On the complexity of linear and stratified context matching problems

Manfred Schmidt-Schauß* Jürgen Stuber†

December 20, 2001

Abstract

We investigate the complexity landscape of context matching with respect to the number of occurrences of variables (i.e. linearity vs. varity 2) and various restrictions of stratification. We show that stratified context matching (SCM) and varity 2 context matching are NP-complete, but that stratified simultaneous monadic context matching (SSMCM) is in P. SSMCM is equivalent to stratified simultaneous word matching (SSWM). We also show that the linear and the Comon-restricted case are in P and of time complexity $O(n^3)$. We give an algorithm for context matching and discuss how the performance of the general case can be improved through the use of information derived from polynomial approximations of the problem.

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, context variables can occur as monadic operators anywhere inside a term. Context matching 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

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

*Institut für 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

†INRIA LORIA, 615 Rue Jardin Botanique, F-54600 Villers-les-Nancy, France. Tel: (+33)383-59 30 36, Fax: (+33)383-27 83 19, Email: stuber@loria.fr

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 $X(book(Y(author(a_1)))) = db \wedge X(book(Y(title(x)))) = db$ 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 formalism such as regular expression matching [14] 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 to both separate the list used for emulating variable arity from other function symbols and for enforcing 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, 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 builtin theory [24, 25] 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:

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

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

Comon-restricted context matching (CRCM): For each context variable all terms occurring immediately below that context variable are identical.

Stratified context matching (SCM): For each context 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).

We show that LCM, CRCM and SSMCM are in P, while V2CM and SCM are NP-complete. It is easily seen that context matching is in NP, hence its restrictions V2CM, SCM 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 [21]. Context matching is a restricted form of linear higher-order matching, which was shown to be in NP and hence NP-complete by de Groote [9]. Here linear means that only solutions where all functions are linear, i.e. contain each of their bound variable exactly once, are considered. 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. An algorithm for general second-order matching is given by Huet and Lang [15]. 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 [26]. For orders above four it is still open whether higher-order matching is decidable [10]. Recently Loader has shown that higher-order beta-matching is undecidable, but this doesn't imply anything about decidability of the beta-eta case [17].

Hirata, Yamada and Harao [13] 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 its decidability [21, 20, 19]. Other decidable fragments are the varity 2 fragment [16][1] and the two-context-variable fragment [22], 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 are constructed from the variables and function symbols in the usual way, where context variables are considered as monadic function symbols. A context equation is a pair of terms written $s \approx t$.

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 $C[]$. 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 of a subterm by $C[t/p]$.

A *substitution* is a mapping from individual variables to terms and from context variables to contexts, such that almost all individual variables are mapped to themselves, and almost all context variables are mapped to themselves applied to the hole. We write $\sigma(x)$ or $\sigma(X)$ for the term or context an individual or context variable maps to, respectively.

[1] Note that the proof of decidability for the stratified case is flawed in this paper.

Substitutions are extended to terms as follows:

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

3 Problem definition

We first briefly discuss different forms of equations and show that they are equivalent.

For instance, we may consider equations between (context) terms or between contexts themselves, where the holes must match. To get from a term equation to a context equation we may use a binary function symbol to put the hole into a side branch, i.e. reduce $s \approx t$ to $h(\Box, s) \approx h(\Box, t)$. In the other direction we may replace the hole by a special constant that doesn't occur elsewhere. This assumes that such a binary function symbol or constant exists.

We may also consider *multi-contexts*, which are terms with an arbitrary number of occurrences of the hole. 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 indicate the number of holes by a subscript, i.e. $C[\ldots_i, s, \ldots_j]$. Note that we do not allow variables that stand for multi-contexts.

If we have an equation between multi-contexts, e.g. $C[f(s_1[\Box], s_2[\Box])] \approx C[g(t_1[\Box], t_2[\Box])]$, then the first (resp. second) hole on the left-hand side must match the first (resp. second) hole on the right-hand side, which forces a match between the function symbols f and g at the longest common prefixes of the positions of the holes. Thus either $f \neq g$ and there is no solution, or we can split this equation into the equations $s[] \approx t[]$, $s_1[] \approx t_1[]$ and $s_2[] \approx t_2[]$, reducing the multi-context equation to a conjunction of context equations. This elimination of multi-contexts can be done in linear space and time.

