On the Complexity of Linear and Stratified : Context Matching Problems

Manfred Schmidt-Schauss, Jürgen Stuber

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

On the Complexity of Linear and Stratified Context Matching Problems

Manfred Schmidt-Schauß — Jürgen Stuber

N° 4923

Juillet 2003

THÈME 2

ISSN 0249-6399 ISRN INRIA/RR--4923--FR+ENG

On the Complexity of Linear and Stratified Context Matching Problems

Manfred Schmidt-Schauß* , Jürgen Stuber†

Thème 2 — Génie logiciel
et calcul symbolique
Projets Protheo

Rapport de recherche n° 4923 — Juillet 2003 — 28 pages

Abstract: We give algorithms for linear and for general context matching and discuss how the performance in the general case can be improved through the use of information derived from approximations that can be computed in polynomial time. We investigate the complexity of context matching with respect to the stratification of variable occurrences, where our main results are that stratified context matching is NP-complete, but that stratified simultaneous monadic context matching is in P. SSMCM is equivalent to stratified simultaneous word matching. We also show that the linear and the shared-linear case are in P and of time complexity $O(n^3)$, and that varity 2 context matching, where variables occur at most twice, is NP-complete.

Key-words: context matching, word matching, stratification, constraint solving

* Fachbereich Informatik, J.-W.-Goethe-Universität, Postfach 11 19 32, D-60054 Frankfurt, Germany. Tel: (+49)69-798-28597, Fax: (+49)69-798-28919, E-mail: schauss@ki.informatik.uni-frankfurt.de

† LORIA École des Mines de Nancy, 615 Rue du Jardin Botanique, F-54600 Villers-lès-Nancy, France. Tel: (+33)383-58 17 11, Fax: (+33)383-58 17 01, Email: stuber@loria.fr

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,

Sur la complexité de problèmes de filtrage de contextes linéaires et stratifiés

Résumé : Nous présentons des algorithmes pour le filtrage de contextes linéaires et généraux et nous montrons comment les performances dans le cas général peuvent être améliorées en utilisant l'information dérivée d'approximations calculables en temps polynomial. Nous étudions la complexité du filtrage de contextes par rapport à la stratification des occurrences de variables, et notre résultat principal est que le filtrage de contextes stratifiés est NP-complet. Par contre le filtrage simultané de contextes stratifiés monadiques, qui peut aussi être vu comme le filtrage simultané de mots stratifiés, est dans P. Nous montrons aussi que le cas linéaire et le cas partagé-linéaire sont dans P et que leur solvabilité est décidable en temps $O(n^3)$. Enfin, le cas du filtrage de contextes de varité 2 (i.e. où chaque variable peut apparaître au maximum deux fois) est montré NP-complet.

Mots-clés : filtrage de contextes, filtrage de mots, stratification, resolution de contraintes

1 Introduction

Context matching extends first-order matching by the availability of context variables which may be instantiated by a context, that is a term with a hole. Standard first-order matching allows only (term) variables, variables that may be instantiated by a first-order term. While these are restricted to the leaves of a term and match only against complete subterms, context variables can occur as monadic operators anywhere inside a term and permit the matching to descend deeply in a single step. Context matching thus allows much more freedom in the selection of subterms, which may be of use for example for querying data that is available in the form of a large term, like XML documents. For example, we may wish to select the titles of all subsections in a certain section of an XML document. We can express this by the linear context matching problem

$$X(\text{section}(\texttt{"Symbolic Data"}, Y(\text{subsection}(y, z)))) = s$$

where X and Y are context variables and y and z are first-order variables. Here each subsection of the section with title `"Symbolic Data"` will result in one solution of the matching problem with y bound to its title and z to its contents. More general queries to databases may be expressed as nonlinear but stratified context matching problems. For example, we might have a database db in a more flexible XML-like format like

$$\begin{array}{l}.(\text{book}(.(\text{author}(a_1), .(\text{title}(t_1), \text{nil}))),\\ \quad .(\text{book}(.(\text{author}(a_1), .(\text{title}(t_2), \text{nil}))),\\ \quad\quad .(\text{book}(.(\text{author}(a_2), .(\text{title}(t_3), \text{nil}))), \text{nil}))),\end{array}$$

where "." is a binary list constructor to encode variable arity. We can pick out the titles of the books of author a_1 with a query

$$X(book(Y(author(a_1)))) = db \wedge X(book(Z(title(x)))) = db$$

by looking at the substitution for x. This query is stratified, because the two occurrences of X have the same prefix of context variables above them, which is empty here.

These two examples show that context matching can be used similarly to XPath [4] matching which is used in the XSLT transformation language [3] for XML documents, which was our initial motivation to start studying context matching. In the context of XML processing there are other proposed formalisms such as regular expression matching [15] that directly take into account the variable arity of XML by allowing regular expressions over arguments. In our context this could be simulated by using argument lists plus regular restrictions on the contexts that can be instantiated for a context variable. This would allow us to both separate the list used for emulating variable arity from other function symbols and to enforce the regular expression types. Technically such regular restrictions could be expressed by Comon's membership constraints [5, 6]. We believe that the complexity results of this paper remain valid under the addition of such a restriction, however we choose not to discuss it in detail to keep the presentation simple.

Another application area for matching techniques is term rewriting [1]. In standard term rewriting the rules are implicitly extended by allowing an arbitrary context above the matching position, and preserving this context over a rewrite step. In many situations more control is needed over the positions where rules are applied, which motivates for example strategies [2]. With context matching we can achieve improved control in a more declarative way by making the context explicit in the rules. A standard rewrite rule $l \Rightarrow r$ would then become a transition rule $X(l) \Rightarrow X(r)$ that is applicable only at the root, and for finer control additional restrictions can be imposed on X, such as a sort discipline [5] or we may consider conditional rules whose conditions refer to X. Since stratified context unification is decidable [21, 20], Knuth-Bendix-like completion on rules with stratified contexts may be feasible, provided rules can be restricted in such a way that stratification is preserved and suitable termination orderings can be found. Logical calculi with context variables treated similar to a built-in theory [25, 26] would also be an interesting application. Together with the regular restrictions mentioned above this could be used for relations that, unlike equality, are sensitive to contexts, for example order relations for which certain monotonicity laws hold.

In this paper we consider context matching both from a theoretical and from a practical point of view. On the theoretical side we try to establish a sharp boundary between problems in P and NP-complete problems. To this end we consider the following problems:

Stratified context matching (SCM): For each variable the paths from the root to its occurrences have the same sequence of context variables (its *variable prefix*).

Simultaneous stratified monadic context matching (SSMCM): A stratified conjunction of equations that contains only monadic function symbols. This is equivalent to stratified word matching (SWM).

Linear context matching (LCM): Variables may occur only once.

Shared-linear context matching (SLCM): For each context variable all terms occurring immediately below that context variable are identical. This implies that in a shared representation variables are linear.

Varity 2 context matching (V2CM): Variables may occur at most twice.

We show that LCM, SLCM and SSMCM are in P, while SCM and V2CM are NP-complete. It is easily seen that context matching is in NP, hence its restrictions SCM and V2CM are in NP, too, and we need only prove NP-hardness.

On the practical side we give an algorithm for solving context matching problems using transformation rules. We also discuss how to reduce the search space of this algorithm by using sufficient criteria for finding *corresponding positions*. We give two criteria, one that approximates the problem by linearizing the variables, and another that derives a system of linear equations from the number of occurrences of function symbols.

General context matching was previously known to be NP-complete [22]. A context may be viewed as a linear second-order function with one argument, where the binder is left

implicit and the hole is the single occurrence of the bound variable. Thus context matching is a restricted form of linear higher-order matching, which was shown to be NP-complete by de Groote [9]. Here linear means that only solutions where all functions are linear, i.e. contain each of their bound variables exactly once, are considered. This has been extended to the case where a bound variable may occur at most k times by Dougherty and Wierzbicki [10], where NP-completeness holds for $k = 1$ and decidability for $k \geq 2$. An algorithm for general second-order matching is given by Huet and Lang [16]. Curien, Qian and Shi [8] give an algorithm for second-order matching that improves efficiency for the case of right-hand sides with many bound variables and few constants. Second-order and third order matching are NP-complete [7], while fourth-order matching is decidable but NEXPTIME-hard [27]. For orders above four it is still open whether higher-order matching is decidable [11]. Recently Loader has shown that higher-order beta-matching is undecidable, but this doesn't imply anything about decidability of the beta-eta case [18].

