\langle s_{(\overline{1})}, \ldots, s_{(\overline{n})}\rangle\rangle_f$ are built using the operator $s_{(i)_f}$ of selection of arguments specified as in Fig. 4.2 of Sec. 4.1. Notice that, according the definitions of the operators $\|t\|_f$ and $t_{(i)_f}$ (Figs. 4.1 and 4.2) one has that $\|fs\|_f = \|s\|_f$ and $(fs)_{(i)_f} = s_{(i)_f}$. Then, also $\|f\langle\langle s_{(1)}, ..., s_{(n)}\rangle\rangle_f\| = \|\langle\langle s_{(1)}, ..., s_{(n)}\rangle\rangle_f\|$ and $\|f\langle\langle s_{(\overline{1})}, ..., s_{(\overline{n})}\rangle\rangle_f\| = \|\langle\langle s_{(\overline{1})}, ..., s_{(\overline{n})}\rangle\rangle_f\|$. This fact will be used in the definition of rules of Figs. 7.1 and 7.2 to simplify the notation.*

Example 7.1. *Let f be an AC function symbol. For $s = f\langle \overline{a}, f\langle \overline{b}, \overline{c}\rangle\rangle$ and $t = f\langle f\langle \overline{a}, \overline{b}\rangle, f\langle \overline{c}, \overline{d}\rangle\rangle$*

a) $\langle\langle s_{(1)}, s_{(2)}, s_{(3)}\rangle\rangle_f$ can represent either: $\langle \overline{a}, \langle \overline{b}, \overline{c}\rangle\rangle$ or $\langle\langle \overline{a}, \overline{b}\rangle, \overline{c}\rangle$;

b) $\langle\langle t_{(1)}, t_{(2)}, t_{(3)}, t_{(4)}\rangle\rangle_f$ can represent either: $\langle\langle \overline{a}, \overline{b}\rangle, \langle \overline{c}, \overline{d}\rangle\rangle$; $\langle \overline{a}, \langle \overline{b}, \langle \overline{c}, \overline{d}\rangle\rangle\rangle$; $\langle\langle\langle \overline{a}, \overline{b}\rangle, \overline{c}\rangle, \overline{d}\rangle$; $\langle\langle \overline{a}, \langle \overline{b}, \overline{c}\rangle\rangle, \overline{d}\rangle$ or $\langle \overline{a}, \langle\langle \overline{b}, \overline{c}\rangle, \overline{d}\rangle\rangle$;

c) $\langle\langle s_{(\overline{1})}, s_{(\overline{2})}, s_{(\overline{3})}\rangle\rangle_f$ can represent either: $\langle \overline{a}, \langle \overline{b}, \overline{c}\rangle\rangle$; $\langle \overline{a}, \langle \overline{c}, \overline{b}\rangle\rangle$; $\langle \overline{b}, \langle \overline{a}, \overline{c}\rangle\rangle$; $\langle \overline{b}, \langle \overline{c}, \overline{a}\rangle\rangle$; $\langle \overline{c}, \langle \overline{a}, \overline{b}\rangle\rangle$; $\langle \overline{c}, \langle \overline{b}, \overline{a}\rangle\rangle$; $\langle\langle \overline{a}, \overline{b}\rangle, \overline{c}\rangle$; $\langle\langle \overline{a}, \overline{c}\rangle, \overline{b}\rangle$; $\langle\langle \overline{b}, \overline{a}\rangle, \overline{c}\rangle$; $\langle\langle \overline{b}, \overline{c}\rangle, \overline{a}\rangle$; $\langle\langle \overline{c}, \overline{a}\rangle, \overline{b}\rangle$ or $\langle\langle \overline{c}, \overline{b}\rangle, \overline{a}\rangle$.

Lemma 7.1. *For $n = \|s\|_f$, $|s| \geq |\langle\langle s_{(1)}, \ldots, s_{(n)}\rangle\rangle_f|$ and $|s| \geq |\langle\langle s_{(\overline{1})}, \ldots, s_{(\overline{n})}\rangle\rangle_f|$.*

Proof. The proof is by induction on the structure of s. The interesting cases are $s = \langle u, v\rangle$ and $s = f\,u$. In the former $|s| = |u| + |v| + 1$, which, by IH, is a value greater or equal to $|\langle\langle u_{(1)}, \ldots, u_{(k)}\rangle\rangle_f| + |\langle\langle v_{(1)}, \ldots, v_{(l)}\rangle\rangle_f| + 1 = |\langle\langle s_{(1)}, \ldots, s_{(n)}\rangle\rangle_f|$. For the latter, observe that $|f\,u| = |u| + 1$, which, by IH, is a value greater than $|\langle\langle u_{(1)}, \ldots, u_{(k)}\rangle\rangle_f|$. □

Example 7.2.

a) If $s = \|\langle \overline{a}, \langle \overline{b}, \overline{c}\rangle\rangle\|_f$, then $\|s\| = 5 = \|\langle\langle s_{(1)}, s_{(2)}, s_{(3)}\rangle\rangle_f\|_f = \|\langle\langle s_{(\overline{1})}, s_{(\overline{2})}, s_{(\overline{3})}\rangle\rangle_f\|_f$

b) If $s = \|f\langle \overline{a}, f\langle \overline{b}, \overline{c}\rangle\rangle\|_f$, then $\|s\| = 7 > 5 = \|\langle\langle s_{(1)}, s_{(2)}, s_{(3)}\rangle\rangle_f\|_f = \|\langle\langle s_{(\overline{1})}, s_{(\overline{2})}, s_{(\overline{3})}\rangle\rangle_f\|_f$

Observe that, rule ($\approx_?$ **A**) of Fig. 7.1 eliminates equations of the form $f_k^A\, s \approx_? f_k^A\, t$, selecting the arguments of s and t w.r.t. f_k^A and arranging them into tuples with arbitrary bracketing (via Def. 7.2), respectively, in the *lhs* and *rhs* of a new equation $\langle\langle s_{(1)}, \ldots, s_{(m)}\rangle\rangle_{f_k^A} \approx_? \langle\langle t_{(1)}, \ldots, t_{(n)}\rangle\rangle_{f_k^A}$.

$$(\approx_? \mathbf{A})\ \frac{\langle \nabla, \mathcal{X}, \sigma, P \uplus \{f_k^A\, s \approx_? f_k^A\, t\}\rangle \ m = \|s\|_{f_k^A},\ n = \|t\|_{f_k^A}}{\langle \nabla, \mathcal{X}, \sigma, P \cup \{\langle\langle s_{(1)}, \ldots, s_{(m)}\rangle\rangle_{f_k^A} \approx_? \langle\langle t_{(1)}, \ldots, t_{(n)}\rangle\rangle_{f_k^A}\}\rangle}$$

Figure 7.1: ($\approx_?$ **A**) for A function symbols

$$(\approx_? \mathbf{AC})\ \frac{\langle \nabla, \mathcal{X}, \sigma, P \uplus \{f_k^{AC}\, s \approx_? f_k^{AC}\, t\}\rangle \ m = \|s\|_{f_k^{AC}},\ n = \|t\|_{f_k^{AC}}}{\langle \nabla, \mathcal{X}, \sigma, P \cup \{\langle\langle s_{(1)}, \ldots, s_{(m)}\rangle\rangle_{f_k^{AC}} \approx_? \langle\langle t_{(\overline{1})}, \ldots, t_{(\overline{n})}\rangle\rangle_{f_k^{AC}}\}\rangle}$$

Figure 7.2: ($\approx_?$ **AC**) for AC function symbols

Similarly to rule ($\approx_?$ **A**), rule ($\approx_?$ **AC**) of Fig. 7.2 transforms equations of the form $f_k^{AC}\, s \approx_? f_k^{AC}\, t$ into $\langle\langle s_{(1)}, \ldots, s_{(m)}\rangle\rangle_{f_k^{AC}} \approx_? \langle\langle t_{(\overline{1})}, \ldots, t_{(\overline{n})}\rangle\rangle_{f_k^{AC}}$, selecting the arguments of s and t w.r.t. f_k^{AC} and arranging them into the tuples of the new equation. The difference is that the arguments of the *rhs* tuple are arranged in an arbitrary sorting. The relations $\Rightarrow_{(\approx_?\mathbf{A})}$ and $\Rightarrow_{(\approx_?\mathbf{AC})}$ denote, respectively, reductions of quadruples by an application of rules ($\approx_?$ **A**) and ($\approx_?$ **AC**).

Example 7.3. *Let f be an A function symbol. It is considered the nominal A, C and AC-matching algorithm applied to*

$$\mathcal{P} = \langle \emptyset, \emptyset, id, \{[a]f\,\langle X, f\,\langle Y, Z\rangle\rangle \approx_? [b]f\,\langle f\,\langle \overline{b}, \overline{c}\rangle, f\,\langle \overline{d}, \overline{e}\rangle\rangle\}\rangle$$

The problem is initially simplified, by rule ($\approx_?$ **[ab]**)*, to*

$$\langle \emptyset, \emptyset, id, \{f\,\langle X, f\,\langle Y, Z\rangle\rangle \approx_? f\,\langle f\,\langle \overline{a}, \overline{c}\rangle, f\,\langle \overline{d}, \overline{e}\rangle\rangle,\ a \#_? f\,\langle f\,\langle \overline{b}, \overline{c}\rangle, f\,\langle \overline{d}, \overline{e}\rangle\rangle\}\rangle.$$

Let s and t be equal, respectively, to $\langle X, f\,\langle Y, Z\rangle\rangle$ and $\langle f\,\langle \overline{a}, \overline{c}\rangle, f\,\langle \overline{d}, \overline{e}\rangle\rangle$. Applying rule ($\approx_?$ **A**)*, one obtains a family of quadruples in the form*

$$\langle \emptyset, \emptyset, id, \{\langle\langle s_{(1)}, s_{(2)}, s_{(3)}\rangle\rangle_f \approx_? \langle\langle t_{(1)}, t_{(2)}, t_{(3)}, t_{(4)}\rangle\rangle_f,\ a \#_? f\,\langle f\,\langle \overline{b}, \overline{c}\rangle, f\,\langle \overline{d}, \overline{e}\rangle\rangle\}\rangle.$$

Then, the possible cases of bracketing in $\langle\langle s_{(1)}, s_{(2)}, s_{(3)}\rangle\rangle_f$ and $\langle\langle t_{(1)}, t_{(2)}, t_{(3)}, t_{(4)}\rangle\rangle_f$ generate ten leaves that are labelled form 1.1 to 1.5, and from 2.1 to 2.5, as presented below.

1.1. $\langle\langle s_{(1)}, s_{(2)}, s_{(3)}\rangle\rangle_f = \langle X, \langle Y, Z\rangle\rangle$ *and* $\langle\langle t_{(1)}, t_{(2)}, t_{(3)}, t_{(4)}\rangle\rangle_f = \langle \overline{a}, \langle \overline{c}, \langle \overline{d}, \overline{e}\rangle\rangle\rangle$

$$\begin{aligned}
&\Rightarrow_{(\approx_?\mathbf{pair})} \langle \emptyset, \emptyset, id, \{\boxed{X \approx_? \overline{a}},\ \langle Y, Z\rangle \approx_? \langle \overline{c}, \langle \overline{d}, \overline{e}\rangle\rangle,\ a \#_? f\,\langle f\,\langle \overline{b}, \overline{c}\rangle, f\,\langle \overline{d}, \overline{e}\rangle\rangle\}\rangle \\
&\Rightarrow_{(\approx_?\mathbf{inst})} \langle \emptyset, \emptyset, \{X/\overline{a}\}, \{\boxed{\langle Y, Z\rangle \approx_? \langle \overline{c}, \langle \overline{d}, \overline{e}\rangle\rangle},\ a \#_? f\,\langle f\,\langle \overline{b}, \overline{c}\rangle, f\,\langle \overline{d}, \overline{e}\rangle\rangle\}\rangle \\
&\Rightarrow_{(\approx_?\mathbf{pair})} \langle \emptyset, \emptyset, \{X/\overline{a}\}, \{\boxed{Y \approx_? \overline{c}},\ Z \approx_? \langle \overline{d}, \overline{e}\rangle,\ a \#_? f\,\langle f\,\langle \overline{b}, \overline{c}\rangle, f\,\langle \overline{d}, \overline{e}\rangle\rangle\}\rangle \\
&\Rightarrow_{(\approx_?\mathbf{inst})} \langle \emptyset, \emptyset, \{X/\overline{a}, Y/\overline{c}\}, \{\boxed{Z \approx_? \langle \overline{d}, \overline{e}\rangle},\ a \#_? f\,\langle f\,\langle \overline{b}, \overline{c}\rangle, f\,\langle \overline{d}, \overline{e}\rangle\rangle\}\rangle
\end{aligned}$$

$$\Rightarrow_{(\approx_? \mathbf{inst})} \langle \emptyset, \emptyset, \{X/\overline{a}, Y/\overline{c}, Z/\langle \overline{d}, \overline{e}\rangle\}, \{\boxed{a \#_? f\, \langle f\, \langle \overline{b}, \overline{c}\rangle, f\, \langle \overline{d}, \overline{e}\rangle\rangle}\}\rangle$$

$$\ldots$$

$$\Rightarrow_{(\#_? \mathbf{atom})} \langle \emptyset, \emptyset, \{X/\overline{a}, Y/\overline{c}, Z/\langle \overline{d}, \overline{e}\rangle\}, \emptyset\rangle$$

