Open access • Book Chapter • DOI:10.1007/3-540-63104-6_4

On Equality Up-to Constraints over Finite Trees, Context Unification, and One-Step Rewriting — Source link

Joachim Niehren, Manfred Pinkal, Peter Ruhrberg

Institutions: Saarland University

Published on: 13 Jul 1997 - Conference on Automated Deduction

Topics: Unification, Context (language use), Rewriting, Decidability and Undecidable problem

On Equality Up-to Constraints over Finite Trees, Context Unification and One-Step Rewriting

Joachim Niehren, Manfred Pinkal, Peter Ruhrberg

On Equality Up-to Constraints over Finite Trees, Context Unification, and One-Step Rewriting

Joachim Niehren[1], Manfred Pinkal[2] and Peter Ruhrberg[2]

[1] Programming Systems Lab
[2] Department of Computational Linguistics
Universität des Saarlandes, Saarbrücken, Germany
niehren@ps.uni-sb.de and {pinkal,peru}@coli.uni-sb.de

Abstract. We introduce equality up-to constraints over finite trees and investigate their expressiveness. Equality up-to constraints subsume equality constraints, subtree constraints, and one-step rewriting constraints. We establish a close correspondence between equality up-to constraints over finite trees and context unification. Context unification subsumes string unification and is subsumed by linear second-order unification. We obtain the following three new results. The satisfiability problem of equality up-to constraints is equivalent to context unification, which is an open problem. The positive existential fragment of the theory of one-step rewriting is decidable. The $\exists^*\forall^*\exists^*$ fragment of the theory of context unification is undecidable.

Keywords tree constraints, subtree relation, string unification, context unification, linear second-order unification, one-step rewriting, semantic processing of natural language.

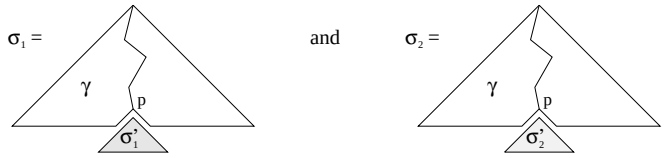
1 Introduction

Trees are widely used in computer science and computational linguistics. They serve as representation for all kinds of symbolic structures such as programs, proofs, data structures, syntactic and semantic analyses of natural language expressions. However, one often needs to represent structures which are only partially known, and relationships between several such partially known structures. For these purposes, it is convenient to use tree constraints. These are ordinary predicate logic formulae with variables denoting trees and with relation symbols interpreted by predefined relations over trees.

We introduce a new class of tree constraints, which we call equality up-to constraints over finite trees, and investigate their expressiveness. Equality up-to constraints subsume equality constraints, subtree constraints, and one-step rewriting constraints. We establish a close correspondence between equality up-to constraints and context unification. Context unification subsumes string unification and is subsumed by linear second-order unification. We obtain the following three new results. The satisfiability problem of equality up-to constraints is equivalent to context unification, which is an open problem. The positive existential frag-

ment of the theory of one-step rewriting is decidable. The $\exists^*\forall^*\exists^*$ fragment of the theory of context unification is undecidable.

Equality Up-to Constraints. The semantics of equality up-to constraints is based on the equality up-to relation over finite trees. Given finite trees σ_1,σ_1', σ_2, σ_2' the equality up-to relation $\sigma_1/\sigma_1'{=}\sigma_2/\sigma_2'$ holds if σ_1 is equal to σ_2 up-to one position p where σ_1 has the subtree σ_1' and σ_2 the subtree σ_2'.



An equality up-to constraint is a conjunction of expressions $s_1/s_1'{=}s_2/s_2'$ where s_1, s_1', s_2, s_2' are first-order terms with first-order variables.
Equality up-to constraints subsume equality constraints $s_1{=}s_2$ by equivalence to $s_1/s_1{=}s_2/s_1$ and subtree constraints $s_2{\ll}s_1$ meaning that the denotation of s_2 is a subtree of the denotation of s_1 by the equivalence $s_2{\ll}s_1 \leftrightarrow s_1/s_2{=}s_1/s_2$. Venkatamaran [28] proves that the $\exists^*\forall^*$ fragment of the theory of the subtree relation (including negation and equality) is undecidable. This implies that the $\exists^*\forall^*$ fragment of the theory of equality up-to constraints is undecidable.

Context Unification. A context is a tree with a hole. More precisely, a context with hole X is a first-order term s with a single occurrence of X and no occurrence of any other variable. We prefer to work with context functions rather than with contexts. A context function γ is a function from trees to trees that is defined by some equation of the form $\gamma(\sigma) = s[\sigma/X]$ where s is a context with hole X. Note that a context function is a second-order function that is linear in that it uses its argument exactly once.
We assume a set of second-order variables ranged over by C. A second-order term t is either a first-order variable X, a construction $a(t_1, \ldots, t_n)$ or an application $C(t)$. A context constraint is a conjunction of expressions $t_1{=}t_2$. A second-order variable denotes a context function whereas a second-order term denotes a tree. The denotation of a second-order term $C(t)$ is defined as the application of the denotation of C to the denotation of t. Context unification is the satisfiability problem of context constraints with respect to the above interpretation.

Correspondence. Context constraints plus existential quantification are at least as expressive as equality up-to constraints, since $s_1/s_1'{=}s_2/s_2'$ is equivalent to $\exists C\,(s_1{=}C(s_1') \wedge s_2{=}C(s_2'))$. Conversely, we can encode context constraints into equality up-to constraints (up to satisfaction equivalence) even if some context variables C occur more than twice. This fact is not obvious, and it is proved in the paper. The given correspondence has the following two consequences. The satisfiability problem of equality up-to constraints is equivalent to context unification. The $\exists^*\forall^*\exists^*$ fragment of the theory of context constraints is undecidable.

One-Step Rewriting. The first-order theory of one-step rewriting has attracted some attention starting with [3] because it allows to express several decidable properties of rewrite systems. The hope was that the whole theory could be decidable. Treinen [26], however, has shown that the $\exists^*\forall^*$ fragment of the first-order theory of one-step rewriting is undecidable. Several improvements of this result have been achieved [14, 29] where only restricted rewrite systems are needed.
Equality up-to constraints subsume one-step rewriting constraints. A one-step rewriting constraint is of the form $X_1 \to X_2$ **with** $s_1 \to s_2$ and means that the tree denotation of X_1 rewrites in one-step to the tree denotation of X_2 via an application of the rewrite rule $s_1 \to s_2$. Let $\{Y_1, \ldots, Y_n\}$ be the set of variables occuring in s_1 and s_2. If $\{X_1, X_2\} \cap \{Y_1, \ldots, Y_n\} = \emptyset$ then the one-step rewriting constraint $X_1 \to X_2$ **with** $s_1 \to s_2$ is equivalent to $\exists Y_1 \ldots \exists Y_n (X_1/s_1 = X_2/s_2)$. The condition $\{X_1, X_2\} \cap \{Y_1, \ldots, Y_n\} = \emptyset$ can always be assumed by renaming the variables in the rewrite rule $s_1 \to s_2$. The equality up-to constraint that we obtained $X_1/s_1 = X_2/s_2$ is satisfaction equivalent to the context constraint $X_1 = C(s_1) \wedge X_2 = C(s_2)$, which is a stratified in the sense of Schmidt-Schauß [22] because of the condition $\{X_1, X_2\} \cap \{Y_1, \ldots, Y_n\} = \emptyset$. The decidability of stratified context unification is proved in [22]. Hence, the positive existential fragment of the theory of one-step rewriting (constraints) is decidable.