Hirata, Yamada and Harao [14] have studied the complexity landscape of the second-order matching problem with respect to several restrictions, i.e. number of second-order variables, number of occurrences of variables, ground, function-free, but not stratification.

Our interest for the stratified fragment has been motivated by the results on the decidability of stratified context unification [22, 20, 21]. Other fragments where unification is decidable are the varity 2 fragment [17][1] and the two-context-variable fragment [23], whose corresponding matching problems we also consider.

2 Preliminaries

We assume a fixed infinite set $\mathcal{X}$ of (individual) variables, a fixed infinite set $\mathcal{C}$ of context variables, and a fixed set Σ of function symbols of fixed arity. We will use x, y, z for individual variables, X, Y, Z for context variables, a, b, c, d for constants and f, g, h for other function symbols.

Context terms (or just *terms*) are constructed from the variables and function symbols in the usual first-order way, where context variables are considered as monadic function symbols. A *position* in a term is a sequence of positive integers that indicates the arguments where one descends in a term to reach the position. For example, the a is at position 2.1 in $f(b, g(a))$ and f is at position ϵ, the root position.

A *context* is a term with a single occurrence of the special operator $\Box$, the *hole*. To emphasize that a term C is a context we write $C[\Box]$ or just $C[]$. In particular we will most of the time write $X[]$ for $X(\Box)$. A context $C[]$ may be applied to a term t, written $C[t]$, and the result is the term consisting of C with $\Box$ replaced by t. We specify the position p of a subterm t inside a context by $C[t/p]$. A *subcontext* of a term t is any context $C[]$ such that $t = D[C[t']]$ for some context $D[]$ and term t'. We will use s, t, l, r for terms or contexts, and C, D for contexts.

[1] Note that the proof of decidability of stratified context unification in this paper is flawed, since the termination proof has a gap, as acknowledged by the author.

We also consider *multi-contexts*, which are terms with an arbitrary number of occurrences of the hole. A *proper multi-context* is a multi-context with at least two holes. We write multi-contexts as $C[\Box, \ldots, \Box]$. Such a multi-context may be viewed as a second order lambda term $\lambda x_1 \ldots \lambda x_n\, C[x_1, \ldots, x_n]$ where the bound variables $x_1, \ldots, x_n$ occur each exactly once and in left-to-right order. So $f[\Box, \Box]$ corresponds to $\lambda x \lambda y\, f(x, y)$ but not to $\lambda x \lambda y\, f(y, x)$. We may replace a subset of holes by terms by writing for instance $C[\ldots, s_1, \ldots, s_2, \ldots]$ where the dots stand for holes. Where we need to be more precise we write $\Box^k$ for k holes and $\Box_i$ for the i-th hole. Note that we do not allow variables that stand for multi-contexts, and that there are no holes in $C[\Box, \ldots, \Box]$ besides the ones inside the square brackets.

A *substitution* is a mapping from individual variables to terms and from context variables to contexts, such that all but finitely many individual variables are mapped to themselves, and all but finitely many context variables are mapped to themselves applied to the hole. Substitutions are extended to terms as follows:

$$f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma)$$
$$x\sigma = \sigma(x)$$
$$(X(t))\sigma = \sigma(X)[t\sigma].$$

Note that substitutions cannot introduce or remove holes.

A *term equation* is an equation between context terms, and a *context equation* is an equation between contexts. An *equation* $s \approx t$ is an equation between multi-contexts, which generalizes term and context equation. A substitution σ is a *solution* of an equation $s \approx t$ if $s\sigma = t\sigma$. A *matching equation* is an equation $s \approx t$ where t doesn't contain any free variables. Thus σ is a solution of a matching equation $s \approx t$ if $s\sigma = t$. A *matching problem instance* is a conjunction of matching equations, and the *size* of an instance is the number of function and variable symbols it contains. Two instances are *equivalent* if they have the same set of solutions. A *matching problem* is a set of instances, and the question for a given instance is whether it is solvable. We will also call the left-hand sides of an instance the *query* and the right-hand sides the *data*. The *query complexity* (resp. the *data complexity*) of a matching problem or an algorithm for it is the complexity where only the query (resp. the data) varies and the data (resp. the query) is fixed.

Proposition 1 *Suppose there exists at least one symbol of arity* ≥ 2. *Then*

1. *for any conjunction of term equations there exists an equivalent term equation, and*
2. *for any term equation there exists an equivalent context equation.*

Proof: Let h be a function symbol of arity $k \geq 2$.

(1) For a conjunction $s_1 \approx t_1 \wedge \ldots \wedge s_n \approx t_n$ take

$$h(s_1, \ldots, s_{k-1}, h(s_k, \ldots, h(\ldots, s_n)\ldots)) \approx h(t_1, \ldots, t_{k-1}, h(t_k, \ldots, h(\ldots, t_n)\ldots)).$$

(2) For a given term equation $s \approx t$ take $h(\Box, s, \ldots, s) \approx h(\Box, t, \ldots, t)$ as the context equation. □

Proposition 2 *Suppose there exist a countably infinite number of constant symbols. Then any context matching equation can be reduced to an equivalent term equation.*

Proof: Replace the hole on both sides of the equation by a constant not occurring in the problem. □

If we have an equation between multi-contexts, e.g.

$$C[f(C_1[\Box], C_2[\Box])] \approx D[g(D_1[\Box], D_2[\Box])],$$

then either $f \neq g$ and there is no solution, or we can split this equation into the equations $C[] \approx D[]$, $C_1[] \approx D_1[]$ and $C_2[] \approx D_2[]$, reducing the multi-context equation to an equivalent conjunction of context equations. In this way we can eliminate all proper multi-contexts, see Proposition 19 for a formal treatment.

A context matching problem instance is called *linear* if each variable occurs at most once and *monadic* if all function symbols occurring in it have an arity of at most one.

The *variable prefix* of an occurrence in a term is the sequence of context variables on the path from the root down to and including that occurrence.[2] A context matching problem instance is called *stratified* if for each individual or context variable all occurrences of that variable in the problem have the same variable prefix.

Example 3 *The problem*

$$X(f(Y(g(a,x)), Y(g(x, Z(z))))) \approx f(g(a,b),b)$$

is stratified, X having the prefix X, Y the prefix XY, x the prefix XYx, Z the prefix XYZ and z the prefix $XYZz$.

Example 4

$$X(Y(\Box)) \approx f(g(\Box)) \wedge Z(X(\Box)) \approx h(f(\Box))$$

is not stratified, because X occurs with the variable prefixes X and ZX.

Proposition 5 *A context matching problem has at most exponentially many solutions.*

Proof: Suppose $l \approx r$ is an instance of context matching. In a solution each of the at most $|l|$ context variables in l has to be instantiated by a subcontext in r, of which there are $O(|r|^2)$. Once context variables are instantiated there is at most one solution for the individual variables, so there are at most $O(|r|^{2|l|})$ solutions. □

Example 6 *This example shows that there may be exponentially many incomparable solutions for a context matching problem, even in the linear monadic case that is included in all*

[2] Note that the standard definition excludes variables at the occurrence itself. This definition is equivalent for the definition of stratification and simplifies the treatment of SSMCM below.

problems treated in this paper except for the k-context variable fragment. For simplicity we omit brackets, since all operators are monadic. Let the multi-context equation be

$$X_1Y_1hX_2Y_2h\dots hX_nY_n(a) \approx ghgh\dots hg(a).$$

Since there is the same number of h's on both sides, the context variables can only match the g's, so there are two possible solutions for each pair X_i, Y_i of context variables:

X_i	Y_i
$\Box$	$g(\Box)$
$g(\Box)$	$\Box$

Since the solutions can be combined independently, the number of solutions is 2^n, and thus exponential.

3 Linear Context Matching is in P

NAME: Linear Context Matching (LCM)
INSTANCE: A linear context matching equation $s \approx t$.
QUESTION: Is $s \approx t$ solvable?