1.2. $\langle\langle s_{(1)}, s_{(2)}, s_{(3)}\rangle\rangle_f = \langle X, \langle Y, Z\rangle\rangle$ *and* $\langle\langle t_{(1)}, t_{(2)}, t_{(3)}, t_{(4)}\rangle\rangle_f = \langle\langle\langle \overline{a}, \overline{c}\rangle, \overline{d}\rangle, \overline{e}\rangle$

$$\Rightarrow_{(\approx_? \mathbf{pair})} \langle \emptyset, \emptyset, id, \{\boxed{X \approx_? \langle\langle \overline{a}, \overline{c}\rangle, \overline{d}\rangle},\ \langle Y, Z\rangle \approx_? \overline{e},\ a \#_? f\, \langle f\, \langle \overline{b}, \overline{c}\rangle, f\, \langle \overline{d}, \overline{e}\rangle\rangle\}\rangle$$

$$\Rightarrow_{(\approx_? \mathbf{inst})} \langle \emptyset, \emptyset, \{X/\langle\langle \overline{a}, \overline{c}\rangle, \overline{d}\rangle\}, \{\langle Y, Z\rangle \approx_? \overline{e},\ a \#_? f\, \langle f\, \langle \overline{b}, \overline{c}\rangle, f\, \langle \overline{d}, \overline{e}\rangle\rangle\}\rangle$$

$$\Rightarrow \bot$$

1.3. $\langle\langle s_{(1)}, s_{(2)}, s_{(3)}\rangle\rangle_f = \langle X, \langle Y, Z\rangle\rangle$ *and* $\langle\langle t_{(1)}, t_{(2)}, t_{(3)}, t_{(4)}\rangle\rangle_f = \langle\langle \overline{a}, \overline{c}\rangle, \langle \overline{d}, \overline{e}\rangle\rangle$

$$\Rightarrow_{(\approx_? \mathbf{pair})} \langle \emptyset, \emptyset, id, \{\boxed{X \approx_? \langle \overline{a}, \overline{c}\rangle},\ \langle Y, Z\rangle \approx_? \langle \overline{d}, \overline{e}\rangle,\ a \#_? f\, \langle f\, \langle \overline{b}, \overline{c}\rangle, f\, \langle \overline{d}, \overline{e}\rangle\rangle\}\rangle$$

$$\Rightarrow_{(\approx_? \mathbf{inst})} \langle \emptyset, \emptyset, \{X/\langle \overline{a}, \overline{c}\rangle\}, \{\boxed{\langle Y, Z\rangle \approx_? \langle \overline{d}, \overline{e}\rangle},\ a \#_? f\, \langle f\, \langle \overline{b}, \overline{c}\rangle, f\, \langle \overline{d}, \overline{e}\rangle\rangle\}\rangle$$

$$\Rightarrow_{(\approx_? \mathbf{pair})} \langle \emptyset, \emptyset, \{X/\langle \overline{a}, \overline{c}\rangle\}, \{\boxed{Y \approx_? \overline{d}},\ Z \approx_? \overline{e},\ a \#_? f\, \langle f\, \langle \overline{b}, \overline{c}\rangle, f\, \langle \overline{d}, \overline{e}\rangle\rangle\}\rangle$$

$$\Rightarrow_{(\approx_? \mathbf{inst})} \langle \emptyset, \emptyset, \{X/\langle \overline{a}, \overline{c}\rangle, Y/\overline{d}\}, \{\boxed{Z \approx_? \overline{e}},\ a \#_? f\, \langle f\, \langle \overline{b}, \overline{c}\rangle, f\, \langle \overline{d}, \overline{e}\rangle\rangle\}\rangle$$

$$\Rightarrow_{(\approx_? \mathbf{inst})} \langle \emptyset, \emptyset, \{X/\langle \overline{a}, \overline{c}\rangle, Y/\overline{d}, Z/\overline{e}\}, \{\boxed{a \#_? f\, \langle f\, \langle \overline{b}, \overline{c}\rangle, f\, \langle \overline{d}, \overline{e}\rangle\rangle}\}\rangle$$

$$\ldots$$

$$\Rightarrow_{(\#_? \mathbf{atom})} \langle \emptyset, \emptyset, \{X/\langle \overline{a}, \overline{c}\rangle, Y/\overline{d}, Z/\overline{e}\}, \emptyset\rangle$$

1.4. $\langle\langle s_{(1)}, s_{(2)}, s_{(3)}\rangle\rangle_f = \langle X, \langle Y, Z\rangle\rangle$ *and* $\langle\langle t_{(1)}, t_{(2)}, t_{(3)}, t_{(4)}\rangle\rangle_f = \langle\langle \overline{a}, \langle \overline{c}, \overline{d}\rangle\rangle, \overline{e}\rangle$

$$\Rightarrow_{(\approx_? \mathbf{pair})} \ldots \Rightarrow \bot$$

1.5. $\langle\langle s_{(1)}, s_{(2)}, s_{(3)}\rangle\rangle_f = \langle X, \langle Y, Z\rangle\rangle$ *and* $\langle\langle t_{(1)}, t_{(2)}, t_{(3)}, t_{(4)}\rangle\rangle_f = \langle \overline{a}, \langle\langle \overline{c}, \overline{d}\rangle, \overline{e}\rangle\rangle$

$$\Rightarrow_{(\approx_? \mathbf{pair})} \ldots \Rightarrow_{(\#_? \mathbf{atom})} \langle \emptyset, \emptyset, \{X/\overline{a}, Y/\langle \overline{c}, \overline{d}\rangle, Z/\overline{e}\}, \emptyset\rangle$$

2.1. $\langle\langle s_{(1)}, s_{(2)}, s_{(3)}\rangle\rangle_f = \langle\langle X, Y\rangle, Z\rangle$ *and* $\langle\langle t_{(1)}, t_{(2)}, t_{(3)}, t_{(4)}\rangle\rangle_f = \langle \overline{a}, \langle \overline{c}, \langle \overline{d}, \overline{e}\rangle\rangle\rangle$

$$\Rightarrow_{(\approx_? \mathbf{pair})} \ldots \Rightarrow \bot$$

2.2. $\langle\langle s_{(1)}, s_{(2)}, s_{(3)}\rangle\rangle_f = \langle\langle X, Y\rangle, Z\rangle$ *and* $\langle\langle t_{(1)}, t_{(2)}, t_{(3)}, t_{(4)}\rangle\rangle_f = \langle\langle\langle \overline{a}, \overline{c}\rangle, \overline{d}\rangle, \overline{e}\rangle$

$$\Rightarrow_{(\approx_? \mathbf{pair})} \ldots \Rightarrow_{(\#_? \mathbf{atom})} \langle \emptyset, \emptyset, \{X/\langle \overline{a}, \overline{c}\rangle, Y/\overline{d}, Z/\overline{e}\}, \emptyset\rangle$$

2.3. $\langle\langle s_{(1)}, s_{(2)}, s_{(3)}\rangle\rangle_f = \langle\langle X, Y\rangle, Z\rangle$ *and* $\langle\langle t_{(1)}, t_{(2)}, t_{(3)}, t_{(4)}\rangle\rangle_f = \langle\langle \overline{a}, \overline{c}\rangle, \langle \overline{d}, \overline{e}\rangle\rangle$

$$\Rightarrow_{(\approx_? \mathbf{pair})} \ldots \Rightarrow_{(\#_? \mathbf{atom})} \langle \emptyset, \emptyset, \{X/\overline{a}, Y/\overline{c}, Z/\langle \overline{d}, \overline{e}\rangle\}, \emptyset\rangle$$

2.4. $\langle\langle s_{(1)}, s_{(2)}, s_{(3)}\rangle\rangle_f = \langle\langle X, Y\rangle, Z\rangle$ *and* $\langle\langle t_{(1)}, t_{(2)}, t_{(3)}, t_{(4)}\rangle\rangle_f = \langle\langle \overline{a}, \langle \overline{c}, \overline{d}\rangle\rangle, \overline{e}\rangle$

$\Rightarrow_{(\approx_?\mathbf{pair})} \cdots \Rightarrow_{(\#_?\,\mathbf{atom})} \langle \emptyset, \emptyset, \{X/\overline{a}, Y/\langle \overline{c}, \overline{d}\rangle, Z/\overline{e}\}, \emptyset\rangle$

2.5. $\langle\langle s_{(1)}, s_{(2)}, s_{(3)}\rangle\rangle_f = \langle\langle X, Y\rangle, Z\rangle$ *and* $\langle\langle t_{(1)}, t_{(2)}, t_{(3)}, t_{(4)}\rangle\rangle_f = \langle \overline{a}, \langle\langle \overline{c}, \overline{d}\rangle, \overline{e}\rangle\rangle$

$\Rightarrow_{(\approx_?\mathbf{pair})} \cdots \Rightarrow \bot$

Failure cases are represented by $\bot$ *and are generated only when an atom term in the* rhs *of an equation is associated is associated with a pair in the* lhs. *The corresponding solutions obtained in each successful leaf are summarised in the following table:*

Leaf label	Corresponding solution			
1.1	$\nabla = \emptyset$	$\sigma = \{X/\overline{a},$	$Y/\overline{c},$	$Z/\langle \overline{d}, \overline{e}\rangle\}$
1.3	$\nabla = \emptyset$	$\sigma = \{X/\langle \overline{a}, \overline{c}\rangle,$	$Y/\overline{d},$	$Z/\overline{e}\}$
1.5	$\nabla = \emptyset$	$\sigma = \{X/\overline{a},$	$Y/\langle \overline{c}, \overline{d}\rangle,$	$Z/\overline{e}\}$
2.2	$\nabla = \emptyset$	$\sigma = \{X/\langle \overline{a}, \overline{c}\rangle,$	$Y/\overline{d},$	$Z/\overline{e}\}$
2.3	$\nabla = \emptyset$	$\sigma = \{X/\overline{a},$	$Y/\overline{c},$	$Z/\langle \overline{d}, \overline{e}\rangle\}$
2.4	$\nabla = \emptyset$	$\sigma = \{X/\overline{a},$	$Y/\langle \overline{c}, \overline{d}\rangle,$	$Z/\overline{e}\}$

Then, the set of solutions provided by the algorithm is given by:

$$\left\{ \begin{array}{l} \langle \emptyset, \{X/\overline{a},\ Y/\overline{c},\ Z/\langle \overline{d}, \overline{e}\rangle\}\rangle, \\ \langle \emptyset, \{X/\langle \overline{a}, \overline{c}\rangle,\ Y/\overline{d},\ Z/\overline{e}\}\rangle, \\ \langle \emptyset, \{X/\overline{a},\ Y/\langle \overline{c}, \overline{d}\rangle,\ Z/\overline{e}\}\rangle \end{array} \right\}$$

Example 7.4. *Let* f *be an AC function symbol. The nominal C and AC unification algorithm applied to*

$$\mathcal{P} = \langle \emptyset, \emptyset, id, \{[a]f\,\langle X, Z\rangle \approx_? [b]f\,\langle f\,\langle X, Y\rangle, \overline{b}\rangle\}\rangle$$

generates twelve leaves: four cases of failure ($\bot$*) and the following eight successful leaves:*

1. $\langle\{a\#X, a\#Y\}, \emptyset, \{Z/\langle (a\,b).Y, \overline{a}\rangle\}, \{(a\,b).X \approx_? X\}\rangle$;
2. $\langle\{a\#X, a\#Y\}, \emptyset, \{Z/\langle \overline{a}, (a\,b).Y\rangle\}, \{(a\,b).X \approx_? X\}\rangle$;
3. $\langle\{a\#X, a\#Y\}, \emptyset, \{X/(a\,b).Y,\ Z/\langle (a\,b)(a\,b).Y, \overline{a}\rangle\}, \emptyset\rangle$;
4. $\langle\{a\#X, a\#Y\}, \emptyset, \{X/(a\,b).Y,\ Z/\langle \overline{a}, (a\,b)(a\,b).Y\rangle\}, \emptyset\rangle$;
5. $\langle\{a\#X, a\#Y\}, \emptyset, \{X/\overline{a},\ Z/\langle \overline{b}, (a\,b).Y\rangle\}, \emptyset\rangle$;
6. $\langle\{a\#X, a\#Y\}, \emptyset, \{X/\overline{a},\ Z/\langle (a\,b).Y, \overline{b}\rangle\}, \emptyset\rangle$;
7. $\langle\{a\#X, a\#Y\}, \emptyset, \{X/\langle (a\,b).Y, \overline{a}\rangle,\ Z/\langle (a\,b)(a\,b).Y, \overline{b}\rangle\}, \emptyset\rangle$;
8. $\langle\{a\#X, a\#Y\}, \emptyset, \{X/\langle \overline{a}, (a\,b).Y\rangle,\ Z/\langle \overline{b}, (a\,b)(a\,b).Y\rangle\}, \emptyset\rangle$.

The derivation for the input problem $\mathcal{P}$ *is described below. Some derivation branches are omitted:*

$$\mathcal{P} = \langle \emptyset, \emptyset, id, \{ [a]f\,\langle X, Z\rangle \approx_? [b]f\,\langle f\,\langle X, Y\rangle, \bar{b}\rangle \}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{[ab]})} \langle \emptyset, \emptyset, id, \{ f\,\langle X, Z\rangle \approx_? f\,\langle f\,\langle (a\,b).X, (a\,b).Y\rangle, \overline{a}\rangle,\ a \#_? f\,\langle f\,\langle X, Y\rangle, \bar{b}\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{AC})} \langle \emptyset, \emptyset, id, \{ \langle X, Z\rangle \approx_? \langle\langle s_{(\overline{1})}, s_{(\overline{2})}, s_{(\overline{3})}\rangle\rangle_f,\ a \#_? f\,\langle f\,\langle X, Y\rangle, \bar{b}\rangle\}\rangle$$

1. $\langle\langle s_{(1)}, s_{(2)}\rangle\rangle_f = \langle X, Z\rangle$ *and* $\langle\langle t_{(\overline{1})}, t_{(\overline{2})}, t_{(\overline{3})}\rangle\rangle_f = \langle (a\,b).X, \langle (a\,b).Y, \overline{a}\rangle\rangle$

$$\Rightarrow_{(\approx_?\mathbf{pair})} \langle \emptyset, \emptyset, id, \{ X \approx_? (a\,b).X,\ Z \approx_? \langle (a\,b).Y, \overline{a}\rangle,\ a \#_? f\,\langle f\,\langle X, Y\rangle, \bar{b}\rangle\}\rangle$$
$$\Rightarrow_{(\approx_?\mathbf{inv})} \langle \emptyset, \emptyset, id, \{(a\,b).X \approx_? X,\ Z \approx_? \langle (a\,b).Y, \overline{a}\rangle,\ a \#_? f\,\langle f\,\langle X, Y\rangle, \bar{b}\rangle\}\rangle$$
$$\Rightarrow_{(\approx_?\mathbf{inst})} \langle \emptyset, \emptyset, \{Z/\langle (a\,b).Y, \overline{a}\rangle\}, \{(a\,b).X \approx_? X,\ a \#_? f\,\langle f\,\langle X, Y\rangle, \bar{b}\rangle\}\rangle$$
$$\Rightarrow_{(\#_?\ldots)} \langle \{a\#X, a\#Y\}, \emptyset, \{Z/\langle (a\,b).Y, \overline{a}\rangle\}, \{(a\,b).X \approx_? X\}\rangle$$

2. $\langle\langle s_{(1)}, s_{(2)}\rangle\rangle_f = \langle X, Z\rangle$ *and* $\langle\langle t_{(\overline{1})}, t_{(\overline{2})}, t_{(\overline{3})}\rangle\rangle_f = \langle (a\,b).X, \langle \overline{a}, (a\,b).Y\rangle\rangle$

$$\Rightarrow_{(\approx_?\mathbf{pair})} \langle \emptyset, \emptyset, id, \{ X \approx_? (a\,b).X,\ Z \approx_? \langle \overline{a}, (a\,b).Y\rangle,\ a \#_? f\,\langle f\,\langle X, Y\rangle, \bar{b}\rangle\}\rangle$$
$$\Rightarrow_{(\approx_?\mathbf{inv})} \langle \emptyset, \emptyset, id, \{(a\,b).X \approx_? X,\ Z \approx_? \langle \overline{a}, (a\,b).Y\rangle,\ a \#_? f\,\langle f\,\langle X, Y\rangle, \bar{b}\rangle\}\rangle$$
$$\Rightarrow_{(\approx_?\mathbf{inst})} \langle \emptyset, \emptyset, \{Z/\langle \overline{a}, (a\,b).Y\rangle\}, \{(a\,b).X \approx_? X,\ a \#_? f\,\langle f\,\langle X, Y\rangle, \bar{b}\rangle\}\rangle$$
$$\Rightarrow_{(\#_?\ldots)} \langle \{a\#X, a\#Y\}, \emptyset, \{Z/\langle \overline{a}, (a\,b).Y\rangle\}, \{(a\,b).X \approx_? X\}\rangle$$

...