Plan of the Paper. In Section 2, we discuss related work. In Section 3, we introduce equality up-to constraints, relate them to subtree constraints, and distinguish decidable fragments. In Section 4, we define context unification and formulate results in analogy to those for equality up-to constraints. In Section 5, we formulate the correspondence between equality up-to constraints and context unification. In Section 6, we relate to the first-order theory of one-step rewriting. In Section 7, we illustrate how to compute solutions of context constraints in several simple examples, based on an algorithm formulated in Appendix B of the full paper.
In the full version of the paper [16], three appendixes are added. In Appendix A, we give a proof omitted in the conference version of the paper. Appendix B presents a simple correct and complete but not necessary terminating algorithm for context unification (in Plotkin style). This algorithm is modified in Appendix C such as to obtain an algorithm for context unification in the style of Lévy's second-order unification algorithm. It is claimed in [12] that this algorithm terminates for all presented decidable fragments of linear second-order unification (and thus context unification). At the time of submitting this paper, however, the termination proof given there for the case of stratified linear second-order unification has not been agreed on to be complete.

2 Related Work

String Unification. String unification is the problem of solving word equations. For instance, all solutions of the word equation $Ca = aC$ map C to a word

described by the regular expression a^*. String unification is subsumed by context unification. The above word equation corresponds to the context constraint $C(a(\epsilon)) = a(C(\epsilon))$ where a is a function constant and ϵ is a fixed first-order constant representing the empty word. Every solution of this context constraint maps C to the context function γ with $\gamma(\sigma) = a(\ldots(a(\sigma)))$. The correspondence between γ and a^* is obvious.
String unification has been discovered and investigated by several independent research communities (for an overview see [2]). The notion of string unification stems from the field of automated deduction [21, 24], where it is also called A-unification [1] with a single associative function symbol. String unification has first been presented by Markov [15] in 1954 and is called Markov's problem by mathematicians in eastern countries. It is called Löb's problem by mathematicians in western countries, for example by A. Lentin and M.P. Schützenberger [11]. A solution to the string unification problem was found by Makanin [13] in 1977. Subsequent papers on this topic [18, 9, 23, 10] were concerned with finding a better description of Makanin's algorithm, closing small gaps in the proof of correctness, and studying its complexity.

Context Unification. Context unification is a subproblem of linear second-order unification (see below) and a generalization of string unification. The notion of context unification was first used in [12] but stems from [4] where it is called unification with context variables. A formal definition of context unification is given implicitly in [22]. To our knowledge, the decidability of context unification is still open (in contrast to string unification).
Three distinct fragments of context unification are solved in [4, 22, 12] respectively. Schmidt-Schauß [22] considers stratified context constraints mentioned above. Lévy [12] handles all context constraints where first-order and second-order variables occur at most twice. Comon's fragment [4] includes context constraints corresponding to equality and subtree constraints (see Section 4.2) but also context membership constraints, which express that a context C belongs to a given regular set of contexts. A typical formula of this fragment is (in the word case) $x{=}Ca_2 \wedge C{\in}(a_1^*a_2)^*$. It seems that nested Kleene stars cannot be expressed in terms of string unification. In contrast, a single Kleene star can be expressed, for instance $x{=}Ca_2 \wedge C{\in}a_1^*$ is equivalent to $x{=}Ca_2 \wedge Ca_1{=}a_1C$, which is however not a member of Comon's fragment, since C is applied with distinct arguments a_2 and a_1.

Linear Second Order Unification. Context unification can also be considered as a subproblem of linear second-order unification [12]. This is the problem of whether a conjunction of equations between second-order λ-terms in long $\beta\eta$ normal form has a solution that maps variables to linear second-order λ-terms. Lévy's algorithm is correct and complete for linear second-order unification but not always terminating. The decidability of linear second-order unification is open but as for context unification, three decidable fragments are known [12]. Note also that linear second-order unification is a subproblem of second-order

unification, which is undecidable [7] but only a fragment of higher-order unification [19, 8].

Ellipses in Natural Language. The motivation of the authors for the investigation of equality up-to constraints stems from the area of semantic processing of natural language. This line of research started with higher-order unification [5, 6] and led to a definition of linear second-order unification [20] independently from Lévy in [12]. An application of context unification for semantic processing of natural language is presented by the authors in [17].
Here, we give a linguistic toy example that illustrates how equality up-to constraints (and thus context unification) can be applied to the analysis of an elliptic sentence. Consider Peter likes Riesling, Chardonnay too. This sentence is composed of two subsentences Peter likes Riesling and Chardonnay too. The semantics of both subsentences are boolean values. The semantics of the word Peter is of type person, and the semantics of the words Riesling and Chardonnay are of type grape. The semantics of the word likes is some function of type person→(grape→bool). Given the semantics of all these words, the semantics of the subsentences can be represented by the following two trees respectively:

$$@(@(\mathsf{likes}\ \mathsf{Riesling})\ \mathsf{Peter}) \qquad \text{and} \qquad @(@(\mathsf{likes}\ \mathsf{Chardonnay})\ \mathsf{Peter})$$

More precisely, the semantics of the subsentences is obtained from the semantics of its words by evaluating the above trees where @ is interpreted as function application.
The above trees can by described as the solutions of the following equality up-to constraint with respect to the variables X_1 and X_2:

$$X_1 = @(@(\mathsf{likes}\ \mathsf{Riesling})\ \mathsf{Peter})) \ \wedge\ X_1/\mathsf{Riesling} = X_2/\mathsf{Chardonnay}$$

Such a constraint can be derived on the basis of a syntactic analysis of the sentences, and a resolution of the ellipsis that takes Riesling and Chardonnay to play a structurally parallel role in their respective contexts. Note that the semantics of the word too in the above sentence is described by the context function γ with $\gamma(\sigma) = @(@(\mathsf{likes}\ \sigma)\ \mathsf{Peter})$, but not by a tree.

3 Equality Up-to Constraints

We define the syntax and semantics of equality up-to constraints, relate them to subtree and equality constraints, and distinguish some decidable and undecidable fragments of their first-order theory.

3.1 Syntax and Semantics

We assume an infinite set of first-order variables ranged over by X, and a set of function constants ranged over by a, a_1 and a_2. Every function constant is equipped with an arity, which is an integer $n \geq 0$. For all undecidability results,

we assume that there is at least one constant of arity ≥ 2 and one constant of arity 0.
A first-order term s is either a first-order variable X or of the form $a(s_1, \ldots, s_n)$ where n is the arity of a and $s_1, \ldots, s_n$ are first-order terms. An *equality up-to constraint* is a conjunction of expressions of the form $\mathsf{eq}_2(s_1, s_1', s_2, s_2')$ that we write as $s_1/s_1'{=}s_2/s_2'$ for better readability. A (finite) tree σ is a ground first-order term, i.e. a first-order term without variables. A *context function* γ is a function from trees to trees that is described by an equation of the form

$$\gamma(\sigma) = s[\sigma/X] \qquad \text{for all trees } \sigma$$

where s is a first-order term that contains a single occurrence of the variable X and no occurrences of any other variable (i.e. s is a context with hole X).
The relation symbol eq_2 is interpreted as the following 4-ary relation between finite trees.

$$\sigma_1/\sigma_1'{=}\sigma_2/\sigma_2' \quad \text{iff} \quad \text{exists } \gamma \text{ such that } \sigma_1 = \gamma(\sigma_1') \text{ and } \sigma_2 = \gamma(\sigma_2')$$

Let α be a variable assignment that maps first-order variables to finite trees. We extend α homomorphically to first-order terms.

$$\alpha(a(s_1, \ldots, s_n)) = a(\alpha(s_1), \ldots \alpha(s_n))$$

An equality up-to constraint $s_1/s_1'{=}s_2/s_2'$ is *satisfiable* over finite trees if there exists a variable assignment α such that $\alpha(s_1)/\alpha(s_1'){=}\alpha(s_2)/\alpha(s_2')$ holds.