If at least one symbol of arity at least two is available, then a conjunction of equations can be coded as a single equation. For example, if a symbol h with arity 2 is available, we may replace

$$s_1 \approx t_1 \wedge \ldots \wedge s_n \approx t_n$$

by

$$h(\ldots h(s_1, s_2) \ldots, s_n) \approx h(\ldots h(t_1, t_2) \ldots, t_n).$$

By these equivalences we can define a *context equation* as any equation between context terms, contexts or multi-contexts, and a *context matching problem* P as either a single context equation or a conjunction of context equations (we will be more precise when the function symbols assumed above are not available). A substitution σ is a *solution* of a context equation $s \approx t$ if $s\sigma = t\sigma$. A solution of P is a substitution that solves all equations in P.

A context matching problem is called *linear* if each variable occurs at most once. A problem is called *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 that occurrence. A context matching problem is called *stratified* if for each individual or context variable all occurrences of that variable in the problem have the same variable prefix.

4 Linear Context Matching is in P

We can solve linear context matching problems by dynamic programming. We build a table which for every pair $\langle s, t\rangle$ 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. The table is built recursively from the bottom up:

- If s is an individual variable then the answer is yes.
- If $s = f(s_1, \ldots, s_m)$ and $t = g(t_1, \ldots, t_n)$ then the answer is yes if and only if $f = g$ and s_i matches t_i for $1 \leq i \leq m = n$.
- 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.

5 Context matching with Comon's restriction is in P

A context matching problem obeys *Comon's restriction* [5, 6] if for each context variable all its occurrences are applied to the same term. This implies that a fully shared graph representation of such a term contains each context variable exactly once. Hence we can apply the algorithm for the linear case and obtain the same time and space complexity.

6 Varity 2 context matching is NP-complete

Varity 2 context matching (V2CM) is context matching restricted to the case where every variable may occur at most twice. The notion "varity of a term or formula" was introduced by Lynch and Morawska [18], it is the maximal number of occurrences of a variable.

Theorem 1 *V2CM is NP-complete.*

Proof: By reduction of V2CM to positive 1-IN-3-SAT [12]. 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

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

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

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

analogously to [21]. 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 a single function symbol of arity 2.

In the terminology of Hirata, Yamada and Harao [13], 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 [13] shows NP-completeness of UnaryPredMatching with varity 2 for function variables and unbounded varity for individual variables.

7 The k-context variable fragment is in P

Now we consider the case with at most k context variables and an unbounded number of individual variables. The value substituted for each context variable must be chosen from the right-hand sides, which contain at most $O(n^2)$ contexts. 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 $O(n^{2k})$.

The proof of NP-completeness for a single second-order variable by Hirata, Yamada and Harao [13] is not applicable here, because it relies on a variable with a large number of arguments and the possibility to permute the arguments by the function instantiated for it.

A consequence of the k-context variable fragment being in P is that we cannot construct an NP-complete problem with a fixed left-hand side, i.e. the data complexity is always in P.

8 Simultaneous stratified monadic context matching is in P

In this case conjunctions cannot be coded in a single equation, hence we make explicit that we allow them by defining a *simultaneous context matching problem* P as a conjunction of equations whose right-hand sides are ground terms. We will show that simultaneous stratified monadic context matching (SSMCM) is in P.

The crucial property that leads to a polynomial algorithm in this case is that there are no branchings in the terms. For any equation $w_1 X_1 \dots w_n X_n[] \approx w[]$ we know that a solution σ must satisfy $|(X_1 \dots X_n)\sigma| = |w| - |w_1 \dots w_n|$, hence we call $|w| - |w_1 \dots w_n|$ 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 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. To simplify the notation we will consider terms as words in the rest of this section.

We give a transformation algorithm to solve SSMCM. We will first discuss the form of the problem at intermediate steps, which will give us a polynomial bound on the size of these problems. We will then see that this form is invariant under our transformation rules. Finally we show that each transformation rule makes enough progress towards a solved form such that a derivation has at most polynomial length. Together this will show that SSMCM is in P.

At each step the problem P is a conjunction of disjunctions of conjunctions of match equations. More precisely, it has the form

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

where $\{s_1, \ldots, s_n\}$ is the set of all variable prefixes for rightmost variables in P, and P_{s_i} is a disjunction of conjunctions of equations whose rightmost variable on the left-hand side has the prefix s_i. The number of prefixes is bounded by the number of 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 we require that P_s has the form

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

where $P_{s,i}$ is a conjunction of equations whose substitution length is i. Note that this implies that for a given unifier σ only the disjunct with matching substitution length is solvable. For the bound k we can use the largest substitution length in the initial problem, which is bounded by the size of the problem. Finally, each initial equation leads to at most one equation in every disjunct $P_{s,i}$ whose sides are a prefix of the original equation. In total we obtain that the size of an intermediate problem is at most $O(n^3)$. In fact, since the top-level conjunction partitions the equations w.r.t. the variable prefix of their rightmost variable, the descendants of an original equation are grouped inside one disjunction, and since any inner conjunction can contain at most one of them each original equation can lead to only k equations in an intermediate problem. This implies the stronger estimation that the size of intermediate problems is $O(n^2)$.