8. $\langle\langle s_{(1)}, s_{(2)}\rangle\rangle_f = \langle X, Z\rangle$ *and* $\langle\langle t_{(\overline{1})}, t_{(\overline{2})}, t_{(\overline{3})}\rangle\rangle_f = \langle\langle \overline{a}, (a\,b).Y\rangle, (a\,b).X\rangle$

$$\Rightarrow_{(\approx_?\mathbf{pair})} \ldots \Rightarrow \langle \{a\#X, a\#Y\}, \emptyset, \{X/\langle \overline{a}, (a\,b).Y\rangle, Z/\langle \bar{b}, (a\,b)(a\,b).Y\rangle\}, \emptyset\rangle$$

9. $\langle\langle s_{(1)}, s_{(2)}\rangle\rangle_f = \langle X, Z\rangle$ *and* $\langle\langle t_{(\overline{1})}, t_{(\overline{2})}, t_{(\overline{3})}\rangle\rangle_f = \langle\langle (a\,b).X, (a\,b).Y\rangle, \overline{a}\rangle$; $\langle\langle (a\,b).X, \overline{a}\rangle, (a\,b).Y\rangle$; $\langle\langle (a\,b).Y, (a\,b).X\rangle, \overline{a}\rangle$; *or* $\langle\langle \overline{a}, (a\,b).X\rangle, (a\,b).Y\rangle$

$$\Rightarrow_{(\approx_?\mathbf{pair})} \ldots \Rightarrow \bot$$

Notice that:

- *If* t *is such that* $\|t\|_f = 3$, *then* $\langle\langle t_{(\overline{1})}, t_{(\overline{2})}, t_{(\overline{3})}\rangle\rangle_f$ *represents* 12 *different tuples:* $\langle (a\,b).X, \langle (a\,b).Y, \overline{a}\rangle\rangle$; $\langle (a\,b).X, \langle \overline{a}, (a\,b).Y\rangle\rangle$; $\langle (a\,b).Y, \langle (a\,b).X, \overline{a}\rangle\rangle$; $\langle (a\,b).Y, \langle \overline{a}, (a\,b).X\rangle\rangle$; $\langle \overline{a}, \langle (a\,b).X, (a\,b).Y\rangle\rangle$; $\langle \overline{a}, \langle (a\,b).Y, (a\,b).X\rangle\rangle$; $\langle\langle (a\,b).X, (a\,b).Y\rangle, \overline{a}\rangle$; $\langle\langle (a\,b).X, \overline{a}\rangle, (a\,b).Y\rangle$; $\langle\langle (a\,b).Y, (a\,b).X\rangle, \overline{a}\rangle$; $\langle\langle (a\,b).Y, \overline{a}\rangle, (a\,b).X\rangle$; $\langle\langle \overline{a}, (a\,b).X\rangle, (a\,b).Y\rangle$ *or* $\langle\langle \overline{a}, (a\,b).Y\rangle, (a\,b).X\rangle$. *In general, if* t *is such that* $\|t\|_f = n$, *then* $\langle\langle t_{(\overline{1})}, \ldots, t_{(\overline{n})}\rangle\rangle_f$ *represents* $n! \times$ *the number of different parenthesising of the* n *arguments of the tuple.*

- *In the failure cases, X is associated with a term whose set of variables contains X, but this term is different from a suspension.*

As showed in Exs. 7.3 and 7.4, the nominal C, AC-unification and nominal A, C and AC matching algorithms through simplification rules ($\approx_?$ **A**) and ($\approx_?$ **AC**) are extremely inefficient. Many derivation branches should be avoided in a reasonable implementation. For the moment, the the goal in these rule based algorithms is just to obtain simple sets of rules that would be easy to formalise to be terminating, sound and complete (see Lems. 7.2 to 7.4). Algorithmic improvements regarding efficiency will be subject of future work.

Lemma 7.2 (Termination of $\Rightarrow_{(\approx_?\mathbf{A})}$ and $\Rightarrow_{(\approx_?\mathbf{AC})}$). *Both rules* ($\approx_?$ **A**) *and* ($\approx_?$ **AC**) *are terminating.*

Proof. Using well-founded induction and the same measure $|\mathcal{P}|$ of the proof of Lem. 5.2, one concludes that if either $\mathcal{P} \Rightarrow_{(\approx_?\mathbf{A})} \mathcal{Q}$ or $\mathcal{P} \Rightarrow_{(\approx_?\mathbf{AC})} \mathcal{Q}$, then for $E = A, AC$, $\mathcal{P} = \langle \nabla, \mathcal{X}, \sigma, P' \rangle$, $\mathcal{Q} = \langle \nabla, \mathcal{X}, \sigma, Q \rangle$, $P' = P \uplus \{f_k^E\, s \approx_? f_k^E\, t\}$ and

$$Q = P \cup \{\langle\langle s_{(1)}, \ldots, s_{(m)} \rangle\rangle_{f_k^A} \approx_? \langle\langle t_{(1)}, \ldots, t_{(n)} \rangle\rangle_{f_k^A} \text{ or } \langle\langle s_{(1)}, \ldots, s_{(m)} \rangle\rangle_{f_k^{AC}} \approx_? \langle\langle t_{(\overline{1})}, \ldots, t_{(\overline{n})} \rangle\rangle_{f_k^{AC}}\}.$$

Then in both cases:

1. $|Var(P'_{\approx})| = |Var(Q_{\approx})|$;

2. $|P'_{\approx}| \geq |Q_{\approx}| + 2 > |Q_{\approx}|$, since at least two function symbols f_k^E are eliminated, and $|s| \geq |\langle\langle s_{(1)}, \ldots, s_{(n)} \rangle\rangle_{f_k^E}|$, and either $|t| \geq |\langle\langle t_{(1)}, \ldots, t_{(n)} \rangle\rangle_{f_k^A}|$ or $|t| \geq |\langle\langle t_{(\overline{1})}, \ldots, t_{(\overline{n})} \rangle\rangle_{f_k^{AC}}|$.

□

Lemma 7.3 (Preservation of solutions by $\Rightarrow_{(\approx_?\mathbf{A})}$ and $\Rightarrow_{(\approx_?\mathbf{AC})}$).
If either $\mathcal{P} \Rightarrow_{(\approx_?\mathbf{A})} \mathcal{Q}$ or $\mathcal{P} \Rightarrow_{(\approx_?\mathbf{AC})} \mathcal{Q}$, and $\langle \Delta, \delta \rangle \in \mathcal{U}_{AC}(\mathcal{Q})$ then $\langle \Delta, \delta \rangle \in \mathcal{U}_{AC}(\mathcal{P})$.

Proof. (sketch)
For $\mathcal{P} \Rightarrow_{(\approx_?\mathbf{A})} \mathcal{Q}$ or $\mathcal{P} \Rightarrow_{(\approx_?\mathbf{AC})} \mathcal{Q}$, with $E = A$ or AC, $\mathcal{P} = \langle \nabla, \mathcal{X}, \sigma, P \uplus \{f_k^E\, s \approx_? f_k^E\, t\} \rangle$ and

$$\mathcal{Q} = \langle \nabla, \mathcal{X}, \sigma, P \cup \{\langle\langle s_{(1)}, \ldots, s_{(m)} \rangle\rangle_{f_k^A} \approx_? \langle\langle t_{(1)}, \ldots, t_{(n)} \rangle\rangle_{f_k^A}\} \rangle \text{ or}$$

$$\mathcal{Q} = \langle \nabla, \mathcal{X}, \sigma, P \cup \{\langle\langle s_{(1)}, \ldots, s_{(m)} \rangle\rangle_{f_k^{AC}} \approx_? \langle\langle t_{(\overline{1})}, \ldots, t_{(\overline{n})} \rangle\rangle_{f_k^{AC}}\} \rangle.$$

Except for the third condition of Def. 7.1, all other conditions are proved trivially. For the third condition, given $\langle \Delta, \delta \rangle \in \mathcal{U}_{AC}(\mathcal{Q})$, observe that

$$\Delta \vdash f_k^A\, s\delta \approx_{\{A,C,AC\}} f_k^A\, \langle\langle s_{(1)}\delta, \ldots, s_{(m)}\delta \rangle\rangle_{f_k^A} \text{ and } \Delta \vdash f_k^A\, t\delta \approx_{\{A,C,AC\}} f_k^A\, \langle\langle t_{(1)}\delta \ldots, t_{(n)}\delta \rangle\rangle_{f_k^A}, \text{ and}$$

$\Delta \vdash f_k^{AC}\, s\delta \approx_{\{A,C,AC\}} f_k^{AC}\, \langle\langle s_{(1)}\delta, \ldots, s_{(m)}\delta\rangle\rangle_{f_k^{AC}}$ and $\Delta \vdash f_k^{AC}\, t\delta \approx_{\{A,C,AC\}} f_k^{AC}\, \langle\langle t_{(\overline{1})}\delta, \ldots, t_{(\overline{n})}\delta\rangle\rangle_{f_k^{AC}}$. But, because $\langle \Delta, \delta\rangle \in \mathcal{U}_{AC}(\mathcal{Q})$, one has either

$$\Delta \vdash \langle\langle s_{(1)}\delta, \ldots, s_{(m)}\delta\rangle\rangle_{f_k^{A}} \approx_{\{A,C,AC\}} \langle\langle t_{(1)}\delta, \ldots, t_{(n)}\delta\rangle\rangle_{f_k^{A}} \text{ or}$$

$$\Delta \vdash \langle\langle s_{(1)}\delta, \ldots, s_{(m)}\delta\rangle\rangle_{f_k^{AC}} \approx_{\{A,C,AC\}} \langle\langle t_{(\overline{1})}\delta, \ldots, t_{(\overline{n})}\delta\rangle\rangle_{f_k^{AC}},$$

which implies that either

$$\Delta \vdash f_k^{A}\, \langle\langle s_{(1)}\delta, \ldots, s_{(m)}\delta\rangle\rangle_{f_k^{A}} \approx_{\{A,C,AC\}} f_k^{A}\, \langle\langle t_{(1)}\delta, \ldots, t_{(n)}\delta\rangle\rangle_{f_k^{A}} \text{ or}$$

$$\Delta \vdash f_k^{AC}\, \langle\langle s_{(1)}\delta, \ldots, s_{(m)}\delta\rangle\rangle_{f_k^{AC}} \approx_{\{A,C,AC\}} f_k^{AC}\, \langle\langle t_{(\overline{1})}\delta, \ldots, t_{(\overline{n})}\delta\rangle\rangle_{f_k^{AC}}.$$

Finally, by transitivity of $\approx_{\{A,C,AC\}}$ (Lem. 4.8) and Def. 2.24, one concludes that either $\Delta \vdash (f_k^{A}\, s)\delta \approx_{\{A,C,AC\}} (f_k^{A}\, t)\delta$ or $\Delta \vdash (f_k^{AC}\, s)\delta \approx_{\{A,C,AC\}} (f_k^{AC}\, t)\delta$, which completes the proof of the third condition of Def. 7.1 for rules ($\approx_?$ **A**) and ($\approx_?$ **AC**) respectively. □

The completeness of rule $\Rightarrow_{(\approx_? \mathbf{A})}$ is showed (in Lem. 7.4) only for the case where the set of protected variables $\mathcal{X}$ contains the *rhs* variables of the problem (the matching case). This restriction is necessary because rule ($\approx_?$ **A**) does not capture all infinite solutions that may be generated in A-unification (see Subsec. 2.1.2). For instance, let f be an A function symbol in the input problem $\mathcal{P} = \langle \emptyset, \emptyset, id, \{f\langle X, \overline{a}\rangle \approx_? f\langle \overline{a}, X\rangle\}\rangle$. Rule ($\approx_?$ **A**) transforms $\mathcal{P}$ into $\langle \emptyset, \emptyset, id, \{\langle X, \overline{a}\rangle \approx_? \langle \overline{a}, X\rangle\}\rangle$, that generates a unique solution $\langle \emptyset, \{X/\overline{a}\}\rangle$. However, a complete solutions set for $\mathcal{P}$ would include:

$$\langle \emptyset, \{X/f\langle \overline{a}, \overline{a}\rangle\}\rangle;\ \langle \emptyset, \{X/f\langle \overline{a}, f\langle \overline{a}, \overline{a}\rangle\rangle\}\rangle;\ \langle \emptyset, \{X/f\langle f\langle \overline{a}, \overline{a}\rangle, f\langle \overline{a}, \overline{a}\rangle\rangle\}\rangle;\ \text{etc.}$$

Lemma 7.4 (Completeness of $\Rightarrow_{(\approx_? \mathbf{A})}$ and $\Rightarrow_{(\approx_? \mathbf{AC})}$)**.** *Let $\mathcal{P} = \langle \Delta, \mathcal{X}, \sigma, P\rangle$ be a quadruple that is not in $\Rightarrow_{\approx}$-nf and either P has occurrences of A function symbols or* $\texttt{Rvar}(P) \subseteq \mathcal{X}$*. If $\langle \Delta, \delta\rangle \in \mathcal{U}_{AC}(\mathcal{P})$, then there exists $\mathcal{Q}$, such that $\mathcal{P} \Rightarrow_{\approx} \mathcal{Q}$ and $\langle \Delta, \delta\rangle \in \mathcal{U}_{AC}(\mathcal{Q})$.*

Proof. (sketch) The proof is by case analysis on the derivation rules of $\Rightarrow_{\upsilon}$. Except for cases of rules ($\approx_?$ **A**) and ($\approx_?$ **AC**), the proof is covered in Lem. 5.2. For these remaining cases:

- If $P = P' \uplus \{f_k^{A}\, s \approx_? f_k^{A}\, t\}$ and $\texttt{Rvar}(P) \subseteq \mathcal{X}$, then, for the third condition of Def. 7.1 one has $\Delta \vdash f_k^{A}\, s\delta \approx_{\{A,C,AC\}} f_k^{A}\, t$, which implies that there exists $\langle\langle s_{(1)}, \ldots, s_{(m)}\rangle\rangle_{f_k^{A}}$ and $\langle\langle t_{(1)}, \ldots, t_{(n)}\rangle\rangle_{f_k^{A}}$, such that $\Delta \vdash \langle\langle s_{(1)}\delta, \ldots, s_{(m)}\delta\rangle\rangle_{f_k^{A}} \approx_{\{A,C,AC\}} \langle\langle t_{(1)}, \ldots, t_{(n)}\rangle\rangle_{f_k^{A}}$.