3.2 Subtree and Equality Constraints

Equality up-to constraints can to express *equality constraints* $s{=}s'$ and *subtree constraints* $s{\ll}s'$. A variable assignment α is a solution of an equation $s{=}s'$ if $\alpha(s) = \alpha(s')$, and of a subtree constraint $s{\ll}s'$ if $\alpha(s)$ is a subtree of $\alpha(s')$.

Proposition 1. *The following equivalences hold in the structure of finite trees:*

$$s_1{=}s_2 \;\leftrightarrow\; s_1/s_1{=}s_2/s_1 \qquad \textit{and} \qquad s_1{\ll}s_2 \;\leftrightarrow\; s_2/s_1{=}s_2/s_1$$

Proof. The implication from the right to the left in the first equivalence can be proved as follows. If $\sigma_1/\sigma_1{=}\sigma_2/\sigma_1$ then there exists γ such that $\sigma_1 = \gamma(\sigma_1)$ and $\sigma_2 = \gamma(\sigma_1)$. This yields $\sigma_1 = \sigma_2$. The second equivalence is trivial.

Theorem 2 (Venkataraman 1987)**.** *The existential fragment of the first-order theory of subtree constraints $s{\ll}s'$ over finite trees is decidable and NP-complete. The $\exists^*\forall^*$ fragment of this theory is undecidable.*

A proof has been given by Venkataraman in [28]. Note that Theorem 2 carries over to the subtree relation on infinite trees [27, 25]. In this case, explicit equality constraints $s{=}s'$ have to be provided, since the equivalence $X{\ll}X' \wedge X'{\ll}X \leftrightarrow X{=}X'$ does not hold over infinite trees in contrast to finite trees.

Definition 3. An equality up-to constraint is *uniform* if all its conjuncts are of the form $s_1/s{=}s_2/s$.

Lemma 4. *An equality up-to constraint is equivalent to a uniform equality up-to constraint if and only if it is equivalent to a conjunction of subtree and equality constraints. The corresponding constraints can be computed in linear time.*

Proof. This follows from the equivalence $s_1/s{=}s_2/s \leftrightarrow s_1{=}s_2 \wedge s{\ll}s_1$, which holds over finite trees (and also over infinite trees).

Theorem 5. *The existential fragment of the first-order theory of uniform equality up-to constraints is decidable and NP-complete. The $\exists^*\forall^*$ fragment of this theory is undecidable.*

Proof. Every uniform equality up-to constraint is equivalent to a conjunction of subtree and equality constraints by Lemma 4 and conversely by Proposition 1. Thus, the theorem is a corollary to Theorem 2.

The decidability of the positive existential fragment of the first-order theory of uniform equality up-to constraints can also be reduced to Comon's result [4] about uniform context constraints reformulated in Theorem 11. Via Lemma 4 this yields an alternative proof to Venkataraman's decidability result in Theorem 2 but without negation.

3.3 Two Occurrences Restriction and Stratification

We distinguish two more fragments of equality up-to constraints that have a decidable satisfiability problem. Both fragments are obtained by translating known analogous results for context unification. In particular, we note that our notion of stratification for equality up-to constraints given here is motivated by an analogous notion for context constraints introduced in Section 4.3.

Theorem 6. *The satisfiability of equality up-to constraints with at most two occurrences per first-order variable is decidable.*

Proof. This theorem reduces via Proposition 15 to the analogous result for context unification formulated in Theorem 12 (which has first been proved by Lévy [12] in the setting of linear second-order unification).

Definition 7. An equality up-to constraint φ is *stratified* if whenever a variable X occurs in s'_1 or s'_2 in a conjunct $s_1/s'_1{=}s_2/s'_2$ of φ then X does not occur in s_1 and s_2 and not in any other conjunct of φ.

Theorem 8. *The satisfiability of stratified equality up-to constraints is decidable.*

Proof. This theorem reduces via Proposition 15 to an analogous result for context unification given in Theorem 14 (which has first been proved by Schmidt-Schauß in [22]).

4 Context Unification

We define the syntax and semantics of context constraints and the notion of context unification. We also distinguish some decidable and undecidable fragments of the first-order theory of context constraints.

4.1 Syntax and Semantics

We assume an additional infinite number of second-order variables ranged over by C. A *second-order term* t is either a first-order variable X, a construction $a(t_1, \ldots, t_n)$ where the arity of a is n, or a term of the form $C(t)$, where $t, t_1, \ldots, t_n$ are second-order terms. In particular, every first-order term s is also a second-order term. A *context constraint* is a conjunction of equations $t{=}t'$ between second-order terms[3].
Semantically, we interpret context variables as context functions and second-order terms as finite trees (like first-order terms). Let α be a variable assignment that maps first-order variables to finite trees and second-order variables to context functions. The interpretation $\alpha(t)$ of a second-order term t under α is defined homomorphically.

$$\alpha(a(t_1, \ldots, t_n)) = a(\alpha(t_1), \ldots, \alpha(t_n))$$
$$\alpha(C(t)) = \alpha(C)\,(\alpha(t))$$

A *solution* of a context constraint ψ is a variable assignment α that satisfies all equations in ψ. A context constraint is called *satisfiable* if it has a solution. *Context unification* is the satisfiability problem of context constraints.

4.2 Subtree and Equality Constraints

As shown in Section 3.2, subtree and equality constraints can be expressed with uniform equality up-to constraints and vice versa. Here, we define uniform context constraints in analogy. This notion has been investigated before by Comon [4] but without stating the correspondence to subtree and equality constraints (see Lemma 10).

Definition 9. We call a context constraint ψ *uniform* if whenever $C(t_1)$ and $C(t_2)$ occur in ψ then t_1 is equal to t_2.

Lemma 10. *Every uniform context constraint is satisfaction equivalent to a conjunction of subtree and equality constraints. The corresponding constraint can be computed in cubic time.*

Proof. As we will show in Propositions 15 and 17, every uniform context constraint is satisfaction equivalent to a uniform equality up-to constraint and vice versa. Thus, the result follows from Lemma 4.

[3] In higher-order unification, a context constraint would be called a context unification problem.

Theorem 11 (Comon 1992). *The positive existential fragment of the first-order theory of uniform context constraints is decidable. The $\exists^*\forall^*\exists^*$ fragment of this theory is undecidable.*

The decidability result in Theorem 11 has also first been proved by Comon in [4]. A simpler proof has been presented in [12] and can also be found in Appendix C. The negative result of Theorem 11 is original to the present paper.

Proof. The full theorem follows from Lemma 10 and Theorem 5 (which is a consequence of the Venkataraman's result).

Note that the correspondence in Lemma 10 is formulated with respect to satisfaction equivalence. We therefore needed an additional layer of existential quantifiers in the undecidability result of Theorem 11, i.e. we obtain a weaker undecidability result for context constraints than for equality up-to constraints.