Linear context matching is an interesting special case of context matching where variables can be regarded as wildcards for contexts. We can solve linear context matching problems by dynamic programming. We build a table which for every pair of a subterm s of the left-hand side and a subterm t of the right hand side of the problem records whether s matches t. We assume that for every subterm we know whether there is a hole in it; this information can be computed in linear time. The table is built recursively from the bottom up:

- If s is an individual variable then the answer is yes if and only if there is no hole in t.
- If $s = f(s_1, \dots, s_m)$ and $t = g(t_1, \dots, t_n)$ then the answer is yes if and only if $f = g$, $m = n$ and s_i matches t_i for $1 \leq i \leq m$.
- If $s = \Box$ then the answer is yes if and only if $t = \Box$.
- If $s = X(s')$ then the answer is yes if and only if s' matches some subterm of t.

Let n be the size of the problem. We have $O(n^2)$ table entries of constant size, and to compute each entry we need at most $O(n)$ steps. Hence the algorithm uses $O(n^3)$ time and $O(n^2)$ space, and linear context matching is in P. Note that this would no longer hold if we were to allow variables for multi-contexts.

4 A Transformation Algorithm for Context Matching

NAME: Context Matching (CM)
INSTANCE: A context matching problem instance P.
QUESTION: Is P solvable?

Even though context matching is NP-hard in general, it is nevertheless interesting to develop practical algorithms for it, as we hope that many practically interesting instances will be solvable easily enough to be useful. We will first give a general algorithm for context matching in this section, and then in the following sections modify and extend it to more efficiently solve some easy cases, and discuss the complexity of special cases of context matching.

We give rules for context matching problems consisting of a conjunction of equations between multi-context terms whose right-hand side is ground. Rules are applied modulo associativity and commutativity (AC) of $\wedge$, i.e., ignoring the order of equations in a conjunction. When a rule fails it rewrites its left-hand side to $\bot$. We first give don't-care nondeterministic rules that may be applied eagerly:

Decompose $f(s_1, \ldots, s_m) \approx t \to s_1 \approx t_1 \wedge \ldots \wedge s_m \approx t_m$

if $t = g(t_1, \ldots, t_n)$, $f = g$ and $m = n$; otherwise fail.

Hole Decompose $\Box \approx C[] \to \top$

if $C[] = \Box$; otherwise fail.

Multi-Context Clash $C[\Box^m] \approx D[\Box^n] \to \bot$

if $m \neq n$.

Term Merge $x \approx s \wedge x \approx t \to x \approx s$

if $s = t$; otherwise fail.

Context Merge $X[] \approx C[] \wedge X[] \approx D[] \to X[] \approx C[]$

if $C[] = D[]$; otherwise fail.

Context Eliminate $X[] \approx C[] \wedge X(s) \approx t \to X[] \approx C[] \wedge s \approx t'$

if $s \neq \Box$ and $t = C[t']$; otherwise fail.

Note that $s \approx t'$ is an equation between multi-contexts, and that t' can be computed in linear time.

Finally we need a don't-know nondeterministic rule that requires backtracking over all applications:

Variable Split $X(s) \approx t \to X[] \approx C[] \wedge s \approx t'$

if $s \neq \Box$ and $t = C[t']$.

A context matching problem is in *solved form* if all its equations have the form $x \approx t$ or $X[] \approx C[]$ and each variable occurs at most once. There is an obvious one-to-one correspondence between solved forms and substitutions.

Example 7 *This example illustrates typical execution possibilities for the transformation algorithm for context matching. As example consider the equation*

$$X(f(Y(a), Y(g\Box))) \approx f(ga, f(a, g\Box)).$$

We show two possibilities for an execution:

1.	*Variable Split:*	$X[] \approx \Box \wedge f(Y(a), Y(g\Box)) \approx f(ga, f(a, g\Box))$
	Decompose:	$X[] \approx \Box \wedge Y(a) \approx ga \wedge Y(g\Box) \approx f(a, g\Box)$
	Variable Split:	$X[] \approx \Box \wedge Y[] \approx g\Box \wedge a \approx a \wedge Y(g\Box) \approx f(a, g\Box)$
	Decompose:	$X[] \approx \Box \wedge Y[] \approx g\Box \wedge Y(g\Box) \approx f(a, g\Box)$
	Context Eliminate:	*Fail*
2.	*Variable Split:*	$X[] \approx f(ga, \Box) \wedge f(Y(a), Y(g\Box)) \approx f(a, g\Box).$
	Decompose:	$X[] \approx f(ga, \Box) \wedge Y(a) \approx a \wedge Y(g\Box) \approx g\Box$
	Variable Split:	$X[] \approx f(ga, \Box) \wedge Y[] \approx \Box \wedge a \approx a \wedge Y(g\Box) \approx g\Box$
	Decompose:	$X[] \approx f(ga, \Box) \wedge Y[] \approx \Box \wedge Y(g\Box) \approx g\Box$
	Context Eliminate:	$X[] \approx f(ga, \Box) \wedge Y[] \approx \Box \wedge g\Box \approx g\Box$
	Decompose:	$X[] \approx f(ga, \Box) \wedge Y[] \approx \Box \wedge \Box \approx \Box$
	Hole Decompose:	$X[] \approx f(ga, \Box) \wedge Y[] \approx \Box$

We now prove the correctness of the algorithm.

Lemma 8 *The don't-care nondeterministic rules preserve the set of solutions.*

Proof: By inspection. □

Lemma 9 *Let P be a context matching problem and let $P_1, \ldots, P_n$ be the context matching problems obtained from P by all possible applications of Variable Split to a matching equation. Then a substitution is a solution of P if and only if it is a solution of some P_i for $1 \leq i \leq n$.*

Proof: The context variable at the root must be substituted by some context. For each possible choice of a context there is a corresponding application of Variable Split. □

Lemma 10 *If a context matching problem is not in solved form then one of the rules is applicable.*

Proof: If there is an equation with a different number of holes on the left- and on the right-hand side then we apply Multi-Context Clash. After exhaustive application all equations have the same number of holes on both sides.

If there is an equation with a function symbol at the root of the left-hand side then we can apply Decompose, or if the left-hand side is a hole we apply Hole Decompose. After exhaustive application of all the rules above all equations in the problem have the form $x \approx t$ or $X(s) \approx t$.

To an equation $X(s) \approx t$ Variable Split can be applied. After exhaustive application of all the rules above all equations in the problem have the form $x \approx t$ or $X[] \approx t$.

If a variable x or X occurs multiple times then Term Merge or Context Merge can be applied, respectively.

We conclude that if none of the rules is applicable then the problem is in solved form. □

Note that multi-contexts are converted into terms and contexts by the combination of Variable Split and Decompose, which eventually separates the holes occurring in parallel subterms. Due to the don't-know nondeterminism of Variable Split this is rather inefficient; we will discuss a better method in Subsection 5.4.

Lemma 11 *The application of rules terminates after at most $O(n)$ steps.*

Proof: This follows from Proposition 14 in the next section where we show termination with respect to an extended set of rules. □

Theorem 12 *The algorithm that applies the don't-care nondeterministic rules above eagerly and backtracks over Variable Split finds solved forms for all solutions of the context matching problem.*

Due to the don't-know nondeterminism inherent in Variable Split this algorithm is exponential, and in particular it may produce exponentially many solutions. For certain special cases discussed below solvability will be decidable in polynomial time, while the number of solutions remains exponential in the worst case. If in these cases we are only interested in solvability and want to avoid exponential behavior due to the number of solutions we may remove equations $x \approx t$ and $X[] \approx C[]$ where x or X occurs only once in the problem.

It is interesting to compare the working of our rules to the rules Simplification, Imitation and Projection that are used in the more general higher-order cases [9, 16]. Simplification is essentially a Decompose with respect to a function symbol, while Imitate and Project together correspond to Variable Split. Applied to context matching they would construct a context step by step, Imitation would add a function symbol at the bottom and Project would put in the hole. Eliminate and Merge are implicit in the handling of substitutions.

Curien, Qian and Shi [8] consider a variant of the second order matching algorithm that replaces the Imitate and Project rules by a rule FR that handles bigger contexts in a single application. They use occurrences of bound variables to limit the search space. In our setting this corresponds to the fact that holes must match, as holes can be seen as implicitly

bound variables. This is one case where some of the don't-know nondeterminism leading to exponential time complexity can be avoided. We discuss this and other polynomial criteria in the next section.

5 Avoiding Search

We describe polynomial methods to detect corresponding positions in the left- and right hand side of match equations, and a method to exploit this information.