Thus

$$\mathcal{Q} = \langle \nabla, \mathcal{X}, \sigma, P' \uplus \{\langle\langle s_{(1)}\delta, \dots, s_{(m)}\delta\rangle\rangle_{f_k^A} \approx_? \langle\langle t_{(1)}, \dots, t_{(n)}\rangle\rangle_{f_k^A}\}\rangle;$$

- Similarly, if $P = P' \uplus \{f_k^{AC}\, s \approx_? f_k^{AC}\, t\}$, then there exists $\langle\langle s_{(1)}, \dots, s_{(m)}\rangle\rangle_{f_k^{AC}}$ and $\langle\langle t_{(\overline{1})}, \dots, t_{(\overline{n})}\rangle\rangle_{f_k^{AC}}$, such that $\Delta \vdash \langle\langle s_{(1)}\delta, \dots, s_{(m)}\delta\rangle\rangle_{f_k^{AC}} \approx_{\{A,C,AC\}} \langle\langle t_{(\overline{1})}\delta, \dots, t_{(\overline{n})}\delta\rangle\rangle_{f_k^{AC}}$. Thus

$$\mathcal{Q} = \langle \nabla, \mathcal{X}, \sigma, P' \uplus \{\langle\langle s_{(1)}\delta, \dots, s_{(m)}\delta\rangle\rangle_{f_k^{AC}} \approx_? \langle\langle t_{(\overline{1})}\delta, \dots, t_{(\overline{n})}\delta\rangle\rangle_{f_k^{AC}}\}\rangle.$$

From this, one concludes that all conditions of Def. 7.1 are satisfied for $\langle \Delta, \delta\rangle$. □

7.2 Solutions for nominal AC FP problems

Derivations by the nominal C-unification algorithm extended with rules of Figs. 7.1 and 7.2 also result in a set of FP problems. This affirmation can be proved by a similar analysis that one presented in the proof of Lem. 5.5. Then combinatorial solutions for these FP problems must also include the A and AC function symbols of the signature. For this reason, the definition of `epc` is changed in Def. 7.3, adding item 4 in the construction of combinatorial solutions with AC function symbols. It is verified that a complete set of solutions for AC FP problems may be infinite. Then, as in Chap. 6, nominal AC-unification has also been showed at least infinitary. As mentioned in Rmk. 6.9, the notion of *more general than* (denoted by $\preceq$) and *complete set of solutions* given by Def. 2.26 are now extended to be used in the context of the $\approx_{\{A,C,AC\}}$ relation.

Definition 7.3 (Extended Pseudo-cycle with C and AC function symbols)**.** *Let $\pi.X \approx_? X$ and $\mathcal{Z}$ a set of variables. The AC* extended pseudo-cycles *(denoted by AC* `epc`*) κ for π relative to $\mathcal{Z}$ are inductively defined from the permutation cycles of π as follows:*

1. *$\kappa = (Y)$, for any variable not occurring in $\mathcal{Z}$, is an AC* `epc` *for π;*
2. *$\kappa = (\overline{a_0} \cdots \overline{a_{k-1}})$ is an AC* `epc` *for $(a_0\ \cdots a_{k-1})$ a permutation cycle in π such that $k = 2^l$, for $l > 0$, called a* trivial extended pseudo-cycle *of π.*
3. *$\kappa = (A_0 \dots A_{k-1})$, for a length $k \geq 1$, is an AC* `epc` *for π, if the following conditions are simultaneously satisfied:*
   - *(a)* i. *each element of κ is of the form $B_i \star B_j$, where $\star$ is a commutative function symbol in the signature, and B_i, B_j are different elements of κ', an AC* `epc`

for π; in this case, κ will be called a first-instance extended pseudo-cycle *of κ' for π; or*

ii. *each element of κ is of the form $B_i \star C_j$ for any commutative symbol $\star$, where B_i and C_j are elements of κ' and κ'' AC* `epc`*'s for π, which might both be the same, but κ is not a first-instance AC* `epc` *for π; or*

iii. *each element of κ is of the form $\langle B_i, C_j\rangle$, where B_i and C_j are elements of κ' and κ'' AC* `epc`*'s for π, which might both be the same; or*

iv. *either each element of κ is of the form $g\, B_i$ or each element is of the form $[e]\, B_i$, where g is a non commutative function symbol in the signature and $e \notin dom(\pi)$, and each B_i is an element of κ' an AC* `epc` *for π; or*

v. *each element of κ is of the form $[a_j]B_i$, where a_j are atoms in $\kappa' = (\overline{a_0} \cdots \overline{a_{k'-1}})$ a trivial AC* `epc` *for π, and B_i elements of κ'' an AC* `epc` *for π; and*

(b) *for $\nabla' = \cup_{Y \in Var(\kappa)}\{dom(\pi)\#Y\}$,*

i. *it does not hold that $\nabla' \vdash A_i \approx_{\{A,C,AC\}} A_j$ for $i \neq j$, $0 \leq i, j \leq k-1$; and*

ii. *for each $0 \leq i \leq k-1$ one has that $\nabla' \vdash \pi(A_i) \approx_{\{A,C,AC\}} A_{(i+1)\, mod\, k}$.*

4. *Let $\star$ be an AC function symbol and $s_{(1)} = A_0, \ldots, s_{(n)} = A_{k-1}$ in $\kappa = (\star\langle\langle s_{(\overline{1})}, \ldots, s_{(\overline{n})}\rangle\rangle_\star)$. For a length $k \geq 1$, κ is a unitary AC* `epc` *for π, if $(A_0 \ldots A_{k-1})$ is an AC* `epc` *for π.*

Remark 7.2. *Observe that, Item 4 of Def. 7.3 allows the construction of unitary AC* `epc`*'s, arranging the elements of a previous arbitrary AC* `epc`*. Thus, differently from C FP equations, not only k-cycles whose length is power of two, but any k-cycle with arbitrary length can generate infinite unitary AC* `epc`*'s. Moreover, Thm. 7.1 shows that each of these unitary AC* `epc`*'s is associated to a solution for the corresponding AC FP equation.*

The definition of the set of *generated solutions for singleton AC FP problems*, denoted also by $\langle\Delta, \mathcal{X}, \{\pi.X \approx_? X\}\rangle_{Sol_G}$, is a simple adaptation of Def. 6.5 that considers the extended definition of AC `epc` (Def. 7.3).

Example 7.5. *Let f be an AC function symbol. The (current) example shows a derivation for the input quadruple $\mathcal{P}$.*

$$\mathcal{P} = \langle\emptyset, \emptyset, id, \{[a][d][b][d]f\langle[c][d]X, f\langle[a][b]X, [b][c]Y\rangle\rangle \approx_? [d][a][a][b]f\langle f\langle[b][c]X, [a][d]X\rangle, [c][a]Y\rangle\}\rangle$$

$$\Rightarrow_{(\approx_?[\mathbf{ab}])} \langle\emptyset, \emptyset, id, \left\{\begin{array}{l} [d][b][d]f\langle[c][d]X, f\langle[a][b]X, [b][c]Y\rangle\rangle \approx_? \\ [d][d][b]f\langle f\langle[b][c](a\,d).X, [d][a](a\,d).X\rangle, [c][d](a\,d).Y\rangle, \\ a \#_? [a][a][b]f\langle f\langle[b][c]X, [a][d]X\rangle, [c][a]X\rangle \end{array}\right\}\rangle$$

$$\Rightarrow_{(\approx_?[\mathbf{aa}])} \langle\emptyset,\emptyset,id,\left\{\begin{array}{l} [b][d]f\langle[c][d]X, f\langle[a][b]X,[b][c]Y\rangle\rangle \approx_? \\ [d][b]f\langle f\langle[b][c](a\,d).X,[d][a](a\,d).X\rangle,[c][d](a\,d).Y\rangle, \\ a \#_? [a][a][b]f\langle f\langle[b][c]X,[a][d]X\rangle,[c][a]Y\rangle \end{array}\right\}\rangle$$

$$\Rightarrow_{(\approx_?[\mathbf{ab}])} \langle\emptyset,\emptyset,id,\left\{\begin{array}{l} [d]f\langle[c][d]X, f\langle[a][b]X,[b][c]Y\rangle\rangle \approx_? \\ [d]f\langle f\langle[d][c](a\,b\,d).X,[b][a](a\,b\,d).X\rangle,[c][b](a\,b\,d).Y\rangle, \\ a \#_? [a][a][b]f\langle f\langle[b][c]X,[a][d]X\rangle,[c][a]Y\rangle, \\ b \#_? [b]f\langle f\langle[b][c](a\,d).X,[d][a](a\,d).X\rangle,[c][d](a\,d)Y\rangle \end{array}\right\}\rangle$$

$$\Rightarrow_{(\approx_?[\mathbf{aa}])} \langle\emptyset,\emptyset,id,\left\{\begin{array}{l} f\langle[c][d]X, f\langle[a][b]X,[b][c]Y\rangle\rangle \approx_? \\ f\langle f\langle[d][c](a\,b\,d).X,[b][a](a\,b\,d).X\rangle,[c][b](a\,b\,d).Y\rangle, \\ a \#_? [a][a][b]f\langle f\langle[b][c]X,[a][d]X\rangle,[c][a]Y\rangle, \\ b \#_? [b]f\langle f\langle[b][c](a\,d).X,[d][a](a\,d).X\rangle,[c][d](a\,d).Y\rangle \end{array}\right\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{AC})} \langle\emptyset,\emptyset,id,\left\{\begin{array}{l} \langle\langle s_{(1)},s_{(2)},s_{(3)}\rangle\rangle_f \approx_? \langle\langle t_{(\overline{1})},t_{(\overline{2})},t_{(\overline{3})}\rangle\rangle_f, \\ a \#_? [a][a][b]f\langle f\langle[b][c]X,[a][d]X\rangle,[c][a]Y\rangle, \\ b \#_? [b]f\langle f\langle[b][c](a\,d).X,[d][a](a\,d).X\rangle,[c][d](a\,d).Y\rangle \end{array}\right\}\rangle$$

Let s and t be, respectively, equal to $\langle[c][d]X, f\langle[a][b]X,[b][c]Y\rangle\rangle$ and $\langle f\langle[d][c](a\,b\,d).X,[b][a](a\,b\,d).X\rangle,[c][b](a\,b\,d).Y\rangle$ The previous application of rule AC generates twenty-four branches, with six different successful leaves. Between these, just one results in a FP problem with infinite AC combinatorial solutions over 4-cycles. A derivation branch that gives rise to this case is presented below.

$$\Rightarrow_{(\approx_?\mathbf{pair})2\times} \langle\emptyset,\emptyset,id,\left\{\begin{array}{l} [c][d]X \approx_? [d][c](a\,b\,d).X,\ [a][b]X \approx_? [b][a](a\,b\,d).X,\ [b][c]Y \approx_? [c][b](a\,b\,d).Y, \\ a \#_? [a][a][b]f\langle f\langle[b][c]X,[a][d]X\rangle,[c][a]Y\rangle, \\ b \#_? [b]f\langle f\langle[b][c](a\,d).X,[d][a](a\,d).X\rangle,[c][d](a\,d).Y\rangle \end{array}\right\}\rangle$$

$$\Rightarrow_{(\approx_?[\mathbf{ab}])} \langle\emptyset,\emptyset,id,\left\{\begin{array}{l} [d]X \approx_? [d](a\,b\,c\,d).X,\ [a][b]X \approx_? [b][a](a\,b\,d).X,\ [b][c]Y \approx_? [c][b](a\,b\,d).Y, \\ c \#_? [c](a\,b\,d).X, \\ a \#_? [a][a][b]f\langle f\langle[b][c]X,[a][d]X\rangle,[c][a]Y\rangle, \\ b \#_? [b]f\langle f\langle[b][c](a\,d).X,[d][a](a\,d).X\rangle,[c][d](a\,d).Y\rangle \end{array}\right\}\rangle$$

$$\Rightarrow_{(\approx_?[\mathbf{aa}])} \langle\emptyset,\emptyset,id,\left\{\begin{array}{l} X \approx_? (a\,b\,c\,d).X,\ [a][b]X \approx_? [b][a](a\,b\,d).X,\ [b][c]Y \approx_? [c][b](a\,b\,d).Y, \\ c \#_? [c](a\,b\,d).X, \\ a \#_? [a][a][b]f\langle f\langle[b][c]X,[a][d]X\rangle,[c][a]Y\rangle, \\ b \#_? [b]f\langle f\langle[b][c](a\,d).X,[d][a](a\,d).X\rangle,[c][d](a\,d).Y\rangle \end{array}\right\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{inv})} \langle\emptyset,\emptyset,id,\left\{\begin{array}{l} (d\,c\,b\,a).X \approx_? X,\ [a][b]X \approx_? [b][a](a\,b\,d).X,\ [b][c]Y \approx_? [c][b](a\,b\,d).Y, \\ c \#_? [c](a\,b\,d).X, \\ a \#_? [a][a][b]f\langle f\langle[b][c]X,[a][d]X\rangle,[c][a]Y\rangle, \\ b \#_? [b]f\langle f\langle[b][c](a\,d).X,[d][a](a\,d).X\rangle,[c][d](a\,d).Y\rangle \end{array}\right\}\rangle$$