4.3 Two Occurrences Restriction and Stratification

We recall two more decidability results for fragments of context unification that have been proved by Lévy [12] and Schmidt-Schauß [22].

Theorem 12 (Lévy 1996). *Context unification restricted to context constraints with at most two occurrences per variable (first and second-order) is decidable.*

Proof. This result has first been proved by Lévy [12] in the framework of linear second-order unification. A proof adapted to the setting of context unification is sketched in Appendix C.

Definition 13. Let ξ be either a first-order or a second-order variable. A *second-order prefix* of ξ in a term t is a word of second-order variables that is obtained when traversing t from the root to an occurrence of ξ in t. We write $\mathcal{P}(\xi, t)$ for the set of all second-order prefixes of ξ in t. The set $\mathcal{P}(\xi, \psi)$ of all second-order prefixes of ξ in a constraint ψ is defined homomorphically:

$$\mathcal{P}(\xi, \psi \wedge \psi') = \mathcal{P}(\xi, \psi) \cup \mathcal{P}(\xi, \psi'), \qquad \mathcal{P}(\xi, t{=}t') = \mathcal{P}(\xi, t) \cup \mathcal{P}(\xi, t')$$

A context constraint ψ is called *stratified* if the set $\mathcal{P}(\xi, \psi)$ contains at most one element for every first-order and second-order variable ξ in ψ.

Theorem 14 (Schmidt-Schauß 1994). *Context unification restricted to stratified context constraints is decidable.*

Proof. This theorem has been first proved by Schmidt-Schauß [22]. A simpler proof has been proposed by Lévy [12]. However, at the time of submitting this paper, the termination proof given there has not been agreed on to be complete.

5 Correspondence

The relationship of context unification and equality up-to constraints over finite trees is formalized in this section. The only non-obvious fact we need is stated in Lemma 16.

Proposition 15. *There is a linear time transformation of equality up-to constraints into satisfaction equivalent context constraints which preserves uniformity, stratification, and the number of occurrences of first-order variables. Second-order variables may be introduced, but the introduced second-order variables occur at most twice.*

Proof. This is a consequence of the following equivalence that can be used as a transformation rule when oriented from the left to the right:

$$s_1/s_1' = s_2/s_2' \quad \leftrightarrow \quad \exists C\,(s_1 = C(s_1') \wedge s_2 = C(s_2')) \qquad \square$$

For reducing context unification to the problem of solving equality up-to constraints, we introduce generalized n-ary equality up-to constraints of the form $s_1/s_1' = \ldots = s_n/s_n'$ for any n. These new constraints are interpreted with respect to the n-ary equality up-to relation $\sigma_1/\sigma_1' = \ldots = \sigma_n/\sigma_n'$, which holds if there exists a context function γ such that $\sigma_i = \gamma(\sigma_i')$ for $i = 1 \ldots n$.

Lemma 16 (Coherence)**.** *The following equivalence holds in the structure of finite trees for all $n \geq 1$:*

$$s_1/s_1' = \ldots = s_n/s_n' \qquad \leftrightarrow \qquad \bigwedge_{i,j \in \{1,\ldots,n\}} s_i/s_i' = s_j/s_j'$$

A proof of this Lemma is given in Appendix A. In order to illustrate that this Lemma is non-trivial, we give the following example:

$$\left.\begin{array}{r} f(a,b)/a = f(b,b)/b \\ \wedge\ f(b,b)/b = f(b,a)/a \end{array}\right\} \not\rightarrow\ f(a,b)/a = f(b,b)/b = f(b,a)/a$$

In this case, the coherence lemma is not applicable because the required assumption

$$f(a,b)/a = f(b,a)/a$$

does not hold. This shows that the coherence lemma needs all pairwise equality up-to constraints. Our proof has to take care of all of them simultaneously.

Proposition 17. *There exists a cubic time transformation of context constraints into satisfaction equivalent equality up-to constraints. This transformation preserves uniformity but neither stratification nor the number of variable occurrences.*

Proof. We apply the following five transformation steps consecutively.

Step 1 We replace equations $t{=}t'$ by conjunctions $X{=}t \wedge X'{=}t'$ where X, X' are fresh, unless t is a variable. Thereafter, we replace equations of the form $X{=}a(t_1, \ldots, t_n)$ by conjunctions $X{=}a(X_1, \ldots, X_n) \wedge X_1{=}t_1 \wedge \ldots \wedge X_n{=}t_n$ and equations $X{=}C(t')$ with $X{=}C(X') \wedge X'{=}t'$ where all X's are fresh.

Step 2 We regroup the conjuncts into equations $s_1{=}s_2$ and conjunctions $s_1{=}C(s'_1) \wedge \ldots \wedge s_m{=}C(s'_m)$ such that C occurs nowhere outside this conjunction in the constraint.

Step 3 We replace a conjunction $s_1{=}C(s'_1) \wedge \ldots \wedge s_n{=}C(s'_n)$ by the formula $\exists C\,(s_1{=}C(s'_1) \wedge \ldots \wedge s_n{=}C(s'_n))$. This is a satisfaction equivalent transformation since we assume that C does not occur elsewhere.

Step 4 We apply the following equivalences as transformations from the left to the right:

$$s_1{=}s_2 \quad \leftrightarrow \quad s_1/s_1{=}s_2/s_1$$

$$\exists C\,(s_1{=}C(s'_1) \wedge \ldots \wedge s_n{=}C(s'_n)) \quad \leftrightarrow \quad s_1/s'_1{=}\ldots{=}s_n/s'_n$$

Step 5 We apply the equivalence of Lemma 16 from the left to the right.

All transformations above can be performed in linear time except step 5. This step is quadratic and does neither preserve stratification nor the number of variable occurrences. All steps preserve uniformity except step 1. But we can slightly modify step 1 in order to preserve uniformity. It is sufficient to always replace equal subterms by the same variable. This needs a quadratic number of equality tests that can be done in cubic time. □

Corollary 18. *The satisfiability problem of equality up-to constraints is equivalent to context unification.*

Proof. From Propositions 15 and 17. □

6 One-Step Rewriting

Context unification is closely related to the first-order theory of one-step rewriting. The theory of one-step rewriting is a set of theories rather than a single theory. Each of these theories is a subset of the same first-order language, which contains all so called *one-step rewriting formulae* that can be built from expressions $X{\to}X'$ and the usual first-order connectives. Let Σ be a signature of function symbols. A *rewrite system* R (over Σ) is a finite set of rules $s{\to}s'$, which are pairs of terms s and s' over the signature Σ. For every rewrite system R, the structure $\mathcal{A}_R$ is an extension of the structure of finite trees with an additional binary relation. A formula $X{\to}X'$ is interpreted in $\mathcal{A}_R$ with respect to the binary relation $\sigma{\to}_R\sigma'$ on trees σ and σ', which holds iff σ rewrites in one step to σ' by using a rule in R.

The *theory of one-step rewriting with respect to* R is the set of valid formulae interpreted over the structure $\mathcal{A}_R$. Treinen has shown in [26] that it is undecidable if a one-step rewriting formula belongs to the $\exists^*\forall^*$ fragment of the theory

of one-step rewriting with respect to R. As proved recently, Treinen's result still holds if the considered rewrite systems are restricted to be linear, right ground rewrite [14] or linear, Noetherian [29].
For our purpose, we prefer to use one-step rewriting constraints rather than one-step rewriting formula. A *one-step rewriting constraint* is a conjunction of expressions $X{\to}X'$ **with** $s{\to}s'$, which holds in the structure of finite trees (and its extensions $\mathcal{A}_R$) if the tree denotation of X rewrites in a one step to the tree denotation of X' by using the rewrite rule $s{\to}s'$.