It remains to give the transformation algorithm and show that its derivations are of polynomial length. First we define a transformation rule analogous to Decompose that is applied at the bottom of an equation (the standard rule is applied at the top), as follows:

Bottom Decompose $\quad sf \approx tg \rightarrow s \approx t$

if $f = g$; otherwise fail,

where f and g are function symbols. We assume that Bottom Decompose is applied eagerly. This leaves only variables at the end of left-hand sides.

The main step that eliminates a variable at the end is as follows:

1. Pick a disjunction for a maximal prefix. At the right end of equations it contains all occurrences of a context variable X in the problem.

2. Generate a disjunction of conjunctions for all the possible lengths of $\sigma(X)$, where the number of disjuncts is bounded by the length of the right-hand side.

3. For each disjunct substitute for X the word determined by the chosen length, then apply Bottom Decompose.

4. Delete the equation binding X. This is necessary because in general there are exponentially many solutions, which can not all be recorded if we want to stay polynomial.

5. Merge the disjunction that is obtained with any existing subproblem P_s for the same shortened variable prefix, removing conjunctions which are inconsistent w.r.t. length. Essentially this pairs up the disjuncts according to their substitution lengths.

   Since equations derived from the same original equations have their left-hand side end at the same context variable, their left-hand sides are identical. For equal substitution length their right-hand sides must be equal as well. These identical equations are merged by idempotency, which ensures that each original equation leads to at most one equation in an inner conjunct.

To formalize this we break it down into several rules. We introduce a new atomic constraint $|X| = d$ that is satisfied if the context substituted for X has length d.

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

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

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

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.

Choice $\quad P_0 \vee \ldots \vee P_k \to \bigvee_i \bigvee_j P_{i+j} \wedge |X| = j$

if X is the rightmost variable of the disjunction $P_0 \vee \ldots \vee P_k$ and occurs nowhere else, and k is the maximal substitution length in P.

Bind $\quad P \wedge |X| \approx d \to P \wedge X \approx t$

if X is the bottom variable of the conjunction P and t is a context of length d at the bottom of some right-hand side in P.

Eliminate $\quad P \wedge X \approx t \to P\{t/X\}$

We assume rules that use idempotency and the laws for true and false to eagerly simplify the problem. Before and after Choice the other rules are applied eagerly to simplify the problem. Rules are iterated until $\top$ or $\bot$ is obtained.

Note that after simplification the inner disjunctions in the Choice rule will consist of conjunctions with substitution length i. After elimination of X all the equations in such a subproblem that are derived from the same original equation become identical. Moreover, a conjunction in the subproblem is either unsatisfiable or trivial or contains an equation for each corresponding original equation. A failure will remove the entire conjunction, and the only rules that can render an equation trivial are Bottom Decompose and Eliminate, and by stratification this can only happen simultaneously when the last variable is removed. Hence after simplification the conjunctions in a subproblem with the same substitution length are identical and can be merged by idempotency.

Proposition 2 *This terminates.*

Proof: As a measure consider the lexicographic combination of the number of variables, the number of substitution length constraints and the size of the problem. All the rules except Choice decrease this measure. Choice enables the application of Bind and in turn Eliminate, who remove the chosen variable and decrease the measure before the next application of Choice. □

Proposition 3 *Each rule preserves solvability.*

Proof: By inspection. □

Proposition 4 *If the problem is not of the form $\top$ or $\bot$ then a rule is applicable.*

Proof: If equations contain function symbols at the bottom they can be removed by Bottom Decompose. Substitution length constraints can be removed by Bind, where a suitable right-hand side exists by stratification. It remains to consider the case of equations with only variables at the bottom. We may choose a variable with a maximal prefix. This variable will occur in a single disjunction, hence we can apply choice. □

There are $O(n)$ variables and each variable is removed in $O(n^2)$ steps, hence the derivation has at most length $O(n^3)$. Space stays $O(n^2)$, since the intermediate blowup during elimination of a variable is done only for one variable at a time and also of the order $O(n^2)$. We obtain the following theorem:

Theorem 5 *SSMCM is solvable in time $O(n^3)$ and space $O(n^2)$.*

To compute a matching substitution we can keep bindings and delete all but one of a set of conflicting bindings upon merging.