5.1 Corresponding Positions

Assume as given a context matching problem P, a match equation $s \approx t$, and positions p_s in s and p_t in t. We say that p_s and p_t *correspond* for a substitution σ if $s[\Box/p_s]\sigma = t[\Box/p_t]\sigma$ and $(s|_{p_s})\sigma = (t|_{p_t})\sigma$. We say that p_s and p_t are *corresponding positions* in a match equation $s \approx t$ if p_s and p_t correspond for all solutions σ of $s \approx t$. Note that for an unsatisfiable problem all positions on the left correspond to all positions on the right. If we know that two positions p_s and p_t correspond in a match equation $s \approx t$ of a context matching problem P then the following rule preserves the set of solutions and can hence be used eagerly in a don't-care nondeterministic way. For termination it is important to exclude trivial cases, namely the root position and the positions of holes, which we call *trivially corresponding*.

Split $$s[s'/p_s] \approx t[t'/p_t] \rightarrow s[\Box/p_s] \approx t[\Box/p_t] \wedge s' \approx t'$$

if p_s and p_t are nontrivially corresponding.

Note that s, s', t and t' may be multi-contexts, and that we assume that the holes not mentioned are handled appropriately.

Proposition 13 *The rule Split preserves the set of solutions.*

Proof: By definition of corresponding position. □

Proposition 14 *The application of the rules in Section 4 and Split terminates after at most $O(n)$ steps.*

Proof: As a measure we may take the sum of the number of function symbols and variables in the problem and the number of arguments of function symbols and variables not equal to a hole. The don't-care rules decrease the first term of the sum and don't increase the second, and the Split rules decrease the second term and preserve the first. Thus derivations terminate after at most $O(n)$ steps. □

Theorem 15 *If for a class of context matching problem instances $\mathcal{P}$ that is closed under rule application there exists an oracle in $O(n^k)$ that always finds some corresponding position, then the matching algorithm runs in $O(n^{k+1})$ and $\mathcal{P}$ is in P.*

This implies that for such a problem there always exist corresponding positions, which is in general not the case. We continue by discussing methods to compute corresponding positions.

5.2 Using Linearization for Detection of Corresponding Positions

Given a match equation $s \approx t$, we can test for unsolvability by linearizing the left hand side: First make all variables (first order and context variables) in s distinct, giving $\overline{s}$. Then check $\overline{s} \approx t$ using the method for linear context matching in Section 3. If this match equation is unsolvable, then, of course, $s \approx t$ is unsolvable. Since the solvability test for linear problems is implemented by dynamic programming, we can use it to detect corresponding positions by checking for rows or columns with a single entry. Since the linearization may enlarge the set of solutions this is only a sufficient criterion for corresponding positions.

Example 16 *Given a match equation*

$$Y(f(X(Z(g(y_1, y_2, a)), Z(h(y, y, b))), X(c))) \approx t,$$

we may approximate the search for a subterm matching $f(\dots)$ in the right-hand side t by searching for a subterm starting with f whose first argument has an occurrence of a g, whose second argument an occurrence of an h, etc. The linearization test covers exactly this intuition; it disregards that the positions of the sub-subterms are related by for example the non-linear context variable Z.

Next we consider the number of function symbols but not their position, which is orthogonal to linear context matching.

5.3 Using Linear Equations for Detecting Corresponding Positions

Let $P = s_1 \approx t_1 \wedge \ldots \wedge s_n \approx t_n$ be a context matching problem. Then we can compute a linear equation system for the number of occurrences of function symbols in a match. Let $occ(f, s)$ be the number of occurrences of the function symbol f in the term s. Given f and a particular equation $s_i \approx t_i$, define the following integer equation:

$$occ(f, s_i) + occ(x_1, s_i) * x_{1,f} + \cdots + occ(x_m, s_i) * x_{m,f} = occ(f, t_i) \qquad (EQ(f, i))$$

where $x_{j,f}$ represents the number of occurrences of symbol f in $\sigma(x_j)$ and x_j is a first-order or context variable. For every symbol f occurring in the context matching problem there is a system of linear integer equations

$$EQ(f, i) \qquad \text{for } i = 1, \ldots n.$$

If the context matching problem P is solvable then for every f occurring in P the system of linear equations is solvable in the nonnegative integers, as a system of Diophantine equations. Deciding whether such a solution exists is NP-complete [13], while it is possible to decide

in polynomial time whether a solution in the integers exists [24]. Note that despite NP-completeness it may in practice still be useful to consider the Diophantine case for certain problem classes.

We can now use this necessary criterion for solvability to reduce the nondeterminism of the Split rules. We may try all splits for a fixed position in the left-hand (resp. right-hand) side, and if there is only one position on the other side that leads to a solvable problem then the two are corresponding positions.

An interesting special case is that a symbol f occurs the same number of times on the left- and on the right-hand side, which implies that it doesn't occur in substitutions and $x_{j,f} = 0$ for all j. In that case, if there is a solution at all, we can establish a one-to-one correspondence between symbols on the left and on the right and obtain corresponding positions.

We may also obtain more general partial solutions of the system of Diophantine equation that fix the number of symbol occurrences for certain variables, which may help to detect corresponding positions using more advanced algorithms.

5.4 Deriving New Corresponding Positions from Known Ones

If one of the methods in the previous paragraphs detects corresponding positions, then in general multi-contexts with two or more holes may be introduced. Instead of using the rules in Section 4 directly on multi-context equations, which requires the don't-know nondeterministic use of Variable Split, we propose in this section don't-care nondeterministic rules that can transform any multi-context equation into equations between multi-contexts with at most one hole. To this end we exploit that the root positions and the positions of corresponding holes always correspond trivially, and that correspondences between positions may imply new correspondences. For example, correspondences between positions propagate across function symbols, which together with the correspondence of holes leads to a don't-care rule that does bottom-up decomposition.

The effect of these rules in combination with the simplification rules in section 4 will be a don't-care simplification such that the terms in the equations have at most one hole, and the hole can only occur in a subterm of the form $X[]$.

Proposition 17 *Let $s \approx t$ be a match equation, and let $p_s = p'_s.i$ and $p_t = p'_y.j$ be positions in s and t such that s has a function symbol f at p'_s. Then p_s and p_t are corresponding positions in $s \approx t$ if and only if p'_s and p'_t are corresponding positions and $i = j$.*

We can use this property to decompose function symbols just above corresponding holes, by applying Split followed by Decompose at such a position:

Bottom Decompose $C[\Box^k, f(s_1, \ldots, s_n), \Box^l] \approx D[\Box^k, f(t_1, \ldots, t_n), \Box^l]$
$\rightarrow C[\Box^{k+l+1}] \approx D[\Box^{k+l+1}] \wedge s_1 \approx t_1 \wedge \ldots \wedge s_n \approx t_n$
if $s_j = t_j = \Box_i$ for some i, j, $1 \leq j \leq n$.

Bottom Clash $C[\Box^{k_1}, f(s_1, \ldots, s_m), \Box^{l_1}] \approx D[\Box^{k_2}, g(t_1, \ldots, t_n), \Box^{l_2}] \to \bot$

if $s_{j_1} = \Box_i$ for $1 \leq j_1 \leq m$, $t_{j_2} = \Box_i$ for $1 \leq j_2 \leq n$, and $f \neq g$ or $j_1 \neq j_2$.

After applying these rules exhaustively there will be only context variables immediately above holes on the left-hand side.

A more elaborate criterion allows us to eliminate multi-contexts with more than one hole.

Proposition 18 *Let p_{s1} and p_{t1} and p_{s2} and p_{t2} be distinct pairs of corresponding positions in $s \approx t$, and let p_s be the longest common prefix of p_{s1} and p_{s2} and p_t the longest common prefix of p_{t1} and p_{t2}. Then p_s and p_t are corresponding positions in $s \approx t$.*

So in particular the longest common prefixes of all holes in s and t are corresponding, and for two or more holes they are nontrivial. Since only function symbols can have holes in distinct arguments, there must be function symbols of arity ≥ 2 at these positions and we may again exploit this to derive a decompose rule:

Multi-context Decompose

$C[f(C_1[\Box^{k_1}], \ldots, C_n[\Box^{k_n}])] \approx D[f(D_1[\Box^{k_1}], \ldots, D_n[\Box^{k_n}])]$
$\to C[\Box] \approx D[\Box] \wedge C_1[\Box^{k_1}] \approx D_1[\Box^{k_1}] \wedge \ldots \wedge C_n[\Box^{k_n}] \approx D_n[\Box^{k_n}]$

if there exist $i, j \in \{1, \ldots, n\}$ with $i \neq j$, $k_i \geq 1$ and $k_j \geq 1$.