Lemma 19. *In the structure $\mathcal{A}_R$, every one-step rewriting formula of the form $\bigwedge_{i=1}^{n}(X_i{\to}X_i')$ is equivalent to a disjunction of one-step rewriting constraints.*

Proof. Let $R = \{s_j{\to}s_j' \mid 1 \leq j \leq m\}$. The following equivalence holds over $\mathcal{A}_R$:

$$\bigwedge_{i=1}^{n}(X_i{\to}X_i') \;\leftrightarrow\; \bigwedge_{i=1}^{n}\bigvee_{j=1}^{m}(X{\to}X' \textbf{ with } s_j{\to}s_j')$$

It is sufficient to compute the disjunctive normal form of the right hand side.

Lemma 20. *Every one-step rewriting constraint is satisfaction equivalent to a disjunction of stratified equality up-to constraints.*

Proof. Consider the one-step rewriting constraint $\bigwedge_{i=1}^{n}(X_i{\to}X_i' \textbf{ with } s_i{\to}s_i')$. Let $\{Y_i^1, \ldots, Y_i^{m(i)}\}$ be the set of variables occuring in s_i and s_i'. We can assume $\cup_{i=1}^{n}\{X_i, X_i'\} \;\cap\; \cup_{j=1}^{n}\{Y_j^1, \ldots, Y_j^{m(j)}\} = \emptyset$ by renaming the variable in $s_i{\to}s_i'$ appropriately. The following equivalence holds for all $1 \leq i \leq n$:

$$X_i{\to}X_i' \textbf{ with } s_i{\to}s_i' \;\leftrightarrow\; \exists Y_i^1 \ldots \exists Y_i^{m(i)}(X/s_i{=}X'/s_i')$$

Thus $\bigwedge_{i=1}^{n}(X_i{\to}X_i' \textbf{ with } s_i{\to}s_i')$ and $\bigwedge_{i=1}^{n}(X/s_i{=}X'/s_i')$ are satisfaction equivalent. The latter equality up-to constraint is stratified because of the above variable disjointness condition.

Theorem 21. *The positive existential fragment of the first-order theory of one-step rewriting is decidable.*

Proof. This follows from Lemma 19, Lemma 20, and Theorem 8.

7 Examples for Solving Context Constraints

We present two simple examples that show how to solve context constraints according to an algorithm given in Appendix B available in the full version of the paper. We give two examples, one example for ellipses in natural languages and one for one-step rewriting.

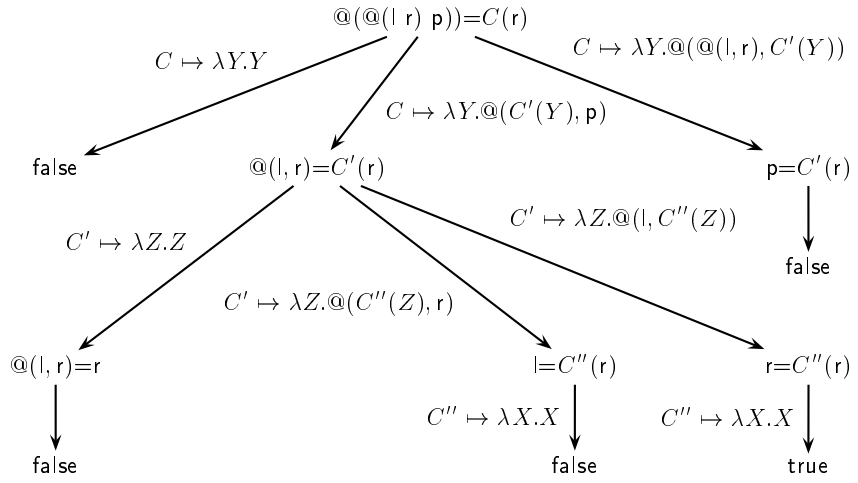
For the analysis of the elliptic sentence Peter likes Riesling, Chardonnay too in Section 2 we have derived the following equality up-to constraint:

$$X_1{=}@(@(\mathsf{likes\ Riesling})\ \mathsf{Peter}))\ \wedge\ X_1/\mathsf{Riesling}{=}X_2/\mathsf{Chardonnay}$$

This constraint is equivalent to the constraint φ_0 given by the following equations:

$$\begin{array}{lcl}\varphi_0 & = & \exists C(\varphi_1 \wedge X_1{=}C(\mathsf{Riesling}) \wedge X_2{=}C(\mathsf{Chardonnay}))\\ \varphi_1 & = & @(@(\mathsf{likes\ Riesling})\ \mathsf{Peter}){=}C(\mathsf{Riesling})\end{array}$$

In the computation below, it is shown that there is unique solution for φ_1 that maps C to the context function γ with $\gamma(\sigma) = @(@(\mathsf{likes}\ \sigma)\ \mathsf{Peter}))$. This context function represents the semantics of too in the given elliptic sentence.



We now give an example for solving a one-step rewriting constraint. We consider a signature of two unary function constant a and b and the following constraint:

$$X{\to}Y\ \textbf{with}\ a(Z){\to}b(Z)\ \wedge\ Y{\to}X\ \textbf{with}\ b(U){\to}U$$

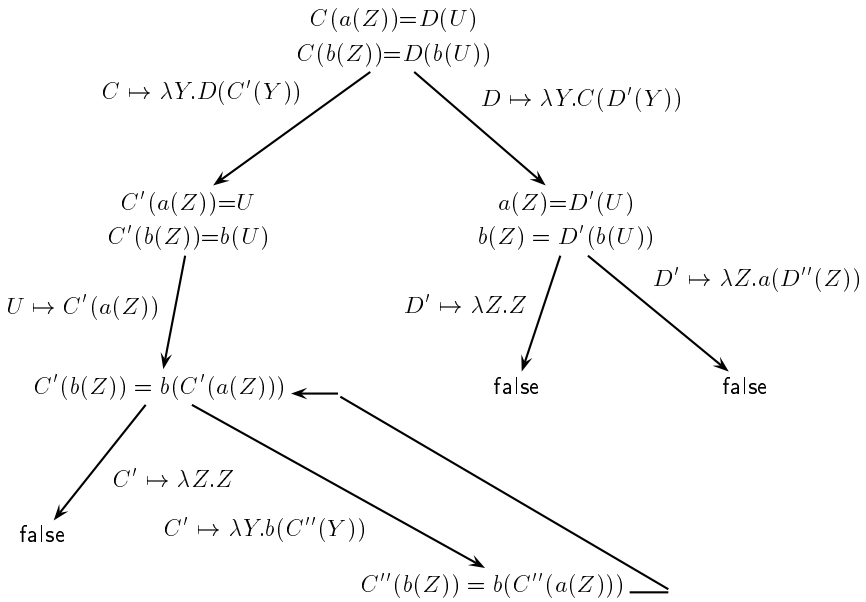
This constraint is equivalent to the context unification constraint φ_2 with

$$\varphi_2 \quad = \quad X{=}C(a(Z))\ \wedge\ Y{=}C(b(Z))\ \wedge\ Y{=}D(b(U))\ \wedge\ X{=}D(U)$$

Below, we solve the following constraint φ_3 that is logically implied by of φ_2:

$$\varphi_3 \quad = \quad C(a(Z)){=}D(U) \wedge C(b(Z)){=}D(b(U))$$

In the single non failed alternative, we obtain a cycle (up to renaming C'' to C') without any exit to a solution. This shows that φ_3 and thereby φ_2 are unsatisfiable.