9 Stratified Context Matching is NP-complete

The technique used in the previous section to solve the monadic case doesn't apply to the general case. Lengths in the monadic case correspond to occurrences (sequences of natural numbers) in the general case. The crucial difference is that the composition of lengths is addition, which is commutative, while the composition of sequences is not commutative. In particular, it is possible to devise two equations with each two context variables on top of each other in the left-hand side, such that in one equation two different matchings lead to the same combined position in the right-hand side, while in the second equation the

same matchings lead to different positions. This is the crucial ingredient in the encoding that we present below to show NP-hardness.

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

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

that contains a dummy equation $s_0 \approx t_0$ and one equation $s_i \approx t_i$ for each clause c_i. The dummy equation restricts matchings so that they correspond to boolean valuations, and clause equation i tests whether such a valuation satisfies clause i. 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 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 on top of each other, which represent the truth values. In the case of an active layer these matches select different subterms on the right-hand side for matching in lower layers, which allows 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 a string containing $*$ in the i-th position when there is no literal containing x_i, 0 when there is a literal containing x_i that is false under the chosen assignment, and 1 when there is a literal containing x_i 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.

To formalize this we recursively define

$$s_{ij} = f(X_j(g(g(x_{ij}, Y_j(s_{i,j+1})), y_{ij}))) \qquad t_{i,\alpha} = f(g(g(g(c, t_{i,\alpha 0}), g(t_{i,\alpha 1}, c)), c)) \qquad (1)$$

$$s_{ij} = f(X_j(g(g(x_{ij}, Y_j(s_{i,j+1})), y_{ij}))) \qquad t_{i,\alpha} = f(g(g(g(c, t_{i,\alpha 1}), g(t_{i,\alpha 0}, c)), c)) \qquad (2)$$

$$s_{ij} = f(X_j(Y_j(s_{i,j+1}))) \qquad t_{i,\alpha} = f(g(t_{i,\alpha *}, c)) \qquad (3)$$

where $0 \leq i \leq m$, $1 \leq j = \mathrm{len}(\alpha) + 1 \leq n$ and we choose the active layer (1) for a positive occurrence of the j-th variable in the i-th clause, the active layer (2) for a negative occurrence and the passive layer (3) for no occurrence. For $i = 0$ we always choose (3). In (1) and (2) a match of $s_{i,j+1}$ against the first subterm of the form $t_{i,\ldots}$ corresponds to assigning false to the variable x_i in the 3SAT problem, while matching against the second such subterm corresponds to assigning true to x_i. 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 $j = \mathrm{len}(\alpha)+1$. 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 passive layer. There are two possible matches in layer j:

$$X_j = \Box \qquad Y_j = g(\Box, c) \tag{4}$$

$$X_j = g(\Box, c) \qquad Y_j = \Box \tag{5}$$

These are the only possibilities to match a passive layer (3). Now let us look what happens in an active layer of the form (1): In case (4) the next lower layer $s_{i,j+1}$ is matched against $t_{i,\alpha 1}$, while in case (5) it is matched against $t_{i,\alpha 0}$. For a layer of the form (2) it is vice-versa. Hence a match of the form (4) corresponds to assigning true to the logical variable x_j, and a match of the form (5) 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 three points, since each clause contains 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 linear in the size of the 3SAT instance.

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

Theorem 6 *SCM is NP-complete.*

10 A Transformation Algorithm for Context Matching

Even though context matching is NP-hard in general it is nevertheless interesting to develop practical algorithms for it. 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.

We give rules for context matching problems consisting of a conjunction of equations between multi-context terms whose right-hand side is ground. We first give don't-care nondeterministic rules that may be applied eagerly:

Decompose $\quad f(s_1, \ldots, s_m) \approx g(t_1, \ldots, t_n) \to s_1 \approx t_1 \wedge \ldots \wedge s_n \approx t_n$

if $f = g$ and $m = n$; otherwise fail.

Term Merge $\quad x \approx t_1 \wedge x \approx t_2 \to x \approx t_1$

if $t_1 = t_2$; otherwise fail.

Context Merge $\quad X[] \approx t_1[] \wedge X[] \approx t_2[] \to X[] \approx t_1[]$

if $t_1[] = t_2[]$; otherwise fail.