$$\Rightarrow_{(\approx_?[\mathbf{ab}])} \langle\emptyset,\emptyset,id,\left\{\begin{array}{l}(d\,c\,b\,a).X \approx_? X,\ \ [b]X \approx_? [b](b\,d).X,\ [b][c]Y \approx_? [c][b](a\,b\,d).Y,\\ a \#_? [a](a\,b\,d).X,\ c \#_? [c](a\,b\,d).X,\\ a \#_? [a][a][b]f\langle f\langle [b][c]X, [a][d]X\rangle, [c][a]Y\rangle,\\ b \#_? [b]f\langle f\langle [b][c](a\,d).X, [d][a](a\,d).X\rangle, [c][d](a\,d).Y\rangle\end{array}\right\}\rangle$$

$$\Rightarrow_{(\approx_?[\mathbf{aa}])} \langle\emptyset,\emptyset,id,\left\{\begin{array}{l}(d\,c\,b\,a).X \approx_? X,\ \ X \approx_? (b\,d).X,\ [b][c]Y \approx_? [c][b](a\,b\,d).Y,\\ a \#_? [a](a\,b\,d).X,\ c \#_? [c](a\,b\,d).X,\\ a \#_? [a][a][b]f\langle f\langle [b][c]X, [a][d]X\rangle, [c][a]Y\rangle,\\ b \#_? [b]f\langle f\langle [b][c](a\,d).X, [d][a](a\,d).X\rangle, [c][d](a\,d).Y\rangle\end{array}\right\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{inv})} \langle\emptyset,\emptyset,id,\left\{\begin{array}{l}(d\,c\,b\,a).X \approx_? X,\ (d\,b).X \approx_? X,\ \ [b][c]Y \approx_? [c][b](a\,b\,d).Y,\\ a \#_? [a](a\,b\,d).X,\ c \#_? [c](a\,b\,d).X,\\ a \#_? [a][a][b]f\langle f\langle [b][c]X, [a][d]X\rangle, [c][a]Y\rangle,\\ b \#_? [b]f\langle f\langle [b][c](a\,d).X, [d][a](a\,d).X\rangle, [c][d](a\,d).Y\rangle\end{array}\right\}\rangle$$

$$\Rightarrow_{(\approx_?[\mathbf{ab}])} \langle\emptyset,\emptyset,id,\left\{\begin{array}{l}(d\,c\,b\,a).X \approx_? X,\ (d\,b).X \approx_? X,\ \ [c]Y \approx_? [c](a\,c\,b\,d).Y,\\ b \#_? [b](a\,b\,d).Y,\ a \#_? [a](a\,b\,d).X,\ c \#_? [c](a\,b\,d).X,\\ a \#_? [a][a][b]f\langle f\langle [b][c]X, [a][d]X\rangle, [c][a]Y\rangle,\\ b \#_? [b]f\langle f\langle [b][c](a\,d).X, [d][a](a\,d).X\rangle, [c][d](a\,d).Y\rangle\end{array}\right\}\rangle$$

$$\Rightarrow_{(\approx_?[\mathbf{aa}])} \langle\emptyset,\emptyset,id,\left\{\begin{array}{l}(d\,c\,b\,a).X \approx_? X,\ (d\,b).X \approx_? X,\ \ Y \approx_? (a\,c\,b\,d).Y,\\ b \#_? [b](a\,b\,d).Y,\ a \#_? [a](a\,b\,d).X,\ c \#_? [c](a\,b\,d).X,\\ a \#_? [a][a][b]f\langle f\langle [b][c]X, [a][d]X\rangle, [c][a]Y\rangle,\\ b \#_? [b]f\langle f\langle [b][c](a\,d).X, [d][a](a\,d).X\rangle, [c][d](a\,d).Y\rangle\end{array}\right\}\rangle$$

$$\Rightarrow_{(\approx_?\mathbf{inv})} \langle\emptyset,\emptyset,id,\left\{\begin{array}{l}(d\,c\,b\,a).X \approx_? X,\ (d\,b).X \approx_? X,\ (d\,b\,c\,a).Y \approx_? Y,\\ b \#_? [b](a\,b\,d).Y,\ \ a \#_? [a](a\,b\,d).X,\ \ c \#_? [c](a\,b\,d).X,\\ a \#_? [a][a][b]f\langle f\langle [b][c]X, [a][d]X\rangle, [c][a]Y\rangle,\\ b \#_? [b]f\langle f\langle [b][c](a\,d).X, [d][a](a\,d).X\rangle, [c][d](a\,d).Y\rangle\end{array}\right\}\rangle$$

$$\Rightarrow_{(\#_?\,\mathbf{a}[\mathbf{a}])5\times} \langle\emptyset,\emptyset,id,\{(d\,c\,b\,a).X \approx_? X,\ (d\,b).X \approx_? X,\ (d\,b\,c\,a).Y \approx_? Y\}\rangle$$

Example 7.6 (Continuing Ex. 7.5). *Let $*$ and $\circ$; $\star$ and $\bullet$; and f and g; be, respectively C, AC and syntactic function symbols. C function symbols used in infix notation. Unitary AC* epc*'s for the equations: 1. $(d\,c\,b\,a).X \approx_? X$; 2. $(d\,b).X \approx_? X$; and 3. $(d\,b\,c\,a).Y \approx_? Y$, generated via Def. 7.3, can be given, respectively, by:*

1. $(\overline{d} * \overline{b}) \circ (\overline{c} * \overline{a})$; $\star\langle\langle\overline{a},\overline{b}\rangle,\langle\overline{d},\overline{c}\rangle\rangle$; $\star\langle\langle\langle\overline{a},Z\rangle,\langle\overline{b},Z\rangle\rangle,\langle\langle\overline{d},Z\rangle,\langle\overline{c},Z\rangle\rangle\rangle$;
   $\star\langle\langle\langle f\overline{a},Z\rangle,\langle f\overline{b},Z\rangle\rangle,\langle\langle f\overline{d},Z\rangle,\langle f\overline{c},Z\rangle\rangle\rangle$;
   $\star\langle\langle\langle f\overline{a},(\overline{d}*\overline{b})\circ(\overline{c}*\overline{a})\rangle,\langle f\overline{b},(\overline{d}*\overline{b})\circ(\overline{c}*\overline{a})\rangle\rangle,\langle\langle f\overline{d},(\overline{d}*\overline{b})\circ(\overline{c}*\overline{a})\rangle,\langle f\overline{c},(\overline{d}*\overline{b})\circ(\overline{c}*\overline{a})\rangle\rangle\rangle$;

2. $Z \circ (\overline{d} * \overline{b})$; $\star\langle\langle\overline{b},\langle W,Z\rangle\rangle,\overline{d}\rangle$; $\star\langle\langle W_0,\langle\overline{b},Z\rangle\rangle,\langle\langle\overline{d},Z\rangle,W_1\rangle\rangle$;
   $(g\,Z) \circ ([e]\overline{d} * [e]\overline{b})$; $\bullet\langle\langle(g\,Z)\circ([e]\overline{d} * [e]\overline{b}),\overline{b}\rangle,\langle(g\,Z)\circ([e]\overline{d} * [e]\overline{b}),\overline{d}\rangle\rangle$;
   $\langle\bullet\langle\langle(g\,Z)\circ([e]\overline{d} * [e]\overline{b}),\overline{b}\rangle,\langle(g\,Z)\circ([e]\overline{d} * [e]\overline{b}),\overline{d}\rangle\rangle,W\rangle$;

3. $(\overline{d}{*}\overline{c})\circ(\overline{b}{*}\overline{a})$; $(f\overline{d}{*}f\overline{c})\circ(f\overline{b}{*}f\overline{a})$; $\star\langle\langle\langle\overline{a},\overline{b}\rangle,\overline{c}\rangle,\overline{d}\rangle$; $\star\langle\langle\langle g\langle\overline{a},Z\rangle,g\langle\overline{b},Z\rangle\rangle,g\langle\overline{c},Z\rangle\rangle,g\langle\overline{d},Z\rangle\rangle$.

Theorem 7.1 (Soundness of solutions for singleton AC FP problems)**.** *Each* $\langle\nabla,\{X/s\}\rangle$ *in* $\langle\Delta,\mathcal{X},\{\pi.X\approx_? X\}\rangle_{Sol_G}$ *is a solution for* $\langle\Delta,\mathcal{X},\{\pi.X\approx_? X\}\rangle$.

Proof. The proof is by case analysis on the definition of AC `epc`. Cases of items 1 to 3 are trivially adapted from the proof of Lem. 6.6. For item 4, observe that $\nabla\vdash\pi(A_i)\approx_{\{A,C,AC\}} A_{(i+1)\,mod\,k}$ and then, because $\star$ is an AC function symbol, $\nabla\vdash\pi(\star\langle\langle A_0,\ldots,A_{k-1}\rangle\rangle_\star)\approx_{\{A,C,AC\}}\star\langle\langle A_0,\ldots,A_{k-1}\rangle\rangle_\star$. □

Theorem 7.2 (Completeness of solutions for singleton AC FP problems)**.** *Let* $\langle\Delta,\mathcal{X},\{\pi.X\approx_? X\}\rangle$ *be a singleton AC FP problem with a solution* $\langle\nabla,\{X/s\}\rangle$*. Then there exists* $\langle\nabla',\{X/t\}\rangle\in\langle\Delta,\mathcal{X},\{\pi.X\approx_? X\}\rangle_{Sol_G}$ *such that* $\langle\nabla',\{X/t\}\rangle\preceq\langle\nabla,\{X/s\}\rangle$.

Proof. The proof is by induction on the structure of s. All the analysis is already done in the proof of Lem. 6.7, except when $s=\star t$, where $\star$ is an AC function symbol. For this case, $\nabla\vdash\pi\cdot(\star t)\approx_{\{A,C,AC\}}\star t$. Observe that, for $n=\|s\|_\star$, $\nabla\vdash\pi\cdot(\star t)\approx_{\{A,C,AC\}}\star\langle\langle\pi\cdot s_1,\ldots,\pi\cdot s_n\rangle\rangle_\star$ and $\nabla\vdash\star t\approx_{\{A,C,AC\}}\star\langle\langle s_1,\ldots,s_n\rangle\rangle_\star$. Then, by transitivity of $\approx_{\{A,C,AC\}}$ (Lem. 4.8), one has

$$\nabla\vdash\langle\langle\pi\cdot s_1,\ldots,\pi\cdot s_n\rangle\rangle_\star\approx_{\{A,C,AC\}}\langle\langle s_1,\ldots,s_n\rangle\rangle_\star.$$

This last assertion is possible only if either, for $i=1..n$, $\nabla\vdash\pi\cdot s_i\approx_{\{A,C,AC\}} s_i$ or there exists $j\neq i$ in $\{1,\ldots,n\}$, such that $\nabla\vdash\pi\cdot s_i\approx_{\{A,C,AC\}} s_j$. The former case is covered by items 3.iii) and 3.iv), and the latter by item 4 of Def. 7.3. □

Example 7.7 (Continuing Ex. 7.6)**.** *Let* $\mathcal{S}_0$, $\mathcal{S}_1$ *and* $\mathcal{S}_2$ *be, respectively, equal to:*

$$\left\{\begin{array}{l}(\overline{d}*\overline{b})\circ(\overline{c}*\overline{a}),\ \star\langle\langle\overline{a},\overline{b}\rangle,\langle\overline{d},\overline{c}\rangle\rangle,\ \star\langle\langle\langle\overline{a},Z\rangle,\langle\overline{b},Z\rangle\rangle,\langle\langle\overline{d},Z\rangle,\langle\overline{c},Z\rangle\rangle\rangle,\\ \star\langle\langle\langle f\overline{a},Z\rangle,\langle f\overline{b},Z\rangle\rangle,\langle\langle f\overline{d},Z\rangle,\langle f\overline{c},Z\rangle\rangle\rangle,\\ \star\langle\langle\langle f\overline{a},(\overline{d}*\overline{b})\circ(\overline{c}*\overline{a})\rangle,\langle f\overline{b},(\overline{d}*\overline{b})\circ(\overline{c}*\overline{a})\rangle\rangle,\langle\langle f\overline{d},(\overline{d}*\overline{b})\circ(\overline{c}*\overline{a})\rangle,\langle f\overline{c},(\overline{d}*\overline{b})\circ(\overline{c}*\overline{a})\rangle\rangle\rangle\end{array}\right\},$$

$$\left\{\begin{array}{l}Z\circ(\overline{d}*\overline{b}),\ \star\langle\langle\overline{b},\langle W,Z\rangle\rangle,\overline{d}\rangle,\ \star\langle\langle W_0,\langle\overline{b},Z\rangle\rangle,\langle\langle\overline{d},Z\rangle,W_1\rangle\rangle,\ (g\,Z)\circ([e]\overline{d}*[e]\overline{b}),\\ \bullet\langle\langle(g\,Z)\circ([e]\overline{d}*[e]\overline{b}),\overline{b}\rangle,\langle(g\,Z)\circ([e]\overline{d}*[e]\overline{b}),\overline{d}\rangle\rangle,\\ \langle\bullet\langle\langle(g\,Z)\circ([e]\overline{d}*[e]\overline{b}),\overline{b}\rangle,\langle(g\,Z)\circ([e]\overline{d}*[e]\overline{b}),\overline{d}\rangle\rangle,W\rangle\end{array}\right\},$$

$\{(\overline{d}{*}\overline{c})\circ(\overline{b}{*}\overline{a}),\ (f\overline{d}{*}f\overline{c})\circ(f\overline{b}{*}f\overline{a}),\ \star\langle\langle\langle\overline{a},\overline{b}\rangle,\overline{c}\rangle,\overline{d}\rangle,\ \star\langle\langle\langle g\langle\overline{a},Z\rangle,g\langle\overline{b},Z\rangle\rangle,g\langle\overline{c},Z\rangle\rangle,g\langle\overline{d},Z\rangle\rangle\}$.

The sets $\{\{X/e\}\mid e\in\mathcal{S}_0\}$, $\{\{X/e\}\mid e\in\mathcal{S}_1\}$, $\{\{Y/e\}\mid e\in\mathcal{S}_2\}$ *contain solutions for the respective singleton problems:* $\langle\emptyset,\emptyset,\{(d\,c\,b\,a).X\approx_? X\}\rangle$, $\langle\emptyset,\emptyset,\{(d\,b).X\approx_? X\}\rangle$ *and* $\langle\emptyset,\emptyset,\{(d\,b\,c\,a).Y\approx_? Y\}\rangle$.

Definition 7.4 (General AC-matchers)**.** *Let s_i, for $i = 1..k$, be terms. A* general AC-matcher *δ is defined as a C, AC-most general solution for the C, AC-unification problem $\{s_i =_? Z\}_{i=1..k}$, where Z is a new variable for s_i, with $i = 1..k$.*

Remark 7.3. *Def. 7.4 can be seen as an extension of Def. 6.6 that uses a first-order C, AC-unification algorithm, instead of just first-order C-unification. Such algorithm can be obtained by the combination of algorithms presented in Subsec. 2.1.2 of Chap. 2.*

The following Def. 7.5 is an adaptation of Def. 6.7.

