Congruence Closure and Syntactic Unification* (Abstract)

Leo Bachmair and Ashish Tiwari

Department of Computer Science, SUNY at Stony Brook,
Stony Brook, NY 11794, U.S.A
{leo,astiwari}@cs.sunysb.edu

1 Introduction

Kanellakis and Revesz [3] clarified the directional duality between congruence closure and unification closure by showing suitable reductions from instances of one problem to instances of the other. We outline here a more direct relationship between computation of unifiers and construction of congruence closures by presenting a set of abstract transformation rules for syntactic unification plus congruence closure. We believe that this approach is also useful for practical purposes. The motivation for combining congruence closure construction with syntactic unification arises from several observations. First, an idempotent substitution σ can be characterized via the ground equational theory induced by the set of equations E_σ obtained by interpreting the substitution assignment as mathematical equality (Proposition 1), and provability in a ground equational theory can be efficiently decided through the use of congruence closure [6]. Secondly, for efficiency reasons, both syntactic unification and congruence closure algorithms use a dag-based term representation. Furthermore, efficient algorithms for syntactic unification maintain a congruence relation on the vertices of the term dag using techniques for manipulating dags that are also common in congruence closure algorithms.

We present a set of abstract transformation rules for syntactic unification that are similar to the standard rules except that we add a rule for explicit congruence closure computation. A detailed formulation of transformation rules for congruence closure has been presented in [2]. These rules are shown to suitably capture manipulation of term dags done in graph based congruence closure algorithms. We briefly discuss some of the known algorithms for syntactic unification in this framework and show where and how the congruence closure computation steps are used therein.

2 Substitutions as Congruences

We assume that the signature Σ is a finite set of function symbols and constants, and $\mathcal{V}$ is a disjoint set of variable symbols. The set of all terms over the signature

* The research described in this paper was supported in part by the National Science Foundation under grant CCR-9902031.

Σ is denoted by $\mathcal{T}(\Sigma, \mathcal{V})$. A *substitution* is a mapping from variables to terms such that $x\sigma = x$ for all but finitely many variables x.

An *equation* is a pair of terms, written as $s \approx t$. The *replacement relation* $\rightarrow_{E^g}$ induced by a set of equations E is defined by: $u[l] \rightarrow_{E^g} u[r]$ if, and only if, $l \approx r$ is in E. The relation $\leftrightarrow^*$ denotes the symmetric, reflexive and transitive closure of $\rightarrow$. Thus, $\leftrightarrow^*_{E^g}$ denotes the *congruence relation* induced by E.

We reason about a substitution σ by considering the congruence relation $\leftrightarrow^*_{E^g_\sigma}$ induced by the set of equations $E_\sigma = \{x\sigma \approx x : x \in \mathcal{D}om(\sigma)\}$.

Proposition 1. *For all terms $s, t \in \mathcal{T}(\Sigma, \mathcal{V})$, we have*
(a) $t \leftrightarrow^*_{E^g_\sigma} t\sigma$.
(b) If a substitution σ is idempotent, then $s\sigma = t\sigma$ if, and only if, $s \leftrightarrow^*_{E^g_\sigma} t$.
(c) If $[x_1 \mapsto t_1; \ldots; x_n \mapsto t_n]$ is a triangular form representation of an idempotent substitution σ, then the congruences induced by E_σ and $\cup_i E_{\sigma_i}$ are identical, where $\sigma_i = \{x_i \mapsto t_i\}$.

Proposition 1 gives a characterization of idempotent syntactic unifiers σ in terms of provability using E^g_σ, or equivalently using $\cup_i E_{\sigma_i}$, where $[\sigma_1; \sigma_2; \cdots; \sigma_n]$ is a triangular presentation of the substitution σ, for details see [10]. It follows that congruence closure provides an efficient decision procedure for testing if two terms are unifiable by a given substitution (that could be given in a triangular form). Lassez, Maher and Marriott [4] also point out a connection between substitution assignments and mathematical equality, and show that idempotent most general unifiers and solved-form equation sets are essentially the same objects.

The *variable dependency ordering* $\succ^S_\mathcal{V}$ induced by a set S of equations on the set $\mathcal{V}$ of variables is defined by: $x \succ^S_\mathcal{V} y$ if there exists an equation $t[x] \approx y$ in S. A finite set S of equations is said to be *substitution-feasible*[1] if (i) the right-hand sides of equations in S are all distinct variables and (ii) the variable dependency ordering $\succ^S_\mathcal{V}$ induced by S is well-founded. If S is a substitution-feasible set $\{t_i \approx x_i : 1 \leq i \leq n\}$ such that $x_j \not\succ^S_\mathcal{V} x_i$ whenever $i > j$, then the idempotent substitution σ represented by the triangular form $[x_1 \mapsto t_1; \ldots; x_n \mapsto t_n]$ is called the substitution *corresponding to* S. Given an idempotent substitution σ and any triangular form representation $\sigma_1\sigma_2 \ldots \sigma_n$, the sets E_σ and $\cup_i E_{\sigma_i}$ are substitution-feasible.

A substitution σ is called *more general on V* than θ, denoted by $\sigma \preceq^\mathcal{V} \theta$, if there exists a substitution σ' such that $x\sigma\sigma' = x\theta$ for all $x \in \mathcal{V}$. Two substitutions σ and θ are said to be *equivalent on V* if $\sigma \preceq^\mathcal{V} \theta$ and $\theta \preceq^\mathcal{V} \sigma$.