Context Eliminate $\quad X[] \approx t_1[] \wedge X(s) \approx t_2 \rightarrow X[] \approx t_1[] \wedge s \approx t_3$

if $t_1[t_3] = t_2$; otherwise fail.

The resulting equation $s \approx t_3$ may be a term or context equation. Note that t_3 can be computed in linear time.

Multi-context Decompose

$s[\ldots_i, f(s_1[], \ldots, s_m[]), \ldots_k] \approx t[\ldots_i, g(t_1[], \ldots, t_n[])], \ldots_l]$
$\rightarrow s[\ldots_i, \Box, \ldots_k] \approx t[\ldots_i, \Box, \ldots_k] \wedge s_1[] \approx t_1[] \wedge \ldots \wedge s_m[] \approx t_m[]$

if $k = l$, $f = g$ and $m = n \geq 2$; fail if $k \neq l$ or $(m, n \geq 2$ and $(f \neq g$ or $m \neq n))$.

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

Variable Split $\quad X(s) \approx t \rightarrow X[] \approx t_1[] \wedge s \approx t_2$

if $s \neq \Box$ and $t_1[t_2] = t$.

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

Lemma 7 *The don't-care nondeterministic rules preserve the set of solutions.*

Proof: By inspection. The only nontrivial case is Multi-Context Decompose. Note that a solution cannot introduce or delete holes, hence it must map the i-th hole on the left-hand side to the i-th hole on the right-hand side. This correspondence fixes the paths from the root to the holes, and in particular induces a correspondence between function symbols in positions where several paths to holes branch (this cannot occur in the substitution since context variables stand for single contexts). The rule uses this correspondence to decompose the term at an innermost branch point. □

Lemma 8 *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: By inspection. □

Lemma 9 *If a context matching problem is not in solved form then one of the rules is applicable.*

Proof: If there are multi-contexts we can apply Multi-Context Decompose. There must exist branch positions with function symbols f and g who have each at least two arguments that are single contexts. After exhaustive application all contexts are single contexts.

If there is an equation with a function symbol at the root of the left-hand side then we can apply Decompose. After exhaustive application 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 Eliminate or Context Eliminate can be applied, respectively.

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

Lemma 10 *The application of rules terminates.*

Proof: This follows from the termination proof for an extended set of rules in the next section. □

Theorem 11 *The algorithm that applies the don't-care rules above eagerly and backtracks over Context 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, in particular it may produce exponentially many solutions. If we are only interested in solvability and want to avoid exponential behavior we may remove equations $x \approx t$ and $X[] \approx t[]$ 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, 15]. 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 implicitly bound variables.

In the next section we will discuss cases where exponential behavior can be avoided by polynomially calculating correspondences between positions.

11 Avoiding Search

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

11.1 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 P if p_s and p_t correspond for all solutions σ of P. 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 $s_i \approx t_i$ 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 $$s[s'/p_s] \approx t[t'/p_t] \to s[\Box/p_s] \approx t[\Box/p_t] \wedge s' \approx t'$$

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 12 *The rule Split preserves the set of solutions.*

Proof: By definition of corresponding position. □

Proposition 13 *The application of all the rules terminates.*

Proof: As a measure we may take the sum of the number of function symbols and variables in the problem and the number of argument positions of function symbols and variables not filled by holes. 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 14 *If for a class of context matching problems C 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 $O(k^{n+1})$ and C is in P.*

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

11.2 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 $\overline{s} \approx t$. 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 15 *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 $f(\ldots)$ 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.

11.3 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 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 [12], while it is possible to decide in polynomial time whether a solution in the integers exists [23]. 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.

12 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. It is open whether stratified context matching problems with a linear prefix ordering are in P or NP-complete. 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). Note that this generalizes SSMCM. We conjecture that this case is in P.

Parallelism constraints [11] 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.

It is interesting to note that many of the rules in this paper are specified using context notation on the meta-level, which in a formal setting may need context matching. It would be nice if this observation allowed a solver for context matching to be bootstrapped.

Acknowledgments

We thank Claude Kirchner whose suggestion initiated this work and Eric Domenjoud for answering a question on integer programming.

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] 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.

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

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

[13] 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.

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

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

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

[17] Ralph Loader. Higher order β matching is undecidable. `http://homepages.ihug.co.nz/~suckfish/match/beta.ps`, October 2001.