Multi-context Clash

$C[f(C_1[\Box^{k_1}], \ldots, C_m[\Box^{k_m}])] \approx D[g(D_1[\Box^{l_1}], \ldots, D_n[\Box^{l_n}])]$
$\to \bot$

if (i) there exist $i, j \in \{1, \ldots, m\}$ with $i \neq j$, $k_i \geq 1$ and $k_j \geq 1$, (ii) there exist $i, j \in \{1, \ldots, n\}$ with $i \neq j$, $l_i \geq 1$ and $l_j \geq 1$, and (iii) $f \neq g$ or $k_i \neq l_i$ for some $i \in \{1, \ldots, m\}$.

Here the conditions (i) and (ii) ensure that the positions of f and g, respectively, are the outermost function symbols where the paths to the holes branch. Condition (iii) captures differences that will prevent a solution.

Applying the last two rules eagerly will remove all proper multi-contexts and leave only term and context equations:

Proposition 19 *Let $C[\Box^m] \approx D[\Box^n]$ be a multi-context equation. It is either unsolvable, or there exists an equivalent conjunction of context and term equations. This can be computed in linear space and time.*

Example 20 *Let the equation be*

$$X(f(b, Y(g\Box))) \approx f(a, f(b, f(c, gg\Box))).$$

By the simple criterion that if a constant occurs exactly once both in the left- and in the right-hand side then its positions correspond, the two occurrences of b give us the corresponding positions 1.1 *and* 2.1. *Application of the rule Split gives*

$$X(f(\Box, Y(g\Box))) \approx f(a, f(\Box, f(c, gg\Box))) \wedge b \approx b,$$

where the equation $b \approx b$ can be eliminated by Decompose.

The longest common prefix of the positions of the two holes are corresponding positions, hence we can apply Multi-context Decompose:

$$X[] \approx f(a, \Box) \wedge Y(g\Box) \approx f(c, gg\Box)$$

The rule Bottom Decompose now allows us to directly compute:

$$X[] \approx f(a, \Box) \wedge Y[] \approx f(c, g\Box) \wedge \Box \approx \Box$$

which immediately gives the solved system:

$$X[] \approx f(a, \Box) \wedge Y[] \approx f(c, g\Box)$$

6 Stratified Context Matching Is NP-complete

NAME: Stratified Context Matching (SCM)
INSTANCE: A stratified context matching problem instance P.
QUESTION: Is P solvable?

We now continue with an investigation of the complexity of special cases of context matching, in particular stratified context matching in this section, and simultaneous stratified monadic context matching in the following section, which illuminate the boundary between polynomial and NP-complete cases.

To show NP-hardness of stratified context matching we reduce 3SAT to SCM.

NAME: 3SAT
INSTANCE: A finite set S of propositional clauses with exactly 3 literals.
QUESTION: Is S satisfiable?

3SAT is known to be NP-complete [13]. Let $c_1, \ldots, c_m$ be the clauses and $x_1, \ldots, x_n$ the propositional variables in an instance of 3SAT. We encode this as an instance of SCM with $m+1$ equations

$$s_0 \approx t_0 \wedge s_1 \approx t_1 \wedge \ldots \wedge s_m \approx t_m$$

where $s_0 \approx t_0$ is a dummy equation and $s_i \approx t_i$ corresponds to clause c_i. The dummy equation restricts matchings so that they correspond to boolean valuations, and equation i matches with such a valuation if and only if the valuation satisfies clause i. The terms in each equation consists of $n+1$ *layers*, n layers for the variables and one at the bottom with

constants that encodes whether clauses are satisfied. Each layer in an equation is either an *active* layer that switches between subterms in the right-hand side or a *passive* layer that doesn't. In equation i layer j is active when variable j occurs in clause i, otherwise it is passive. The dummy equation consists only of passive layers. The dummy equation assures that each layer allows only two different matches for two context variables in that layer (on top of each other), where every match represents a truth value. In the case of an active layer these matches select different subterms on the right-hand side for matching in lower layers, which allows us to test for satisfiability. In the bottom layer we need to distinguish the choices made in the layers above, in order to determine whether the clause is satisfied or not. We record the truth values of literals in clause c_i in a string containing $*$ in the j-th position when there is no literal containing x_j, 0 when there is a literal containing x_j that is false under the chosen assignment, and 1 when there is a literal containing x_j that is true under the chosen assignment. For the dummy equation we construct a satisfiable bottom layer, while for the other equations we chose a satisfiable bottom layer for a certain subterm only when the string leading to that subterm contains at least one 1, meaning that the clause is satisfied in this branch. Figure 1 gives the global structure of this encoding, and Figure 2 illustrates the internal workings of the gadgets.

To formalize this we recursively define

$$s_{ij} = \begin{cases} f(X_j(g(g(x_{ij}, Y_j(s_{i,j+1})), y_{ij}))) & \text{if } x_j \text{ occurs in } c_i, \text{ and} \\ f(X_j(Y_j(s_{i,j+1}))) & \text{otherwise.} \end{cases}$$

$$t_{i,\alpha} = \begin{cases} f(g(g(g(c, t_{i,\alpha 0}), g(t_{i,\alpha 1}, c)), c)) & \text{if } x_{|\alpha|+1} \text{ occurs positively in } c_i, \\ f(g(g(g(c, t_{i,\alpha 1}), g(t_{i,\alpha 0}, c)), c)) & \text{if } x_{|\alpha|+1} \text{ occurs negatively in } c_i, \text{ and} \\ f(g(t_{i,\alpha *}, c)) & \text{otherwise.} \end{cases}$$

where $0 \leq i \leq m$ and $1 \leq j = |\alpha| + 1 \leq n$. For $i = 0$ we always choose the (passive) otherwise-case. For the leaves we use $s_{i,n+1} = a$ and

$$t_{i,\alpha} = \begin{cases} a & \text{if } i = 0 \text{ or } \alpha \text{ contains at least one 1} \\ b & \text{otherwise.} \end{cases}$$

Finally the stratified context matching problem P is

$$s_{01} \approx t_{0,\epsilon} \wedge \ldots \wedge s_{m1} \approx t_{m,\epsilon}.$$

It remains to show that P has a solution if and only if the original instance of 3SAT is solvable.

First we show that $s_{i,j}$ can only match $t_{i,\alpha}$ for $1 \leq i \leq m$ and some α with $j = |\alpha| + 1$. This holds because there are the same number of fs in the left- and in the right-hand side of the dummy equation, hence none can be matched by a context variable and the layers are separated. Furthermore, the main paths must proceed into the nontrivial subterms, as f doesn't match the constant c. Let us now consider a single layer in the dummy equation.