Acknowledgment The authors are grateful to Ralf Treinen for pointing out the existence of a relationship between the theory of one-step rewriting and equality up-to constraints. Ralf also contributed a pointer to the results of Venkataraman on the first-order theory of the subtree relation. We would like to thank Jörg Siekmann for telling us about the history of string unification and providing us with the related references. We are grateful to Jordi Lévy for many discussions on linear second-order unification and string unification. Martin Müller contributed to the coherence property and gave many other useful comments.

The research reported in this paper has been supported by the SFB 378 at the Universität des Saarlandes and the Esprit Working Group CCL II (EP 22457).

References

1. F. Baader and K. Schulz. Unification in the union of disjoint equational theories. *Int. Conference on Automated Deduction*, volume 607 of *LNCS*, pages 50–65, 1992.
2. F. Baader and J. Siekmann. *Handbook of Logic in Artificial Intelligence and Logic Programming*, chapter Unification Theory. Oxford University Press, 1993.
3. A. Caron, J.-L. Coquide, and M. Dauchet. Encompassment properties and automata with constraints. *Int. Conference on Rewriting Techniques and Applications*, volume 690 of *LNCS*, pages 328–342. 1993.
4. H. Comon. Completion of rewrite systems with membership constraints. *Int. Coll. on Automata, Languages and Programming, LNCS 623*, Vienna, 1992.
5. M. Dalrymple, S. Shieber, and F. Pereira. Ellipsis and higher order unification. *Linhuistics and Philosophy*, 14:399–452, 1991.

6. C. Gardent, and M. Kohlhaase. Higher-order coloured unification and natural language semantics. *Anual Meeting of the ACL*, 1996.
7. W. D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.
8. G. Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.
9. J. Jaffar. Minimal and complete word unification. *Journal of the ACM*, 37(1):47–85, 1990.
10. A. Kościelski and L. Pacholski. Complexity of makanin's algorithm. *Journal of the ACM*, 43(4), July 1996.
11. A. Lentin and M. Schützenberger. A combinatorial problem in the theory of free monoids. *Conference on Combinatorical Mathematics and its Applications.*, 1969.
12. J. Lévy. Linear second order unification. *Int. Conference on Rewriting Techniques and Applications*, number 1103 of *LNCS*, 1996.
13. G. Makanin. The problem of solvability of equations in a free semigroup. *Soviet Akad. Nauk SSSR*, 223(2), 1977.
14. J. Marcinkowski. Undecidability of the first order theory of one-step right ground rewriting. *Int. Conference on Rewriting Techniques and Applications*. LNCS, 1997.
15. A. Markov. The theory of algorithms. *Trudy Mat. Inst. Steklov*, 1(42), 1954. English Translation in Israel Program for Scientific Translations, Jerusalem, 1968.
16. J. Niehren, M. Pinkal, and P. Ruhrberg. On equality up-to constraints over finite trees, context unification, and one-step rewriting, 1997. Full version available through WWW from http://www.ps.uni-sb.de/ niehren.
17. J. Niehren, M. Pinkal, and P. Ruhrberg. A uniform approach to underspecification and parallelism, 1997. Submitted. Available through WWW from http://www.ps.uni-sb.de/ niehren.
18. J. Pécuchet. Solution principale et rang d'un system d'èquations avec constantes dans le monoïde libre. *Discrete Mathematics*, 48:253–274, 1984.
19. T. Pietrzykowski. A complete mechanization of second-order logic. *Journal of the ACM*, 20(2):333–364, 1973.
20. M. Pinkal. Radical underspecification. CLAUS Report 72, Universität des Saarlandes, Saarbrücken, 1996.
21. G. D. Plotkin. Building in equational theories. *Machine Intelligence*, (7):73–90, 1972.
22. M. Schmidt-Schauß. Unification of stratified second-order terms. Internal report 12/94, J. W. Goethe Universität, Frankfurt, Germany, 1994.
23. K. U. Schulz. Word unification and transformation of generalized equations. *J. of Automated Reasoning*, 11:149–184, 1993.
24. J. H. Siekmann. String-unification: Part 1. Technical Report Internal Report CSM7, Essex University, 1975.
25. R. Treinen. A new method for undecidability proofs of first order theories. *journal of symbolic computation*, 14:437–457, 1992.
26. R. Treinen. The first-order theory of one-step rewriting is undecidable. *Int. Conf. on Rewriting Techn. and Appl.*, number 1103 in LNCS, pages 276–286, 1996.
27. S. Tulipani. Decidability of the existential theory of infinite terms with subterm relation. *Journal on Information and Computation*, 103(2), 1993.
28. K. N. Venkataraman. Decidability of the purely existential fragment of the theory of term algebra. *Journal of the ACM*, 34(2):492–510, Apr. 1987.
29. S. Vorobyov. The first-order theory of one step rewriting in linear noetherian systems is undecidable. *Int. Conf. on Rewriting Techn. and Appl. LNCS*, 1997.

A Proof of the Coherence Lemma

We present the proof of the Coherence Lemma, which is omitted in the conference version of the present paper.

Lemma 16 [Coherence] *The following equivalence holds in the structure of finite trees for all $n \geq 1$:*

$$s_1/s'_1 = \ldots = s_n/s'_n \qquad \leftrightarrow \qquad \bigwedge_{i,j \in \{1,\ldots,n\}} s_i/s'_i = s_j/s'_j$$

Proof. The cases $n = 1$ and $n = 2$ are trivial. The implication from the left to the right is also trivial. We show the implication from the right to the left for $n \geq 3$. We assume trees σ_i and context functions γ_{ij} such that the pairwise equations

$$\sigma_i = \gamma_{ij}(\sigma'_i) \qquad \text{and} \qquad \sigma_j = \gamma_{ij}(\sigma'_j)$$

hold for $i, j \in \{1, \ldots, n\}$ and $i < j$. If there exists $i \neq j$ such that $\sigma'_i = \sigma'_j$ then $\sigma_i = \sigma_j$. By induction on n there exists a context function γ with $\sigma_k = \gamma(\sigma'_k)$ for all $k \neq i$. Thus, $\sigma_i = \sigma_j = \gamma(\sigma'_j) = \gamma(\sigma'_i)$. It remains the case where all σ'_i are pairwise distinct. In this case, we have $\gamma_{ij} = \gamma_{i'j'}$ for all $i < j$ and $i' < j'$ as proved in Lemma 22.

Lemma 22. *Let $n \geq 3$, σ_i, σ'_i be trees and γ_{ij} context functions where $i, j \in \{1, \ldots, n\}$. If all σ'_i are pairwise distinct and $\sigma_i = \gamma_{ij}(\sigma'_i)$ and $\sigma_j = \gamma_{ij}(\sigma'_j)$ for all $i < j$ then $\gamma_{ij} = \gamma_{i'j'}$ for all $i < j$ and $i' < j'$.*

Proof. The case $n = 3$ is solved in Lemma 23 and the case $n > 3$ trivially reduces to the case $n = 3$. □

Lemma 23. *Let $\sigma_1, \sigma_2, \sigma_3, \sigma'_1, \sigma'_2, \sigma'_3$ be trees and γ_{12}, γ_{23}, and γ_{13} context functions. If $\sigma'_1, \sigma'_2, \sigma'_3$ are pairwise distinct and $\sigma_i = \gamma_{ij}(\sigma'_i)$ and $\sigma_j = \gamma_{ij}(\sigma'_j)$ for all $i < j$ then $\gamma_{12} = \gamma_{23} = \gamma_{13}$.*

Proof. A *path* p is a sequence of integers. A path p_1 is called a *prefix* of p_2 if there exists a path p'_2 such that $p_2 = p_1 p'_2$. A *proper prefix* of p is a prefix of p that is distinct from p. Two path p_1 and p_2 *diverge* if p_1 is a not a prefix of p_2 and p_2 is not a prefix of p_1. Given a tree σ we write $\sigma|p$ for the subtree of σ at path p and assume that such a subtree exists.