Definition 7.5 (Generated solutions for a variable in AC FP problems)**.** *Let the AC FP problems for X in $\mathcal{P}$ be given by $\langle \nabla, \mathcal{X}, \pi_i.X \approx_? X\rangle$, for $\pi_i \in \Pi_X$, and such that $|\Pi_X| = k$. If there exist*

- *solutions $\langle \nabla_i, \mathcal{X}, \{X/t_i\}\rangle \in \langle \nabla, \mathcal{X}, \pi_i.X \approx_? X\rangle_{Sol_G}$ for each AC FP problem and*
- *a most general AC-matcher δ of the terms $\{t_i\}_{i=1..k}$ with X as new variable*

such that the problem $\langle \emptyset, \cup_{(a\#Y)\in\nabla''}\{a\#Y\delta\}\rangle$, where $\nabla'' := \cup_{i=1}^{k}\nabla_i$, has a solution $\langle \nabla', \emptyset\rangle$, then one says that $\langle \nabla', \{X/X\delta\}\rangle$ is a generated solution *for X. The set of all generated solutions is denoted by $[X]_{\mathcal{P}_G}$.*

Example 7.8 (Continuing Exs. 7.5 and 7.7)**.**
Let $\mathcal{Q}$ be equal to $\langle \emptyset, \emptyset, id, \{(d\,c\,b\,a).X \approx_? X,\ (d\,b).X \approx_? X,\ (d\,b\,c\,a).Y \approx_? Y\}\rangle$, then $\{X/(\overline{d} * \overline{b}) \circ (\overline{c} * \overline{a})\}$, $\{X/\star\langle\langle \overline{a}, \overline{b}\rangle, \langle \overline{d}, \overline{c}\rangle\rangle\}$ and $\{X/\star\langle\langle\langle \overline{a}, Z\rangle, \langle \overline{b}, Z\rangle\rangle, \langle\langle \overline{d}, Z\rangle, \langle \overline{c}, Z\rangle\rangle\rangle\} \in [X]_{\mathcal{Q}_G}$, since for δ_0, δ_1 and δ_2, respectively, equal to

$$\{Z/\overline{c} * \overline{a}\}, \{W/\overline{a}, Z/\overline{c}\} \text{ and } \{W_0/\langle \overline{a}, Z\rangle, W_1/\langle \overline{c}, Z\rangle\},$$

one has that: $(\overline{d} * \overline{b}) \circ (\overline{c} * \overline{a}) \approx_{C,AC} Z \circ (\overline{d} * \overline{b})\delta_1$; $\star\langle\langle \overline{a}, \overline{b}\rangle, \langle \overline{d}, \overline{c}\rangle\rangle \approx_{C,AC} \star\langle\langle \overline{b}, \langle W, Z\rangle\rangle, \overline{d}\rangle\delta_2$; and $\star\langle\langle\langle \overline{a}, Z\rangle, \langle \overline{b}, Z\rangle\rangle, \langle\langle \overline{d}, Z\rangle, \langle \overline{c}, Z\rangle\rangle\rangle \approx_{C,AC} \star\langle\langle W_0, \langle \overline{b}, Z\rangle\rangle, \langle\langle \overline{d}, Z\rangle, W_1\rangle\rangle\delta_3$.

Definition 7.6 (Generated Solutions for AC FP problems)**.** *Let $\mathcal{P}$ be an AC FP problem. The set of generated solutions for $\mathcal{P}$, denoted as $[\mathcal{P}]_{Sol_G}$, is defined as the set that contains all solutions of the form*
$\left\langle \bigcup_{X\in Var(P)} \nabla_X, \bigcup_{X\in Var(P)} \{X/s_X\} \right\rangle$*, where each $\langle \nabla_X, \{X/s_X\}\rangle \in [X]_{\mathcal{P}_G}$.*

Example 7.9 (Continuing Exs. 7.6 and 7.8)**.** *For instance, the pairs:*
$\langle \emptyset, \{X/(\overline{d} * \overline{b}) \circ (\overline{c} * \overline{a}), Y/(\overline{d} * \overline{c}) \circ (\overline{b} * \overline{a}),\ (f\overline{d} * f\overline{c}) \circ (f\overline{b} * f\overline{a})\}\rangle$;
$\langle \emptyset, \{X/\star\langle\langle \overline{a}, \overline{b}\rangle, \langle \overline{d}, \overline{c}\rangle\rangle,\ Y/\star\langle\langle\langle \overline{a}, \overline{b}\rangle, \overline{c}\rangle, \overline{d}\rangle\}\rangle$;
$\langle \emptyset, \{X/\star\langle\langle\langle \overline{a}, Z\rangle, \langle \overline{b}, Z\rangle\rangle, \langle\langle \overline{d}, Z\rangle, \langle \overline{c}, Z\rangle\rangle\rangle,\ Y/\star\langle\langle\langle g\langle \overline{a}, Z\rangle, g\langle \overline{b}, Z\rangle\rangle, g\langle \overline{c}, Z\rangle\rangle, g\langle \overline{d}, Z\rangle\rangle\}\rangle$;
$\langle \emptyset, \{X/(\overline{d} * \overline{b}) \circ (\overline{c} * \overline{a}), Y/\star\langle\langle\langle g\langle \overline{a}, Z\rangle, g\langle \overline{b}, Z\rangle\rangle, g\langle \overline{c}, Z\rangle\rangle, g\langle \overline{d}, Z\rangle\rangle\}\rangle$;

$\langle\emptyset,\{X/\star\langle\langle\overline{a},\overline{b}\rangle,\langle\overline{d},\overline{c}\rangle\rangle,\ Y/\star\langle\langle\langle\overline{a},\overline{b}\rangle,\overline{c}\rangle,\overline{d}\rangle\}\rangle$;
$\langle\emptyset,\{X/\star\langle\langle\langle\overline{a},Z\rangle,\langle\overline{b},Z\rangle\rangle,\langle\langle\overline{d},Z\rangle,\langle\overline{c},Z\rangle\rangle\rangle,\ Y/\star\langle\langle\langle g\langle\overline{a},Z\rangle,g\langle\overline{b},Z\rangle\rangle,g\langle\overline{c},Z\rangle\rangle,g\langle\overline{d},Z\rangle\rangle\}\rangle$; *are in* $\mathcal{U}_{AC}(\mathcal{Q})$.

Remark 7.4. *Considering FP AC problems and AC-matchers, instead of FP C problems and C-matchers, the proofs of Cors. 7.1 and 7.2 are trivially adapted, respectively, from the proofs of Cors. 6.1 and 6.2. Then these new proofs are omitted.*

Corollary 7.1 (Soundness and completeness of gen. sol. for a variable in AC FP problems)**.** *Let* $\mathcal{P}=\langle\Delta,\mathcal{X},P\rangle$ *be an AC FP problem. Any solution in* $[X]_{\mathcal{P}_G}$ *is a solution of each AC FP equation for* X *in* $\mathcal{P}$. *If* $\langle\nabla,\{X/s\}\rangle$ *is a solution for each AC FP equation for* X *in* $\mathcal{P}$ *then there exists* $\langle\nabla',\{X/X\delta\}\rangle\in[X]_{\mathcal{P}_G}$ *such that* $\langle\nabla',\{X/X\delta\}\rangle\preceq\langle\nabla,\{X/s\}\rangle$.

Corollary 7.2 (Soundness and completeness of generated solutions for AC FP problems)**.** *Let* $\mathcal{P}$ *be an AC FP problem. Any solution in the set of solutions* $[\mathcal{P}]_{Sol_G}$ *is a correct solution of* $\mathcal{P}$. *For any* $\langle\nabla,\delta\rangle$ *solution of* $\mathcal{P}$ *there exist a pair* $\langle\nabla',\sigma\rangle\in[\mathcal{P}]_{Sol_G}$ *such that* $\langle\nabla',\sigma\rangle\preceq\langle\nabla,\delta\rangle$.

Remark 7.5. *Given Lems. 7.3 and 7.4, and Cor. 7.2, and since the simplification rules of nominal unification modulo AC has AC FP problems as successful leaves, where each of these problems may have infinite solutions, one concludes that nominal unification modulo AC is also at least infinitary.*

Chapter 8

Conclusion and future work

Formalisations in Coq and OCaml implementations of nominal A, C and AC equality-checking (Chap. 4), C-matching and C-unification (Chap. 5) were developed. Specially for equality-checking, the soundness of the $\approx_{\{A,C,AC\}}$ was proved and an algorithm was automatically extracted from the specification. Execution tests were performed comparing the extracted implementation with an improved one. The development of these algorithms allowed proposing upper bounds: for the syntactic case with A symbols; for the case that includes A and C symbols; and, for the case that include A, C and AC symbols.

The nominal C-unification algorithm that transforms an input problem $\mathcal{P}$ into a family of sets of FP problems was proved sound and complete. In Chap. 6, it is showed that each of these sets may have infinite solutions in the standard presentation as pairs of the form $\langle \nabla, \sigma \rangle$. A sound and complete generator of solutions for nominal C FP problems was provided. Also a sound and complete nominal C-matching algorithm was obtained instantiating a set of protected variables with the right-hand side variables of the input problem $\mathcal{P}$. Nominal C-matching has been showed finitary. Moreover, an extension to nominal C and AC-unification and nominal A, C and AC matching was presented in Chap. 7, where it has been showed that FP problems with AC function symbols may also have infinite solutions. From this, one concludes that nominal C and nominal AC-unification are both in the class of at least infinitary problems.

Since there exists a correspondence between higher-order patterns and nominal unification, an interesting topic of future work is the comparison between these two problems in the presence of AC function symbols. Intuitively, nominal AC-unification seems be infinitary, however higher-order patterns unification modulo AC is known as being of type 0 (*zero*) [18, 59]. If one presents solutions as pairs of fixed-point constraints and substitutions [10], nominal C-unification returns to the finitary class. This suggests that using this representation in nominal AC-unification would also bring it to the finitary class.

Other subjects of future work are the development of more efficient implementations for nominal equality-checking, matching and unification modulo algorithms. These implementations would make use of other data structures to represent nominal terms, permutations, freshness, substitutions, nominal constraints, etc. Also, efficient strategies for simplifying A and AC equations in unification and matching should be investigated. The process of automatic code extraction presented in Sec. 4.5 could be applied to the formalised unification and matching algorithms. These algorithms should be specified through well-founded recursive definitions operating over trees whose nodes are labelled with quadruples. The equivalence between this new specification and the inductive one, provided by equ_sys, fresh_sys, unif_step and match_step (resp., Defs. 5.4, 5.5, 5.9 and 5.13), should be proved. One method to prove this equivalence would be checking that each branch of a tree obtained by the recursive algorithm corresponds to a unif_step (or a match_step) reduction. For further developments, the use of other proof assistants such as PVS or Isabelle/HOL should be considered, as well as the adoption of a more direct strategy in the formalisation of the algorithms. This strategy would consist in proving sound and completeness directly from recursive definitions.

It remains to be formalised the algorithms of nominal C and AC-unification and nominal A, C and AC matching, as well as the combinatorial part for FP problems presented in Chaps. 6 and 7. Both formalisations should be non-trivial, since the former relies on the exponential possible associations on the *lhs* and *rhs* arguments of the AC function symbols that occurs in the equations, and the former would be based on algebraic properties of k-cycles.

Another interesting topic of future work is to explore how nominal syntax could affect the Manakin's [50] proof of the decidability of first-order A-unification. Also, investigating A-unification in the particular cases where it is finitary is an interesting subject of future work. For instance, problems without atom terms that play the role of constants could simulate A-*elementary unification* in the nominal syntax.

It is also matter of interest, the extension of these algorithms to other equational theories such as those that include D, U and I. Finally, applications to nominal rewriting and narrowing and types in the nominal syntax are interesting topics. Specially, DU theories could be applied to the di Cosmo's system [34]. Also nominal anti-unification and nominal disunification modulo equational theories are interesting subjects of future work.

References

[1] T. Aoto and K. Kikuchi. *Nominal Confluence Tool*. In *Proc. of the 8th Int. Joint Conf.: Automated Reasoning (IJCAR)*, volume 9706 of *LNCS*, pages 173–182. Springer, 2016. 4

[2] A. B. Avelar, A. L. Galdino, F. L. C. de Moura, and M. Ayala-Rincón. First-order unification in the PVS proof assistant. *Logic Journal of the IGPL*, 22(5):758–789, 2014. 5

[3] M. Ayala-Rincón, W. Carvalho-Segundo, M. Fernández, and D. Nantes-Sobrinho. *Nominal C-Unification*. In *Proc. of the 27th Int. Symp. Logic-Based Program Synthesis and Transformation (LOPSTR)*, volume 10855 of *LNCS*, pages 235–251. Springer, 2017. 5, 7, 72, 107

[4] M. Ayala-Rincón, W. Carvalho-Segundo, M. Fernández, and D. Nantes-Sobrinho. *On Solving Nominal Fixpoint Equations*. In *Proc. of the 11th Int. Symp. on Frontiers of Combining Systems (FroCoS)*, volume 10483 of *LNCS*, pages 209–226. Springer, 2017. 7, 107

[5] M. Ayala-Rincón, W. de Carvalho Segundo, M. Fernández, and D. Nantes-Sobrinho. A formalisation of nominal α-equivalence with A and AC function symbols. *ENTCS*, 332:21–38, 2017. 6, 44

[6] M. Ayala-Rincón, W. de Carvalho Segundo, M. Fernández, and D. Nantes-Sobrinho. A Formalisation of Nominal α-equivalence with A, C, and AC Function Symbols. *TCS*, 2019. Accepted manuscript in Theoretical Computer Science. 6, 44

[7] M. Ayala-Rincón, M. Fernández, W. Carvalho-Segundo, and D. Nantes-Sobrinho. A Formalisation of Nominal C-Matching through Unification with Protected Variables. In *Pre-proc. of Logical and Semantic Frameworks with Applications (LSFA)*, to appear in *ENTCS*, pages 29–41. Elseviser, 2018. 7, 72

[8] M. Ayala-Rincón, M. Fernández, M. J. Gabbay, and A. C. Rocha Oliveira. Checking Overlaps of Nominal Rewriting Rules. *ENTCS*, 323:39–56, 2016. 4