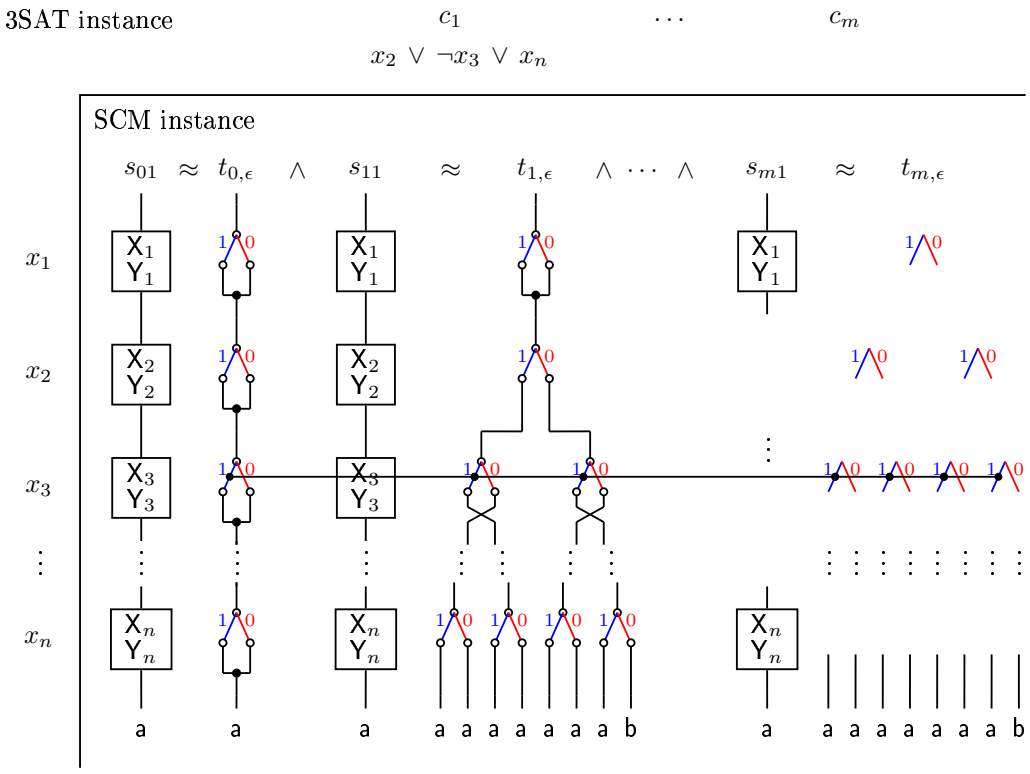

Figure 1: Global Structure of the Encoded 3SAT Problem

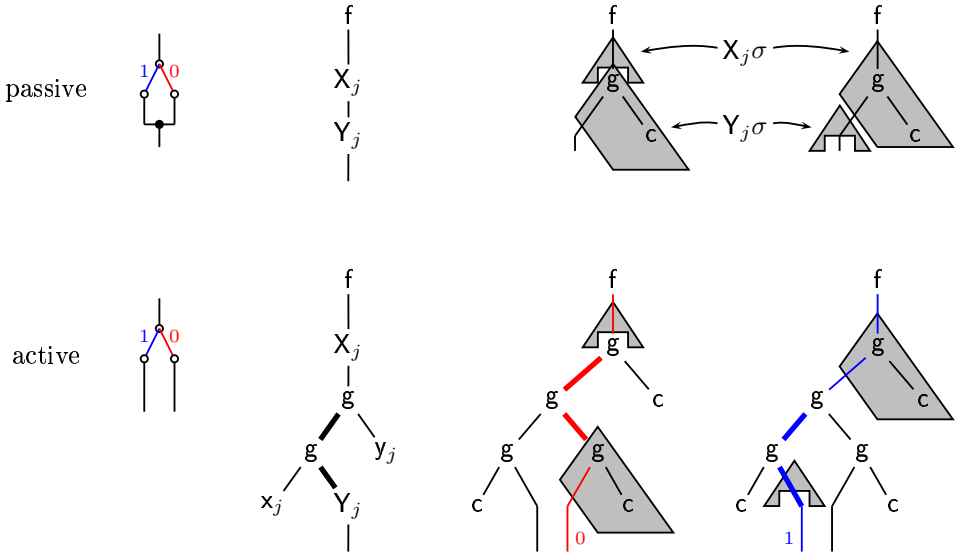

Figure 2: Gadgets for the Reduction of 3SAT to SCM

There are two possible matches in layer j (see Figure 2):

$$X_j = \Box \qquad Y_j = g(\Box, c) \tag{1}$$
$$X_j = g(\Box, c) \qquad Y_j = \Box \tag{2}$$

These are the only matches for a passive layer. Now let us look what happens in a positive active layer: In case (1) the next lower layer $s_{i,j+1}$ is matched against $t_{i,\alpha 1}$, while in case (2) it is matched against $t_{i,\alpha 0}$. For a negative active layer it is vice-versa. Hence a match of the form (1) corresponds to assigning true to the logical variable x_j, and a match of the form (2) corresponds to assigning false to x_j. This in mind we see that at the leaves α reflects the truth values of the literals in the clause. If these make the clause true, we have put the solvable matching problem $a \approx a$ at the leaf, otherwise the unsolvable $a \approx b$. Hence each match corresponds to a satisfying truth assignment and vice-versa.

If we look at the size of P we see that each right-hand side of an equation branches at most at three points, since each clause contains at most three distinct variables. This gives a factor of $2^3 = 8$ but doesn't lead to exponential growth. The depth is proportional to the number of variables, and the number of equations to the number of clauses, hence the size of the context matching problem is quadratic in the size of the 3SAT instance.

We conclude that 3SAT can be reduced to SCM, and obtain:

Theorem 21 *SCM is NP-complete.*

7 Simultaneous Stratified Monadic Context Matching Is in P

NAME: Simultaneous Stratified Monadic Context Matching (SSMCM)
INSTANCE: A stratified monadic context matching problem instance P.
QUESTION: Is P solvable?

Simultaneous Stratified Monadic Context Matching (SSMCM) is context matching restricted to function symbols of arity at most one. In this case conjunctions cannot be coded in a single equation, hence we allow them explicitly. We will show that simultaneous stratified monadic context matching (SSMCM) is in P.

We may easily eliminate as unsolvable any problem containing an equation that has two distinct constants at the bottom of its left- and right-hand side, and assume from now on that all equations are context equations and have a hole at the bottom. In particular we replace an individual variable at the bottom of a left-hand side by a context variable applied to a hole. With these remarks it becomes obvious that SSMCM is equivalent to simultaneous stratified word matching, where the context variables become ordinary variables, the hole at the bottom is omitted, and where the prefix of a variable consists of all variables to its left plus itself. To simplify the notation we will consider terms as words in the rest of this section. We write ε for the empty word.

The crucial property that leads to a polynomial algorithm in this case is that there are no branchings in the terms. For any equation $w_1X_1 \dots w_nX_nw_{n+1} \approx w$ we know that a solution σ must satisfy $|(X_1 \dots X_n)\sigma| = |w|-|w_1 \dots w_nw_{n+1}|$, hence we call $|w|-|w_1 \dots w_nw_{n+1}|$ the *substitution length* of this equation. Equations with negative substitution length and conjunctions containing two equations with the same context variables but different substitution lengths are inconsistent.

We give a transformation algorithm to solve SSMCM that consists of simplification rules and a variable elimination rule that does the major work. The rules work bottom-up (right-to-left) and carefully merge possible instantiations of variables in order to stay within polynomial size.

We start by describing the form of a problem after exhaustive application of the simplification rules. Such a problem P is a conjunction of disjunctions of conjunctions of match equations. More precisely, it has the form

$$P_{s_1} \wedge \dots \wedge P_{s_n}$$

where each P_{s_i} is a disjunction of conjunctions of equations such that the rightmost variables in the left-hand sides of equations in P_{s_i} all have the variable prefix s_i. All the s_i are distinct, but there may be conjuncts P_{s_i} and P_{s_j} such that s_i is a proper prefix of s_j. The number of variable prefixes is bounded by the number of variable prefixes in the initial problem, which in turn is bounded by the number of variables and hence the size of the problem. For each prefix s the conjunct P_s has the form

$$P_{s,0} \vee \dots \vee P_{s,k}$$

where $P_{s,i}$ is a conjunction of equations whose substitution length is i. For the bound k we can use the largest substitution length in the initial problem, which is bounded by the size of the problem.

Each conjunction $P_{s,i}$ is either $\bot$, $\top$, or a conjunction of equations whose sides are prefixes of original equations that contain the variable prefix s. These are exactly those equations that contain the prefix s, and which don't contain a longer prefix that is still present in another disjunction.

This form includes simplified initial problems that are just a conjunction of equations. In this case the disjunctions each contain exactly one nonempty conjunction of equations with the same maximal prefix, and the other disjuncts for this prefix may be assumed to be $\bot$.

The transformation algorithm consists of simplification rules that are applied eagerly anywhere inside the problem, where the Restricted Distributivity rule is only applied when no other simplification rule is applicable, and the Variable Elimination rule that is applied to a full disjunction and only when no simplification rule is applicable. The rules are iterated until one of the solved forms $\top$ or $\bot$ is obtained. Here rules are applied modulo associativity and commutativity of $\wedge$ and $\vee$, and failure again means rewriting to $\bot$. The simplification rules are the rules for idempotency and the laws of true and false together with the following rules:

Delete $\varepsilon \approx t \to \top$

if $t = \varepsilon$; otherwise fail.

Bottom Decompose $sf \approx tg \to s \approx t$

if $f = g$; otherwise fail,

where f and g are function symbols.

Substitution Length Clash 1 $s \approx t \to \bot$

if the substitution length of $s \approx t$ is negative.

Substitution Length Clash 2 $s_1 \approx t_1 \wedge s_2 \approx t_2 \to \bot$

if s_1 and s_2 have the same rightmost variable but $s_1 \approx t_1$ and $s_2 \approx t_2$ differ in substitution length.