Let p_{ij} be the path that leads to the unique variable occurrence in the term defining the context function γ_{ij}. The paths p_{12}, p_{13}, and p_{23} cannot be proper prefixes of each other. Otherwise, for the case p_{12} and p_{13} for example, $\sigma_1|p_{12} = \sigma'_1$ and $\sigma_1|p_{13} = \sigma'_1$, so that σ'_1 would be a proper subtree of itself, which is not possible over finite trees. The other cases are analogous.

We prove $\gamma_{12} = \gamma_{13}$. The proof of the remaining equalities is symmetric. If $p_{12} = p_{13}$ then $\gamma_{12} = \gamma_{13}$ since $\gamma_{12}(\sigma'_1) = \sigma_1 = \gamma_{13}(\sigma'_1)$. Otherwise (i.e. $p_{12} \neq p_{13}$), the

path p_{12} and p_{13} must diverge (they cannot be proper prefixes of each other as shown above and they cannot be equal by assumption). If $p_{12} = p_{23}$ then we get

$$\sigma'_1 = \sigma_1|p_{12} = \sigma_3|p_{12} = \sigma_3|p_{23} = \sigma'_3\,,$$

which contradicts $\sigma'_1 \neq \sigma'_3$. Thus p_{12} and p_{23} must also be disjoint. This implies

$$\sigma'_1 = \sigma_1|p_{12} = \sigma_3|p_{12} = \sigma_2|p_{12} = \sigma'_2\,,$$

which contradicts $\sigma'_1 \neq \sigma'_2$ such that $p_{12} \neq p_{13}$ is not possible in any case. □

B An Algorithm for Context Unification

In this section, we present an algorithm that adapts Plotkin's string unification algorithm [21] for context unification. This algorithm is correct and complete but does not terminate in the all cases we are interested in. However, it justifies the examples in Section 7.

The algorithm operates on sets Γ of symmetric equations $t{=}t'$. In other words, we identify context constraints up to the following equality relation:

$$t{=}t' \equiv t'{=}t \qquad \varphi \wedge \psi \equiv \psi \wedge \varphi$$
$$(\varphi \wedge \psi) \wedge \varphi \equiv \varphi \wedge (\psi \wedge \varphi) \qquad \varphi \wedge \varphi \equiv \varphi$$

Our algorithm operates as a state transformer. A *state* is a pair $\langle \Gamma, \rho \rangle$ where Γ is the set of equations and ρ is a substitution. The intuition is that Γ contains those constraints that have still to be solved and that ρ represents a partial solution of the initial context constraint. A *unifier* for a set Γ is a substitution ρ such that $\rho(t)$ is syntactically identical to $\rho(t')$ for each equation $t{=}t'$ in Γ.

For a given context constraint Γ the starting state is $\langle \Gamma, Id \rangle$. The constraint is solved if a final state of the form $\langle \emptyset, \rho \rangle$ can be reached by (indeterministically) applying transformation rules. A *transformation rule* is of the form

$$t{=}t' \;\longrightarrow\; \Gamma \;\mid\; \rho$$

which applied to the state $\langle \{t{=}t'\} \cup \Gamma', \rho' \rangle$ yields the new state $\langle \rho(\Gamma \cup \Gamma'), \rho \circ \rho' \rangle$.

Here, we use a notation for describing context functions by means of (linear second order) λ-terms. These are of the form $\lambda X.t$ where t is a second-order term such that the (first order) variable X occurs exactly once in t. If such a λ-term contains no free variables it uniquely specifies a context function. We implicitly assume that in performing a substitution $\rho(\Gamma)$ we also normalize the terms, i.e. we β-convert any subterms of the form $(\lambda X.t)(t')$ that can occur when a context variable is replaced by a λ-term. We note that a context constraint Γ has a solution if and only if Γ has a unifier.

The state transformation rules of our algorithm are given in Table 1. We call the path of a context that leads to it's hole the context's *exception path*. Notice that the rule Flex-Flex1 assumes that the exception path of the context C is a prefix of the exception path of C', so that t' must be a subtree of t. For context

(Subst)	$X{=}t \longrightarrow \mathsf{true}$ if $X \notin V(t)$	$\mid X \mapsto t$
(Decomp)	$a(t_1,...,t_n){=}a(t'_1,...,t'_n) \longrightarrow \bigwedge_{i=1..n} t_i{=}t'_i$	$\mid Id$
(Proj)	$a(t_1,...,t_n){=}C(t') \longrightarrow a(t_1,...,t_n){=}t'$	$\mid C \mapsto \lambda X.X$
(Imit)	$a(t_1,...,t_n){=}C(t') \longrightarrow t_i{=}C'(t')$	$\mid C \mapsto \lambda X.a(t_1,..,t_{i-1},C'(X),t_{i+1},..,t_n)$
(Simpl)	$C(t){=}C(t') \longrightarrow t{=}t'$	$\mid Id$
(Flex-Flex1)	$C(t){=}C'(t') \longrightarrow t{=}C''(t')$	$\mid C' \mapsto \lambda X.C(C''(X))$
(Flex-Flex2)	$C(t){=}C'(t') \longrightarrow \mathsf{true}$	$\mid C \mapsto \lambda Y.C_1(a(\,\pi(\overline{X}, C_2(Y), C_3(t'))\,)),$ $C' \mapsto \lambda Z.C_1(a(\,\pi(\overline{X}, C_2(t), C_3(Z))\,))$ where π is a permutation

Table 1. A Correct and Complete Algorithm for Context Unification

unification with constants of arity ≥ 2 this rule does not suffice, as two subtrees of some tree can live on diverging branches. The rule Flex-Flex2 covers these cases by containing for each constant a of arity n and each permutation π of n-ary sequences an instance with $n-2$ fresh variables $\overline{X}$.
The algorithm is sound and complete (which we will not prove here) but has certain disadvantages. Besides introducing a potentially very large search space, the Flex- Flex2 rule increases the size of a constraint even within the fragment of context unification where every variable occurs at most twice.
The *size* of a context constraint (seen as a set) Γ is defined as follows (where ξ ranges over constants and variables of appropriate arity):

$$size(\xi(t_1,\ldots,t_n)) = n + \textstyle\sum_{i=1}^{n} size(t_i)$$
$$size(\{t_1{=}t'_1,\ldots,t_n{=}t'_n\}) = \textstyle\sum_{i=1}^{n} size(t_i) + size(t'_i)$$

Under this definition of the size of a context constraint, the above algorithm would not yield a termination result for the 2-occurrence fragment. We therefore present a different algorithm in Appendix C, that uses n-ary context variables.