[9] M. Ayala-Rincón, M. Fernández, and D. Nantes-Sobrinho. *Nominal Narrowing*. In *Proc. of the 1st Int. Conf. on Formal Structures for Computation and Deduction (FSCD)*, volume 52 of *LIPIcs*, pages 11:1–11:17. SDLZI, 2016. 6

[10] M. Ayala-Rincón, M. Fernández, and D. Nantes-Sobrinho. *Fixed-Point Constraints for Nominal Equational Unification*. In *Proc. of the 3rd Int. Conf. on Formal Structures

for Computation and Deduction (FSCD), volume 108 of *LIPIcs*, pages 7:1–7:16. SDLZI, 2018. 7, 148

[11] M. Ayala-Rincón, M. Fernández, and A. C. Rocha-oliveira. *Completeness in PVS of a Nominal Unification Algorithm*. *ENTCS*, 323:57–74, 2016. 5, 32

[12] M. Ayala-Rincón, M. Fernández, A. C. Rocha-Oliveira, and D. L. Ventura. Nominal essential intersection types. *Theoretical Computuper Science*, 737:62–80, 2018. 5

[13] B. Aydemir, A. Bohannon, and S. Weirich. Nominal Reasoning Techniques in Coq. *ENTCS*, 174(5):69–77, 2007. 6

[14] F. Baader and T. Nipkow. *Term Rewriting and* All That. Cambridge UP, 1998. 19

[15] F. Baader and W. Snyder. Unification Theory. In *Handbook of Automated Reasoning (in 2 volumes)*, pages 445–532. Elsevier and MIT Press, 2001. 3

[16] A. Baumgartner, T. Kutsia, J. Levy, and M. Villaret. Nominal Anti-Unification. In *Rewriting Techniques and Applications, (RTA)*, pages 57–73. SDLZI, 2015. 4

[17] D. Benanav, D. Kapur, and P. Narendran. Complexity of Matching Problems. *J. of Sym. Computation*, 3(1/2):203–216, 1987. 3, 22, 62, 71

[18] A. Boudet and E. Contejean. AC-Unification of Higher-Order Patterns. In *Proc. of Principles and Practice of Constraint Programming*, volume 1330 of *LNCS*, pages 267–281. Springer, 1997. 148

[19] A. Boudet, E. Contejean, and H. Devie. A New AC Unification Algorithm with an Algorithm for Solving Systems of Diophantine Equations. *Logic in Computer Science*, pages 289–299, 1990. 3

[20] T. Braibant and D. Pous. *Tactics for Reasoning Modulo AC in Coq*. In *In Proc. of the 1st. Int. Conf. on Certified Programs and Proofs (CPP)*, volume 7086 of *LNCS*, pages 167–182. Springer, 2011. 6

[21] K. F. Brandt, A. Schlichtkrull, and J. Villadsen. Formalization of First-Order Syntactic Unification. In *Pre-proc. of the 32nd Int. Workshop on Unification (UNIF)*, 2018. 5

[22] W. E. Byrd and D. P. Friedman. αKanren: A Fresh Name in Nominal Logic Programming. *In Proc. of the Workshop on Scheme and Functional Programming*, pages 79–90, 2007. 4

[23] C. F. Calvès. *Complexity and implementation of nominal algorithms*. PhD Thesis, King's College London, 2010. 4

[24] C. F. Calvès and M. Fernández. Matching and Alpha-Equivalence Check for Nominal Terms. *J. of Computer and System Sciences*, 76(5):283–301, 2010. 4, 64, 69, 70

[25] C. F. Calvès and M. Fernández. *The First-order Nominal Link*. In *Proc. of the 20th Int. Symp. Logic-based Program Synthesis and Transformation (LOPSTR)*, volume 6564 of *LNCS*, pages 234–248. Springer, 2011. 4, 31

[26] J. Cheney. *αProlog User's Guide & Language Reference Version 0.3 DRAFT*, 2003. 4

[27] J. Cheney. Relating nominal and higher-order pattern unification. *Proceedings of the 19th international workshop on Unification (UNIF 2005)*, 2005. 4

[28] J. Cheney. *Equivariant unification*. *J. of Autom. Reasoning*, 45(3):267–300, 2010. 4

[29] M. Clausen and A. Fortenbacher. Efficient Solution of Linear Diophantine Equations. *J. of Sym. Computation*, 8(1-2):201–216, 1989. 3

[30] E. Contejean. *A Certified AC Matching Algorithm*. In *Proc. of the 15th Int. Conf. on Rewriting Techniques and Applications (RTA)*, volume 3091 of *LNCS*, pages 70–84. Springer, 2004. 6, 133

[31] E. Copello, E. Tasistro, N. Szasz, A. Bove, and M. Fernández. *Principles of Alpha-Induction and Recursion for the Lambda Calculus in Constructive Type Theory*. *ENTCS*, 323:109–124, 2016. 6

[32] CoqTeam. The Coq Proof Assistant Reference Manual, 2019. 32

[33] T. H. Cormen, C. E. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2009. 71

[34] R. Di Cosmo. Second Order Isomorphic Types: A Proof Theoretic Study on Second Order lambda-Calculus with Surjective Paring and Terminal Object. *Inf. Comput.*, 119(2):176–201, 1995. 149

[35] F. Durán, S. Eker, S. Escobar, N. Martí-Oliet, J. Meseguer, and C. L. Talcott. Associative unification and symbolic reasoning modulo associativity in maude. In *Proc. of Rewriting Logic and Its Applications (WLRA)*, volume 11152 of *LNCS*, pages 98–114. Springer, 2018. 6

[36] S. M. Eker. Associative-Commutative Matching Via Bipartite Graph Matching. *The Computer J.*, 38:381–399, 1995. 3

[37] S. M. Eker. Associtative-Commutative Rewriting on Large Terms. In *Rewriting Techniques and Applications, (RTA)*, volume 2706 of *LNCS*, pages 14–29, 2003. 3

[38] François Fages. Associative-Commutative Unification. *J. of Sym. Computation*, 3:257–275, 1987. 3, 22

[39] E. Fairweather, M. Fernández, N. Szasz, and A. Tasistro. Dependent Types for Nominal Terms with Atom Substitutions. In *Int. Conf. on Typed Lambda Calculi and Applications, (TLCA)*, pages 180–195. SDLZI, 2015. 5

[40] M. Fernández and M. Gabbay. Curry-Style Types for Nominal Terms. In *Int. Work. on Types for Proofs and Programs (TYPES)*, volume 4502 of *LNCS*, pages 125–139. Springer, 2006. 5

[41] M. Fernández and M. J. Gabbay. *Nominal Rewriting*. *Information and Computation*, 205(6):917–965, 2007. 4, 5

[42] M. J. Gabbay and A. M. Pitts. *A New Approach to Abstract Syntax with Variable Binding*. *Formal Aspects of Computing*, 13(3-5):341–363, 2002. 3

[43] D. Kapur and P. Narendran. NP-Completeness of the Set Unification and Matching Problems. In *8th International Conference on Automated Deduction (CADE)*, volume 230 of *LNCS*, pages 489–495. Springer, 1986. 3, 22

[44] D. Kapur and P. Narendran. *Matching, Unification and Complexity*. *SIGSAM Bulletin*, 21(4):6–9, 1987. 3, 22

[45] D. Kapur and P. Narendran. Complexity of Unification Problems with Associative-Commutative Operators. *J. of Autom. Reasoning*, 9(2):261–288, 1992. 3, 22

[46] R. Kumar and M. Norrish. *(Nominal) Unification by Recursive Descent with Triangular Substitutions*. In *Proc. of the 1st Int. Conf. of Interactive Theorem Proving (ITP)*, volume 6172 of *LNCS*, pages 51–66. Springer, 2010. 5

[47] D. Larchey-Wendling and J.-F. Monin. Simulating Induction-Recursion for Partial Algorithms. Accepted to TYPES. Available at `https://members.loria.fr/DLarchey/files/papers/TYPES_2018_paper_19.pdf`, 2018. 61

[48] J. Levy and M. Villaret. *An Efficient Nominal Unification Algorithm*. In *Proc. of the 21st Int. Conf. on Rewriting Techniques and Applications (RTA)*, volume 6 of *LIPIcs*, pages 209–226. SDLZI, 2010. 4, 31

[49] J. Levy and M. Villaret. Nominal unification from a higher-order perspective. *ACM Trans. Comput. Log.*, 13(2):10:1–10:31, 2012. 4

[50] G. S. Makanin. The Problem of Solvability of Equations in a Free Semigroup. *Math. USSR Sbornik*, 32(2):129–198, 1977. 17, 22, 149

[51] A Martelli and U. Montanari. Unification in linear time and space: A structured presentation. Technical report, Ist. di Elaborazione delle Informazione, Consiglio Nazionale delle Ricerche, Pisa, Italy, 1976. Internal Rep. B76-16. 3, 16, 22

[52] A. Martelli and U. Montanari. An Efficient Unification Algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, 1982. 16

[53] D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. Log. Comput.*, 1(4):497–536, 1991. 4

[54] T. Nipkow. *Equational Reasoning in Isabelle*. *Science of Computer Programming*, 12(2):123–149, 1989. 6

[55] T. Nipkow. Functional unification of higher-order patterns. In *Proc. of the Eighth Annual Symp. on Logic in Computer Science (LICS)*, pages 64–74. IEEE, 1993. 4

[56] M. S. Paterson and M. N. Wegman. Linear Unification. *J. of Computer and System Sciences*, 16(2):158–167, 1978. 3, 4, 16, 22

[57] A. M. Pitts. Nominal Logic, a First Order Theory of Names and Binding. *Information and Computation*, 186(2):165–193, 2003. 3, 22

[58] A. M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge UP, 2013. 3

[59] Z. Qian and K. Wang. Modular AC unification of higher-order patterns. In *Proc. of the First Int. Conf. Constraints in Computational Logics*, volume 845 of *LNCS*, pages 105–120. Springer, 1994. 148

[60] J. A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *J. of the ACM*, 12(1):23–41, 1965. 3, 13, 22

[61] A. C. Rocha-Oliveira. *Unificação, confluência e tipos com interseção para sistemas de reescrita nominal*. PhD thesis, Universidade de Brasília (UnB), 2016. 5, 32

[62] Sagan, B. E. *The Symmetric Group: Representations, Combinatorial Algorithms, and Symmetric Functions*. Springer, 2nd edition, 2001. 107

[63] M. Schmidt-Schauß, T. Kutsia, J. Levy, and M. Villaret. *Nominal Unification of Higher Order Expressions with Recursive Let*. In *Post-proc. of the 26th Int. Sym. on Logic-Based Program Synthesis and Transformation (LOPSTR 2016)*, volume 10184 of *LNCS*, pages 328–344. Springer, 2017. 7

[64] J. H. Siekmann. *Unification of Commutative Terms*. In *Proc. of the Int. Symposium on Symbolic and Algebraic Manipulation*, volume 72 of *LNCS*, pages 22–29. Springer, 1979. 3, 18

[65] M. Sozeau. Subset Coercions in Coq. In *Proc. of the Int. Work. on Types for Proofs and Programs (TYPES)*, volume 4502 of *LNCS*, pages 237–252. Springer, 2006. 61

[66] M. Sozeau. Equations: A Dependent Pattern-Matching Compiler. In *Proc. of the 1st Int. Conf. of Interactive Theorem Proving (ITP)*, volume 6172 of *LNCS*, pages 419–434. Springer, 2010. 58

[67] M. E. Stickel. A Unification Algorithm for Associative-Commutative Functions. *J. of the Association for Computing Machinery*, 28(3):423–434, 1981. 3, 19, 22

[68] E. Tiden and S. Arnborg. Unification Problems with One-Sided Distributivity. *J. of Sym. Computation*, 3:183–202, 1987. 22

[69] C. Urban. Nominal Techniques in Isabelle/HOL. *J. of Autom. Reasoning*, 40(4):327–356, 2008. 5

[70] C. Urban. *Nominal Unification Revisited*. In *Proc. of the 24th Int. Work. on Unification (UNIF)*, volume 42 of *EPTCS*, pages 1–11, 2010. 5, 32

[71] C. Urban and C. Kaliszyk. General Bindings an Alpha-Equivalence in Nominal Isabelle. *Logical Methods in Computer Science*, 8:1–35, 2012. 6

[72] C. Urban, A. M. Pitts, and M. J. Gabbay. *Nominal Unification*. *Theoretical Computer Science*, 323(1-3):473–497, 2004. 3, 4, 5, 26, 31, 32

[73] Z. Qian. Linear Unification of Higher-Order Patterns. *Theory and Practice of Software Development*, pages 391–405, 1993. 4

Appendix A

The hierarchy of the Coq formalisation is presented in the diagram of Fig. A.1. Below, for each file of the formalisation, a short description is given:

- `Basics.v` - Necessary results on arithmetics and lists that are not in the standard libraries of Coq;
- `Terms.v` - The nominal syntax of terms (see Fig. 3.1);
- `Perm.v` - Permutation action over atoms and terms (see Defs. 2.19 and 2.20);
- `Disagr.v` - The disagreement sets of permutations (see Def. 2.21);
- `Tuples.v` - The operators for obtaining the number of arguments, selecting an element and deleting an element of a tuple, which are given, respectively, by the recursive definitions `TPlength`, `TPith` and `TPithdel` (see Figs. 4.1 to 4.3 and Lem. 4.1);
- `Fresh.v` - The freshness relation given by the inductive definition **fresh** (see Fig. 3.2);
- `w_Equiv.v` - The auxiliary nominal "weak" α-equivalence (see Fig. 3.5);
- `Alpha_Equiv_old.v` - Proof of the soundness of the nominal α-equivalence (see Fig. 3.3 and Lems. 3.3, 3.10 and 3.11) according to Urban's weak α-equivalence Isabelle/HOL approach;
- `Alpha_Equiv.v` - Proof of the soundness of the nominal α-equivalence (see Fig. 3.3 and Lems. 3.3, 3.16 and 3.17) according to Rocha-Oliveira's PVS direct approach;
- `Equiv.v` - The inductive definition **equiv** that extensions nominal α-equivalence with A, C and AC function symbols (see Fig. 4.8);
- `AACC_Equiv_rec.v` - Recursive versions of **fresh** and **equiv** with an automatic executable OCaml code extraction (see Sec. 4.5);

- Equiv_Tuples.v - Results about **equiv** and operators TPlength, TPith and TPithdel (see Lem. 4.7);
- AACC_Equiv.v - Soundness of **equiv** (see Lems. 4.6, 4.8 and 4.9 and Cor. 4.1);
- C_Equiv.v - Specific lemmas about the nominal α, C-equivalence (see Lems. 4.2 to 4.5);
- Problems.v - Definitions of unification problem and set of solutions (see Defs. 5.1 and 5.2);
- Substs.v - Substitutions and lemmas about the α, C-equivalence and composition of substitutions (see Lem. 5.6);
- C_Unif.v - Transformation systems for simplifying freshness constraints and equations ($\Rightarrow_{\#}$ and $\Rightarrow_{\approx}$) which are the basis of the definition of the nominal C-unification algorithm (see Figs. 5.5 and 5.5);
- C_Unif_Termination.v - Termination of $\Rightarrow_{\#}$ and $\Rightarrow_{\approx}$ (see Lem. 5.2);
- C_Unif_Soundness.v - Soundness of $\Rightarrow_{\#}$ and $\Rightarrow_{\approx}$ (see Thm. 5.1);
- C_Unif_Completeness.v - Completeness of $\Rightarrow_{\#}$ and $\Rightarrow_{\approx}$ (see Thm. 5.2);
- C_Matching.v - The nominal C-matching algorithm and its termination, soundness and completeness (see Sec. 5.2).