Note that the Substitution Length Clash rules are only needed in an initial simplification, since the following two rules do not generate subproblems that are inconsistent with respect to substitution length.

Restricted Distributivity

$$(P_{s,0} \vee \ldots \vee P_{s,k}) \wedge (P'_{s,0} \vee \ldots \vee P'_{s,k}) \to \bigvee_{0 \le i \le k} (P_{s,i} \wedge P'_{s,i})$$

where all equation have the same rightmost variable.

Restricted Distributivity combines disjunctions whose variable prefix has become equal after Variable Elimination and keeps only conjunctions with consistent substitution length.

We say a problem is *simplified* if these rules have been applied exhaustively.

Variable Elimination eliminates a rightmost context variable X by generating a disjunction of $k+1$ conjunctions from each original conjunction, where each conjunction corresponds to a possible length of the word substituted for X.

Variable Elimination $P_{s,0} \vee \ldots \vee P_{s,k} \rightarrow \bigvee_{0 \leq i \leq k} \bigvee_{0 \leq j \leq i} P_{s,i}\{t_{s,j}/X\}$

if X is a variable with maximal variable prefix s, k is the maximal substitution length in P_s, and $t_{s,j}$ is a suffix of length j in some right-hand side of $P_{s,i}$.

A priori Variable Elimination can lead to a quadratic blowup of the disjunction at each step. However, as we will argue below, simplification eliminates all but linearly many distinct conjunctions. Note that X having the maximal variable prefix s implies that X is the rightmost variable in the left-hand sides of equations in the disjunction $P_{s,0} \vee \ldots \vee P_{s,k}$ and that X occurs nowhere else in the problem. Note also that we do not keep the bindings of the variables, as in general this results in an exponential number of solutions.

We observe that for an equation transformed by Bottom Decompose each side of the result is a prefix of the corresponding side of the original equation, and that the other simplification rules do not generate new equations. For Variable Elimination this is not true for the left-hand sides, as a word is substituted for the variable. However, simplification by Bottom Decompose removes this word, so Variable Elimination followed by eager simplification also preserves this prefix property, and hence it is preserved by the algorithm in general for simplified problems. Thus an equation in a simplified problem is completely determined by the initial equation it is derived from, its variable prefix and its substitution length. The last variable in the variable prefix determines the end of the left-hand side, and given the left-hand side the substitution length determines the length of the right-hand side.

Moreover, inner conjunctions are either eliminated completely by a failing rule, or they retain for each original equation of suitable variable prefix some corresponding equation in the conjunction. Since variable prefix and substitution length are fixed for each inner conjunction $P_{s,i}$, an inner conjunction also contains at most one equation derived from some initial equation; if there were two then they would be identical and thus merged by idempotency for $\wedge$. Thus nontrivial inner conjunctions in a simplified problem are also determined by their variable prefix and their substitution length, and the $O(n^2)$ conjunctions introduced by Variable Elimination lead to only $O(n)$ distinct conjunctions after simplification, where conjunctions are either removed by failing rules, or merged with equal conjunctions by the idempotency rule for $\vee$.

Finally we note that since the top-level conjunction partitions the equations with respect to the variable prefix of their rightmost variable, the descendants of an original equation are grouped inside one disjunction. Since each inner conjunction contains at most one descendant, each original equation can lead to only k equations in an intermediate problem, where k is the maximal substitution length in the problem that bounds the number of

disjuncts. This implies that the size of intermediate simplified problems is $O(n^2)$. Since the temporary blowup in Variable Elimination is also of size $O(n^2)$ all intermediate problems have size $O(n^2)$.

Theorem 22 *This algorithm decides solvability of SSMCM.*

Proof: By inspection we see that the rules preserve solvability.

For completeness we note that any equation without a variable at the end of its left-hand side can be simplified either by Delete or Substitution Length Clash 1 if one side is empty, or by Bottom Decompose otherwise. Restricted Distributivity will merge any two disjunctions with the same prefix. To any nontrivial simplified problem Variable Elimination is applicable. In particular, choosing k as the maximal substitution length of P_s ensures that a sufficiently long right-hand side exists. Hence a rule applies to any problem that is not in solved form.

For termination we use the lexicographic combination of the number of variables, which takes care of Variable Elimination steps, the number of conjuncts in the top conjunction, which takes care of Restricted Distributivity steps, and the size of the problem for the other simplification rules. □

To obtain a bound on time complexity we note that Restricted Distributivity only applies to at most two disjuncts with the same variable prefix, one preexisting and one created by the preceding application of Variable Elimination. The other simplification rules all strictly decrease the number of symbols and thus lead to a linear number of rule applications, so they normalize this subproblem of size $O(n^2)$ in $O(n^2)$ steps. We get an overall bound of $O(n^2)$ steps for exhaustively applying the simplification rules. There are $O(n)$ variables and each variable is removed in $O(n^2)$ steps, hence the derivation has at most length $O(n^3)$. Although a single rule application may need linear time for finding a subproblem to which it is applicable, with suitable algorithms the cost of this over a derivation that is at least of linear length in the size of the problem will be linear in the length of the derivation, and hence not exceed time $O(n^3)$. We thus obtain the following theorem:

Theorem 23 *SSMCM is solvable in time $O(n^3)$ and space $O(n^2)$.*

Given a derivation, a solution can easily be extracted from the derivation.

Example 24 *As an example we consider the problem* $XfY = ff \wedge XZ = g$ *with maximal substitution length* 1. *We obtain the following sequence of rule applications:*

$$\begin{aligned}
& XfY \approx ff \wedge XZ \approx g \\
\rightarrow_{\mathit{Variable\ Elimination\ Y}} & (Xf \approx ff \vee Xff \approx ff) \wedge XZ \approx g \\
\rightarrow^*_{\mathit{Bottom\ Decompose}} & (X \approx f \vee X \approx \epsilon) \wedge XZ \approx g \\
\rightarrow_{\mathit{Variable\ Elimination\ Z}} & (X \approx f \vee X \approx \epsilon) \wedge (X \approx g \vee Xg \approx g) \\
\rightarrow_{\mathit{Bottom\ Decompose}} & (X \approx f \vee X \approx \epsilon) \wedge (X \approx g \vee X \approx \epsilon) \\
\rightarrow^*_{\mathit{Restricted\ Distributivity}} & (X \approx f \wedge X \approx g) \vee (X \approx \epsilon \wedge X \approx \epsilon) \\
\rightarrow_{\mathit{Idempotency}} & (X \approx f \wedge X \approx g) \vee X \approx \epsilon \\
\rightarrow_{\mathit{Variable\ Elimination\ X}} & ((\epsilon \approx f \wedge \epsilon \approx g) \vee \epsilon \approx \epsilon) \vee (f \approx f \wedge f \approx g) \\
\rightarrow^* & \epsilon \approx \epsilon \\
\rightarrow & \top
\end{aligned}$$

8 Other complexity results

8.1 Shared-Linear Context Matching Is in P

NAME: Shared-Linear Context Matching (SLCM)
INSTANCE: A context matching problem instance P such that for each context variable all its occurrences are applied to the same term.
QUESTION: Is P solvable?

A context matching problem instance is *shared-linear* for each context variable all its occurrences are applied to the same term. This restriction has been introduced by Comon [5, 6], it is a special case of stratification. It implies that a fully shared graph representation of such a problem instance contains each context variable exactly once. We can apply the algorithm for the linear case, where we replace subterms by the corresponding sub-DAGs in the graph representation. This leads to the same time and space complexity, where n is taken to be the size of the graph.

8.2 Varity 2 Context Matching Is NP-complete

NAME: Varity 2 Context Matching (V2CM)
INSTANCE: A context matching problem instance P such that context and individual variables occur at most twice.
QUESTION: Is P solvable?

Varity 2 context matching (V2CM) is context matching restricted to the case where every context or individual variable may occur at most twice. The notion "varity of a term or formula" was introduced by Lynch and Morawska [19], it is the maximal number of occurrences of a variable.