C Lévy's Algorithm

We modify our algorithm of the previous section such that it terminates for several decidable fragments of context unification including the restrictions to uniform context constraints (Theorem 11) and to the two occurence case (Theorem 12). The algorithm we present here can also be seen as a reformulation of Lévy's linear second-order unification algorithm for context unification. We omit proofs of correctness, completeness, and termination, since these can be found

in [12]. There, it is also claimed that Lévy's algorithm terminates for stratified context constraints (Theorem 14). But up to the time point of writing this paper, it has not become clear wether the proof sketch given in [12] can be completed.

We now consider context variables C and D with arities $n \geq 1$ but possibly different from 1. An (extended) second-order term t additionally admits applications of the form $C(t_1, \ldots, t_n)$ where C is a context variable of arity n. An (extended) *context constraint* is a conjunction of equations $t{=}t'$ between extended second-order terms.

A tree σ is as before a ground first-order term. An *(n-ary) context function* γ is a function from sequences of trees to trees that is described by an equation of the form

$$\gamma(\sigma_1, ..., \sigma_n) = s[\sigma_i/X_i]_{i=1}^n \qquad \text{for all trees } \sigma_1, ..., \sigma_n$$

where s is a first-order term that contains a single occurrence of each variable $X_1, ..., X_n$ and no occurrences of any other variables. A *solution* to a context constraint is defined as before as an assignment α that satisfies all equations of the constraint. We now use linear λ-terms of the more general form $\lambda\overline{X}.t$ in the substitutions of the algorithm, which as before are normalized when the substitution is carried out.

Notice that the different exception paths of an n-ary context cannot be prefixes of one-another. Thus the Flex-Flex2 case of the first algorithm, with monadic context variables whose exception paths stand in no prefix relation, can be neatly rephrased so that the size of the problem can be better controlled, by writing

$$C(t){=}C'(t') \;\longrightarrow\; \mathsf{true} \;\mid\; C \mapsto \lambda Y.C''(t', Y)\,,\; C' \mapsto \lambda Z.C''(Z, t)$$

In Table 2, our simple algorithm is reformulated for the more general problem of n-ary context unification, following closely Lévy's [12] algorithm for linear second order unification. In particular, the Flex-Flex rule has to account for all possible prefix relations between the holes of two contexts of arbitrary arity.

To state the Flex-Flex rule correctly, we need a few auxiliary definitions. Let P and Q run over sets of indices which are linearly ordered, so that we may write them as $P = \{p_1, \ldots, p_n\}$ and $Q = \{q_1, \ldots, q_m\}$ respectively. These ordered index sets correspond to the exception paths of two contexts. The expression $\overline{t}_P$ stands for the sequence $t_{p_1}, \ldots, t_{p_n}$. For any $P' \subseteq P$, the expression $\overline{t}_{P'}$ stands for the sequence $t_{p_i}, \ldots, t_{p_k}$, where t_{p_j} occurs in the sequence iff $p_j \in P'$, and t_{p_j} occurs before t_{p_l} iff $j < l$. If the form of the members of such a sequence depends on the index p via some function ζ we also use the notation $\overline{\zeta(p)}_{p \in P'}$, presuming again that the order of P is preserved.

For two sets of indices P, Q we need to define admissible functions θ that encode a possible prefix relation between the two sets of exception paths of two contexts. We say that a function $\theta : P \cup Q \to \wp(P) \cup \wp(Q)$ is *admissible* if

1. $\theta(p) \subseteq Q$ for all $p \in P$,
2. $\theta(q) \subseteq P$ for all $q \in Q$,
3. $\theta(r) \cap \theta(u) = \emptyset$ for $r \neq u$,

4. $u \in \theta(r) \Rightarrow \theta(u) = \emptyset$.

We need the following subsets of indices:

$$P' =_{df} \{p \in P \mid \theta(p) \neq \emptyset\} \qquad P'' =_{df} \{p \in P \mid \theta(p) = \emptyset \wedge \neg\exists q(p \in \theta(q))\}$$
$$Q' =_{df} \{q \in Q \mid \theta(q) \neq \emptyset\} \qquad Q'' =_{df} \{q \in Q \mid \theta(q) = \emptyset \wedge \neg\exists p(q \in \theta(p))\}$$

The imitation rule is also more complex than before. We need to guess a $\vartheta : P \to \wp(Q)$ such that

1. $\vartheta(r) \cap \vartheta(u) = \emptyset$ for $r \neq u$,
2. $\bigcup_{p \in P} \vartheta(p) = Q$.

We define $P' =_{df} \{p \in P \mid \vartheta(p) \neq \emptyset\}$ for the Imitation rule, and let $\zeta(p) =_{df} C'_p(\overline{X}_{\vartheta(p)})$ if $p \in P'$, otherwise $\zeta(p) =_{df} t_p$. All variables introduced in the rules are assumed to be fresh.

(Subst)	$X{=}t \longrightarrow \mathsf{true}$ if $X \notin V(t)$	$\mid X \mapsto t$
(Decomp)	$a(\overline{t}_P) = a(\overline{t'}_P) \longrightarrow \bigwedge_{p \in P} t_p = t'_p$	$\mid Id$
(Proj)	$a(\overline{t}_P) = C(t') \longrightarrow a(\overline{t}_P) = t'$	$\mid C \mapsto \lambda X.X$
(Imit)	$a(\overline{t}_P) = C(\overline{t'}_Q) \longrightarrow \bigwedge_{p \in P'} t_p = C'_p(\overline{t'}_{\vartheta(p)})$	$\mid C \mapsto \lambda \overline{X}_Q.a(\overline{\zeta(p)}_{p \in P})$
(Simpl)	$C(\overline{t}_P) = C(\overline{t'}_P) \longrightarrow \bigwedge_{p \in P} t_p = t'_p$	$\mid Id$
(FlexFlex)	$C(\overline{t}_P) = C'(\overline{t'}_Q) \longrightarrow \bigwedge_{p \in P'} t_p = D'_p(\overline{t'}_{\theta(p)}) \wedge \bigwedge_{q \in Q'} t'_q = D_q(\overline{t}_{\theta(q)})$	$\mid C \mapsto \lambda \overline{Y}_P.C''(\overline{D_q(\overline{Y}_{\theta(q)})}_{q \in Q'}, \overline{Y}_{P'}, \overline{t'}_{Q''}, \overline{Y}_{P''}),$ $C' \mapsto \lambda \overline{Z}_Q.C''(\overline{Z}_{Q'}, \overline{D'_p(\overline{Z}_{\theta(p)})}_{p \in P'}, \overline{Z}_{Q''}, \overline{t}_{P''})$

Table 2. Lévy's Algorithm for Context Unification

The following properties of the algorithm were originally formulated by Lévy [12] for linear second order unification. We state them here for Context Unification without proof. The transfer of the results is based on a simple, but tedious, translation of Context Unification into his framework.

Proposition 24 Lévy. *Soundness. If $\langle \Gamma, Id \rangle$ can be transformed into $\langle \emptyset, \rho \rangle$, then ρ is a unifier for Γ.*
Completeness. If ρ is a minimal unifier for Γ then there exists a derivation of $\langle \emptyset, \rho \rangle$ from the the starting state $\langle \Gamma, Id \rangle$.

Termination. If Γ is either uniform or no variable occurs more than twice, then there exists no infinite, cycle free derivation of $\langle \Gamma, Id \rangle$ (where cycle freeness is defined up to renaming of second-order variables).

The proofs are carried out by Lévy [12] in some detail. Termination for the two occurrence case can be seen by inspection of the rules, which shows that the size of the constraint does not increase in that fragment. Notice that the rules do not introduce new constants. Thus, there are only finitely many context constraints of a given size to be considered. We can therefore avoid running into loops.