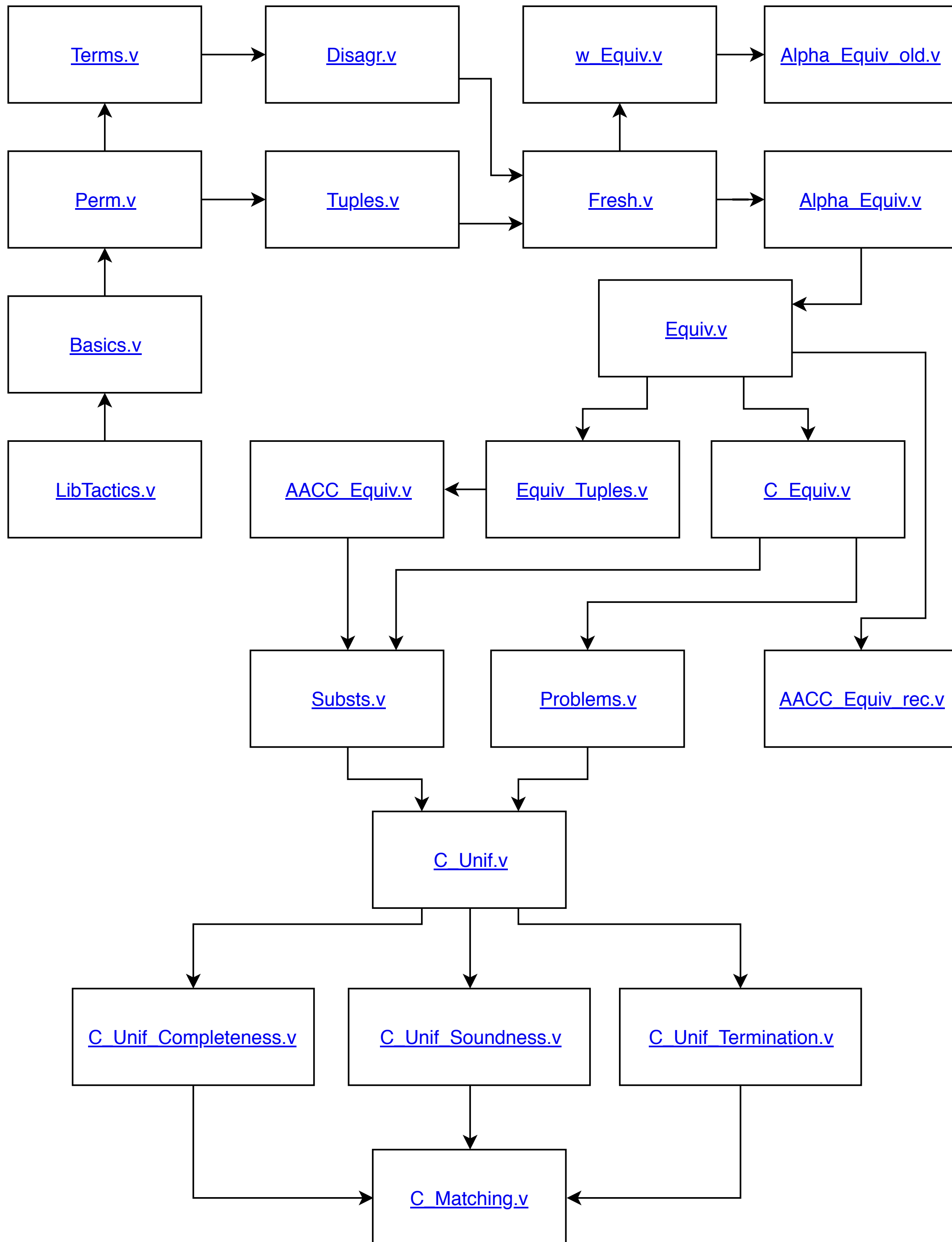

Figure A.1: Dependencies diagram of the files of the Coq formalisation.

Index

$\Rightarrow_{\#}$ and $\Rightarrow_{\approx}$, 73
$\approx_{\{A,C,AC\}}$ equality-checking implementations, 54
$\approx_{\{A,C,AC\}}$ execution tests, 61
α and A tests, 65
α tests, 64
α, A and C tests, 66
α, A, C and AC equality-checking, nominal, 43
α, A, C and AC tests, 67
α-equivalence rules, nominal, 24
α-equivalence, nominal, 23
$(\approx_? \mathbf{A})$, 132
$(\approx_? \mathbf{AC})$, 132

A, C and AC-unification, first-order, 16
A-unification, first-order, 16
A, C and AC-unification and matching, nominal, 131
abstraction, nominal term, 22
AC-unification, first-order, 19
algorithm for checking $\approx_{\{A,C,AC\}}$, 55
application, nominal term, 22
arity, function symbol, 10
atom, 22
atom, nominal term, 22
atoms of interest in C FP problems on a variable, 128

basic properties of $\mathcal{M} = (a_{ij})_{k-1\times k}$, 110
bracketing, 131

C-matching, nominal, 92
C-unification and matching, nominal, 71
C-unification with protected variables, nominal, 72
C-unification, first-order, 17
characterisation of $\mathcal{M} = (a_{ij})_{k-1\times k}$, 112
combinatorial solutions, 114
complete set, first-order solutions, 13
completeness of $\Rightarrow_{(\approx_? \mathbf{A})}$ and $\Rightarrow_{(\approx_? \mathbf{AC})}$, 138
completeness of $\mathcal{M} = (a_{ij})_{k-1\times k}$, 112
completeness of solutions for singleton AC FP problems, 142
completeness of solutions for singleton C FP problems, 121
complexities, first-order problems, 22
complexity, nominal unification, 30
composition, substitution, 12
constant, 10
constraints, nominal, 25
Coq proof assistant, 31

derivation tree, 77
difference set, 24
domain, substitution, 12

empty tuple, nominal term, 22
equality judgement, 24
equality-checking, first-order, 12
equational property, 10
extended pseudo-cycle, 115
extended pseudo-cycle correspondence for π and π^2, 119
extended Pseudo-cycle with C and AC function symbols, 139

finitary (ω), type, 13
first-instance pseudo-cycle matrix, 109
formalisation approaches, comparing, 41
freshness, 23
freshness context, 23

freshness judgement, 24
freshness rules, 24
function symbol, 10

general AC-matchers, 143
general C-matchers, 124
general solutions for C FP problems, 114
generated solutions for a variable, 124
generated solutions for AC FP problems, 144
generated solutions of singleton C FP problems, 118

hierarchy of the Coq formalisation, 152

image, substitution, 12
improvements in the generation of solutions for C FP problems, 127
independent solutions, 13
infinitary (∞), type, 13

minimal-complete, first-order solutions, 13
more general first-order solution, 13
more general nominal C solution, 114
more general nominal solution, 27

nominal C-unification, OCaml implementation, 78
nominal constraints, 25
nominal fixed point problems, 106
nominal syntax, 22
nominal triple, 25
nominal unification, termination, 27
NP-completeness, first-order C-unification, 18
NP-completeness, nominal C-unification, 91

pair, nominal term, 22
permutation, 22
permutation action, atoms, 23
permutation action, terms, 23
permutation cycles in the top of Π_X, 128
permutation factor, 127
preservation of solutions by $\Rightarrow_{(\approx_? \mathbf{A})}$ and $\Rightarrow_{(\approx_? \mathbf{AC})}$, 137
problem, first-order, 11
problem, syntactic first-order, 11
protected variables, nominal C-matching, 92
pseudo-cycle, 106, 107
pseudo-cycle C equivalence, 108

reduction strategy, nominal unification, 28
rewriting notation, first-order, 14
rewriting notation, nominal, 28
rules for nominal A, C and AC problems, 131

set of variables of a pair, 13
set size, 11
signature, 10
size, set of nominal constraints, 27
solution for a nominal C-matching problem, 93
solution for quadruples and unification problems, nominal, 72
solution, first-order unification/matching, 12
solution, nominal, 26
solutions for nominal AC FP problems, 138
sorting, 131
soundness and completeness of generated solutions for a variable, 125
soundness and completeness of generated solutions for a variable in AC FP problems, 144
soundness and completeness of generated solutions for AC FP problems, 144
soundness and completeness of generated solutions for C FP problems, 126
soundness of solutions for singleton AC FP problems, 142
soundness of solutions of singleton C FP problems, 118
specification & formalisation, 31
  ($\approx_\alpha$ **A**), 47
  ($\approx_\alpha$ **AC**), 48
  ($\approx_\alpha$ **C**), 47
  α equality-checking, 31
  ($\approx_\alpha$ $\overline{\mathbf{app}}$), 47

$\mathcal{U}_C$ and $\mathcal{M}_C$ equivalence, 93
NF, 74
equ_sys, 75
equiv_rec, 59
fresh_rec, 58
fresh_sys, 76
iter, 58
match_leaf, 93
match_path, 93
match_step, 92
tr_clos, 74
equiv, 43, 48, 49
leaf, 88
unif_path, 88
unif_step, 88
TPithdel, 44, 45
TPith, 44
TPlength, 44
alpha_equiv, 34
fresh, 33
w_equiv, 37
alpha-equivalence, 34
automatic code extraction, 57
basic properties of freshness, 39
basic properties over permutations with $ds(\pi, \pi') = \emptyset$, 39
C-unification, nominal, 71
characterisation of successful leaves, 84
characterisation of successful matching leaves, 97
combination of AC arguments, 52
completeness of $\Rightarrow_\mu$, 95
completeness of $\Rightarrow_\mu^*$, 96
completeness of $\mathcal{T}_{\langle\Delta,\mathcal{X},P\rangle}$, 86
correctness of equiv_rec, 60
decidability of $\Rightarrow_\mu$, 94
equivalence of $\sim_\omega$, 36
equivalence of **equiv**(S), for $S \subseteq \{0,1,2\}$, 53
equivariance of $\approx_\alpha$ through $\sim_\omega$, 37
equivariance of $\approx_{\{A,C,AC\}}$, 51
equivariance of $\approx_\alpha$ by the direct approach, 40
equivariance of $\sim_\omega$, 36
extension of $\approx_\alpha$, 47
freshness, 33
freshness preservation of $\approx_\alpha$, 33
freshness preservation of $\sim_\omega$, 36
freshness preservation under $\approx_{\{A,C,AC\}}$, 51
intermediate transitivity for $\approx_\alpha$ with $\sim_\omega$, 36
intermediate transitivity for $\approx_{\{A,C,AC\}}$ with $\approx_\alpha$, 51
intersection emptyness preservation with *rhs* variables by $\Rightarrow_\mu$, 94
invariance under $\approx_\alpha$, 33
inversion of permutations over $\approx_\alpha$, 40
main properties of $\Rightarrow_\upsilon$, 88
matching-step, 92
nominal syntax, 32
permutation, 32
preservation of `Rvar` by $\Rightarrow_\mu$, 94
preservation of solutions by $\Rightarrow_\approx$, 81
preservation of solutions by $\Rightarrow_\#$, 79
preservation of solutions by $\Rightarrow_\mu$, 95
preservation of valid quadruples by $\Rightarrow_\mu$, 94
reflexivity of $\approx_\alpha$, 35
reflexivity of $\approx_{\{A,C,AC\}}$, 52
reverse of $\approx_{\{A,C,AC\}}$, 50
second intermediate transitivity for $\approx_\alpha$, 37
soundness of $\approx_{\{A,C,AC\}}$, 50
soundness of $\Rightarrow_\mu^*$, 96
Soundness of $\mathcal{T}_{\langle\Delta,\mathcal{X},P\rangle}$, 85
symmetry of $\approx_\alpha$ by the direct approach, 40
symmetry of $\approx_\alpha$ through $\sim_\omega$, 38
symmetry of $\approx_{\{A,C,AC\}}$, 53
termination of $\Rightarrow_\mu$, 94
termination of $\Rightarrow_\approx$ and $\Rightarrow_\#$, 77
transitivity of $\approx_\alpha$ by the direct approach, 41

transitivity of $\approx_\alpha$ through $\sim_\omega$, 38
transitivity of $\approx_{\{A,C,AC\}}$, 52
tuples, operations over, 44
tuples, properties the operators over, 46
unification-step, 88
valid quadruples, preservation, 76
weak alpha-equivalence, 37
substitution, 12
substitution action, first-order problem, 12
substitution action, first-order term, 12
substitution action, nominal term, 26
substitutions and permutations commute, 26
successful leaves, 84
suspension, nominal term, 22
swapping, 22
syntactic function symbol, 10
syntax, first-order, 10

term size, first-order, 11
term size, nominal, 23
term, first-order, 10
term, nominal, 23
term, syntactic first-order, 10
termination of $\Rightarrow_{(\approx_?\mathbf{A})}$ and $\Rightarrow_{(\approx_?\mathbf{AC})}$, 136
termination, first-order C-unification, 17
termination, first-order syntactic unification, 13
triple, nominal, 25
type classification, 13
type classification, first-order problems, 22
type, first-order C-unification, 18
type, nominal unification, 30

unification algorithm, first-order, 14
unification rules, equations, nominal, 27
unification rules, equations, nominal C, 73
unification rules, freshness constraints, nominal, 28
unification rules, freshness constraints, nominal C, 74
unification, first-order, 10
unification, nominal, 25
unification, Robinson algorithm, 14
unification, syntactic first-order, 13
unitary (1), type, 13
upper bounds for $\approx_{\{A,C,AC\}}$, 63, 68

valid quadruple, 76
variables, set of, 11
variables, set of, nominal constraints, 27
variables, set of, nominal triple, 27

weak α-equivalence rules, 36

zero (0), type, 13