Theorem 25 *V2CM is NP-complete.*

Proof: By reduction of positive 1-IN-3-SAT [13] to V2CM. Truth values are represented by a for false and $f(a)$ for true. The essential point is that n different individual variables can be set to the same truth value by a match equation

$$X_i(g(y_{i1}, ..., y_{in_i})) \approx h(g(a, ..., a), g(f(a), ..., f(a))).$$

We use such an equation to obtain n_i variables for each logical variable in the 1-IN-3-SAT instance, where n_i is the number of times it occurs in the instance. Clauses are then represented by

$$Z_j(g(y_{i_{j1}}, y_{i_{j2}}, y_{i_{j3}})) \approx h(g(f(a), a, a), g(a, f(a), a), g(a, a, f(a))),$$

analogously to [22]. Here we use a different individual variable for each occurrence of a logical variable. Thus each individual variable occurs twice, once in a clause of the first type and once in a clause of the second type. □

In fact, this uses only the subcase of V2CM where context variables are linear and individual variables occur at most twice. The same applies to the case where all individual variables have been replaced by context variables applied to some fixed constant. We also note that the g function symbols may be replaced by suitable iterations of a single function symbol of arity 2.

In the terminology of Hirata, Yamada and Harao [14], who didn't consider varity restrictions, the context matching problem constructed falls into UnaryPredMatching, where Unary restricts function variables to be unary (our contexts are always unary) and Pred forbids function variables below function variables. Hence this shows NP-completeness of UnaryPredMatching with tightened varity bounds. The proof of Hirata, Yamada and Harao [14] shows NP-completeness of UnaryPredMatching with varity 2 for function variables and unbounded varity for individual variables.

8.3 The k-context variable fragment is in P

Name: k-Context Variable Context Matching (k-CVCM)
Instance: A context matching problem instance P that contains at most k context variables.
Question: Is P solvable?

The case with at most k context variables and an unbounded number of individual variables is trivially in P, because the value substituted for each context variable must be chosen from the subcontexts of the right-hand sides. A subcontext is defined by two positions in the right hand side, one for the root of the subcontext and one for its hole below, which leads to at most $O(n^2)$ subcontexts.

After instantiating the context variables the remaining first-order matching problem is solvable in linear time, hence the k-context variable fragment is solvable in time $O(n^{2k})$.

Corollary 26 *The data complexity of any context matching problem is in P.*

Thus we cannot construct an NP-complete context matching problem with a fixed left-hand side.

9 Conclusion and Further Work

A borderline case that is a strengthening of stratification remains open: Consider all the variable prefixes of variables in the problem, ordered by the prefix ordering. For a general stratified context matching problem the Hasse diagram for the prefix ordering forms a tree. In other words, this restricted *linearly stratified* case is defined by the property that there are no two different variables immediately below a context variable (ignoring function symbols). It is open whether stratified context matching problems with a linear prefix ordering are in P or NP-complete, or maybe even equivalent to the graph isomorphism problem.

It remains to analyze context matching in terms of query and data complexity. We conjecture that deciding linear context matching has linear data complexity, as for a given linear left-hand side there is a tree automaton that accepts all matching right-hand sides.

From the point of view of applications it may be interesting to explore the case modulo associativity and/or commutativity. For example, it would be natural to replace the "." operator in the introductory example by an associative and/or commutative list constructor.

Parallelism constraints [12] are equivalent in power to context unification but perform better on certain linguistic problems. It could be interesting to consider a matching variant of this problem where the tree is given.

Acknowledgments

We thank Claude Kirchner whose suggestion initiated this work and K. Vikram for his valuable remarks, in particular for pointing out that Bottom Decompose is also applicable in the nonmonadic case.

References

[1] Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge University Press, Cambridge, UK, 1998.

[2] Peter Borovanský, Claude Kirchner, Hélène Kirchner, and Christophe Ringeissen. Rewriting with strategies in ELAN: a functional semantics. *Int. Journal of Foundations of Computer Science*, 1999.

[3] James Clark, editor. *XSL Transformation (XSLT) Version 1.0.* W3C, 16 November 1999. `http://www.w3.org/TR/1999/REC-xslt-19991116`.

[4] James Clark and Steve DeRose, editors. *XML Path Language (XPath) Version 1.0.* W3C, 16 November 1999. `http://www.w3.org/TR/1999/REC-xpath-19991116`.

[5] Hubert Comon. Completion of rewrite systems with membership constraints. Part I: Deduction rules. *Journal of Symbolic Computation*, 25(4):397–419, 1998.

[6] Hubert Comon. Completion of rewrite systems with membership constraints. Part II: Constraint solving. *Journal of Symbolic Computation*, 25(4):421–453, 1998.

[7] Hubert Comon and Yan Jurski. Higher-order matching and tree automata. In *Proc. Conf. on Computer Science Logic (CSL-97)*, LNCS 1414, pages 157–176, Aarhus, 1997. Springer.

[8] Régis Curien, Zhenyu Qian, and Hui Shi. Efficient second-order matching. In *Proc. 7th Int. Conf. on Rewriting Techniques and Applications*, LNCS 1103, pages 317–331, New Brunswick, NJ, USA, 1996. Springer.

[9] Philippe de Groote. Linear higher-order matching is NP-complete. In *Proc. 11th Int. Conf. on Rewriting Techniques and Applications*, LNCS 1833, pages 127–140, Norwich, UK, 2000. Springer.

[10] Dan Dougherty and ToMasz Wierzbicki. A decidable variant of higher-order matching. In *Proc. 13th Int. Conf. on Rewriting Techniques and Applications (RTA)*, LNCS 2378, page 340ff, Copenhagen, Denmark, 2002. Springer.

[11] Gilles Dowek. Higher-order unification and matching. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 16, pages 1009–1062. North-Holland, 2001.

[12] Katrin Erk and Joachim Niehren. Parallelism constraints. In *Proc. 11th Int. Conf. on Rewriting Techniques and Applications*, LNCS 1833, pages 110–126, 2000.

[13] M. Garey and D. Johnson. *Computers and Intractability.* W.H. Freeman, 1979.

[14] Kouichi Hirata, Keizo Yamada, and Masateru Harao. Tractable and intractable second-order matching problems. In *Proc. 5th Ann. Int. Computing and Combinatorics Conference (COCOON'99)*, LNCS 1627, pages 432–441. Springer, 1999.

[15] Haruo Hosoya and Benjamin Pierce. Regular expression pattern matching. In *Proc. 28th Ann. Symp. on Principles of Programming Languages (POPL)*, 2001.

[16] Gérard Huet and Bernard Lang. Proving and applying program transformations expressed with second-order patterns. *Acta Informatica*, 11:31–55, 1978.

[17] Jordi Levy. Linear second-order unification. In *Proc. 7th Int. Conf. on Rewriting Techniques and Applications*, LNCS 1103, pages 332–346, 1996.

[18] Ralph Loader. Higher order β matching is undecidable. *Logic Journal of the IGPL*, 11(1):51–68, 2003.

[19] Christopher Lynch and Barbara Morawska. Approximating E-unification. `http://www.clarkson.edu/~clynch/publications.html`, 2001.

[20] Manfred Schmidt-Schauß. Stratified context unification is in PSPACE. In *CSL 2001*, LNCS 2142, pages 498–512. Springer, 2001.

[21] Manfred Schmidt-Schauß. A decision algorithm for stratified context unification. *Journal of Logic and Computation*, 12(6):929–953, 2002.

[22] Manfred Schmidt-Schauß and Klaus U. Schulz. On the exponent of periodicity of minimal solutions of context equations. In *Proc. 9th Int. Conf. on Rewriting Techniques and Applications*, LNCS 1379, pages 61–75, Tsukuba, Japan, 1998. Springer.

[23] Manfred Schmidt-Schauß and Klaus U. Schulz. Solvability of context equations with two context variables is decidable. *Journal of Symbolic Computation*, 33(1):77–122, 2002.

[24] Alexander Schrijver. *Theory of Linear and Integer Programming.* Wiley, Chichester, 1986.

[25] Jürgen Stuber. Deriving theory superposition calculi from convergent term rewriting systems. In *Proc. 11th Int. Conf. on Rewriting Techniques and Applications (RTA 2000)*, volume 1833 of *LNCS*, pages 229–245, Norwich, UK, 2000. Springer.

[26] Jürgen Stuber. A model-based completeness proof of Extended Narrowing And Resolution. In *Proc. 1st Int. Joint Conf. on Automated Reasoning (IJCAR-2001)*, LNCS 2083, pages 195–210, Siena, Italy, 2001.

[27] ToMasz Wierzbicki. Complexity of the higher-order matching. In *Proc. 16th Int. Conf. on Automated Deduction (CADE-16)*, LNAI 1632, pages 82–96, Trento, Italy, 1999. Springer.

Contents

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399