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

[19] Manfred Schmidt-Schauß. A decision algorithm for stratified context unification. Frank-Report 12, Fachbereich Informatik, J.W. Goethe-Universitat Frankfurt, Frankfurt, Germany, 1999. Accepted for publication in the Journal of Logic and Computation.

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

[21] 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.

[22] Manfred Schmidt-Schauß and Klaus U. Schulz. Solvability of context equations with two context variables is decidable. Accepted for publication in the Journal of Symbolic Computation, 2001.

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

[24] 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.

[25] 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.

[26] 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.

A An example for solving SSWM

Example 16 *To keep the example concise we write the monadic terms as words, without parentheses and omitting most of the holes at the bottom of every term.*

$$X_1gX_2gX_3ggX_4 = ggggg \wedge X_1ggX_2gX_3gX_4 = ggggg \tag{6}$$

$$\rightarrow_{Choice\ X_4} (X_1gX_2gX_3ggX_4 = ggggg \wedge X_1ggX_2gX_3gX_4 = ggggg \wedge X_4[] = [])$$
$$\vee\ (X_1gX_2gX_3ggX_4 = ggggg \wedge X_1ggX_2gX_3gX_4 = ggggg \wedge X_4 = g) \tag{7}$$

$$\rightarrow_{Eliminate\ X_4} (X_1gX_2gX_3gg = ggggg \wedge X_1ggX_2gX_3g = ggggg)$$
$$\vee\ (X_1gX_2gX_3ggg = ggggg \wedge X_1ggX_2gX_3gg = ggggg) \quad (8)$$
$$\rightarrow_{Bottom\ Decompose} (X_1gX_2gX_3 = ggg \wedge X_1ggX_2gX_3 = gggg)$$
$$\vee\ (X_1gX_2gX_3 = gg \wedge X_1ggX_2gX_3 = ggg) \quad (9)$$
$$\rightarrow_{Choice\ X_3} ((X_1gX_2gX_3 = ggg \wedge X_1ggX_2gX_3 = gggg)$$
$$\vee\ (X_1gX_2gX_3 = gg \wedge X_1ggX_2gX_3 = ggg)) \wedge (X_3[] = [] \vee X_3 = g) \quad (10)$$
$$\rightarrow_{Eliminate\ X_3} ((X_1gX_2g = ggg \wedge X_1ggX_2g = gggg)$$
$$\vee\ (X_1gX_2g = gg \wedge X_1ggX_2g = ggg))$$
$$\vee\ ((X_1gX_2gg = ggg \wedge X_1ggX_2gg = gggg)$$
$$\vee\ (X_1gX_2gg = gg \wedge X_1ggX_2gg = ggg)) \quad (11)$$
$$\rightarrow_{Bottom\ Decompose} (X_1gX_2 = gg \wedge X_1ggX_2 = ggg)$$
$$\vee\ (X_1gX_2 = g \wedge X_1ggX_2 = gg)$$
$$\vee\ (X_1gX_2 = g \wedge X_1ggX_2 = gg)$$
$$\vee\ (X_1gX_2[] = [] \wedge X_1ggX_2 = g) \quad (12)$$
$$\rightarrow_{Substitution\ Length\ Clash\ 1} (X_1gX_2 = gg \wedge X_1ggX_2 = ggg)$$
$$\vee\ (X_1gX_2 = g \wedge X_1ggX_2 = gg) \quad (13)$$
$$\rightarrow_{Choice\ X_2} ((X_1gX_2 = gg \wedge X_1ggX_2 = ggg)$$
$$\vee\ (X_1gX_2 = g \wedge X_1ggX_2 = gg)) \wedge (X_2[] = [] \vee X_2 = g) \quad (14)$$
$$\rightarrow_{Eliminate\ X_2} ((X_1g = gg \wedge X_1gg = ggg)$$
$$\vee\ (X_1g = g \wedge X_1gg = gg))$$
$$\vee\ ((X_1gg = gg \wedge X_1ggg = ggg)$$
$$\vee\ (X_1gg = g \wedge X_1ggg = gg)) \quad (15)$$
$$\rightarrow_{Bottom\ Decompose} ((X_1 = g \wedge X_1 = g)$$
$$\vee\ (X_1[] = [] \wedge X_1[] = []))$$
$$\vee\ ((X_1[] = [] \wedge X_1[] = [])$$
$$\vee\ \bot) \quad (16)$$
$$\rightarrow X_1 = g \vee X_1[] = [] \quad (17)$$
$$\rightarrow \top \quad (18)$$

The example uses just one letter. If the alphabet is larger, there will be more clashes, so the problem will become easier.