If σ is a unifier of s and t, then there exists an idempotent unifier of s and t that is more general than σ; hence, we will be concerned only with idempotent unifiers. Comparisons between *idempotent* substitutions can be characterized via congruences.

Theorem 1. *Let σ and θ be idempotent substitutions and V the set of variables in the domain or range of σ. Then, $\sigma \preceq^\mathcal{V} \theta$ if and only if $E_\sigma \subseteq \leftrightarrow^*_{E^g_\theta}$.*

[1] Also called *solved-form* in literature.

3 Abstract Rules for Syntactic Unification

We present a class of algorithms for syntactic unification using transformation rules that work on quadruples of the form $(K, E?, E, S)$, where K is a set of constant symbols and $E?, E$ and S are sets of equations over $\Sigma \cup \mathcal{V} \cup K$. If we are interested in unifying terms s and t then we apply transformation rules to an initial state $(\emptyset, \{c_1 \approx c_2\}, \{s \approx c_1, t \approx c_2\}, \emptyset)$, where the intent is that the equations in $E?$-component are to be unified, the set E represents the dag, K contains the names of vertices in the dag, and S stores the answer substitution.

Construction of congruence closure is a crucial component in almost all syntactic unification algorithms even though this fact is not explicitly mentioned always. We do not list the rules for congruence closure here, and refer the reader to [2] for details. For simplicity, we assume that we only create *simple D-rules* and C-rules during computation of congruence closure, although the rules remain essentially unchanged for arbitrary D-rules.

Cong Closure: $$\frac{(K, E?, E, S)}{(K', E?', E', S)}$$

if $(K, E \cup E?) \vdash_{CC} (K', E' \cup E?')$.

Decomposition: $$\frac{(K, E? \cup \{c \approx d\}, E, S)}{(K, E? \cup \{c_1 \approx d_1, \ldots, c_k \approx d_k\}, E, S)}$$

if $c, d \in K$ and $f(c_1, \ldots, c_k) \approx c$ and $f(d_1, \ldots, d_k) \approx d$ are equations in E.

Clash: $$\frac{(K \cup \{c, d\}, E? \cup \{c \approx d\}, E, S)}{\perp}$$

if there exist equations $f(c_1, \ldots, c_k) \approx c$ and $g(c_1, \ldots, c_l) \approx d$ in E with $f \neq g$.

Occur Check: $$\frac{(K, E?, E, S)}{\perp}$$

if there exists a constant $c \in K$, and a term $t \in \mathcal{T}(\Sigma \cup \mathcal{V} \cup K)$ containing c such that $t \leftrightarrow^*_{E^g} c$.

Var-Elimination: $$\frac{(K, E? \cup \{c \approx d\}, E, S)}{(K, E?, E \cup \{c \approx d\}, S \cup \{x \approx d\}}$$

if the congruence class of c modulo E^g contains only variables and constants, and x is a variable in this class that does not occur as a left-hand side term in S.

Additionally, we can (i) use equations in E to simplify terms occurring in $E?$; and, (ii) delete trivial equations $c \approx c$ that occur in $E?$. Rather than specifying these by separate transformation rules, we have included them in the congruence closure rule by having the $E?$-component participate in the congruence closure steps. Note also that we assume that the equations in $E?$ are treated symmetrically.

We say that a state ξ is a final state if either $\xi = \perp$, or $\xi = (K, \emptyset, E, S)$ such that clash and occur check are not applicable to the state ξ (congruence closure could still be applicable). It is fairly easy to establish that each transformation rule preserves all unifiers (using the characterization of unifiers in terms of congruences), and that starting from $(\emptyset, \{c_1 \approx c_2\}, \{s \approx c_1, t \approx c_2\}, \emptyset)$, we reach a final state $\perp$ if and only if s and t are not unifiable.

4 Specific Unification Algorithms

A cautious reader might note that no occur check is performed in the variable elimination rule. Occur check is presented as an independent transition rule. The fact that occur check can be delayed until the very end is a feature of the *unification-closure* approach as described by Huet [1].

Another feature of the abstract transformation rules presented above is that no explicit substitutions are ever generated and hence, never applied either. The substitution generated during the the variable elimination rule $c \approx d$ is in the form of term dag nodes (constants c and d represent nodes of the term dag). We add this equation to the E-component so that the rule $c \approx d$ can be used to simplify (collapse) other equations in E (as part of the congruence closure transition rule). In the language of term dags, this can be seen to performing *explicit dag-substitutions*, as described in [1, 9]. This clearly brings out the implicit use of the congruence closure rule in the quadratic-time algorithm for unification as proposed by Corbin and Bidoit [1, 9]. In fact, this particular algorithm can be readily described using the transformation rules presented here, without delving into the details of term dags and dag-substitutions as has been done in [9].

The naive worst-case exponential recursive descent algorithms for syntactic unification differ from most of the other algorithms by (i) using explicit term representation with no sharing and (ii) not using congruence closure to merge certain terms to maintain an equivalence on the set of all terms. Repeated calls to unify the same pair of terms leads to repeated computation.

We have also assumed that the unifier S is returned using pointers to the term dag presented in E. Details of extracting a substitution-feasible set from the S-component using E are straight forward.

If S is a substitution-feasible set, then the congruence classes induced by S are finite and contain only variables and terms with the same top symbol. The class of linear and almost-linear time unification algorithms use the special nature of congruences induced by substitution-feasible sets in order to (i) speculatively perform deductions arising out of the congruence closure deduction rule, and (ii) apply decomposition in a way that the collapse rule for congruence closure is never applicable. Thus, we can modify decomposition as follows.

$$\textbf{Spec Decompose}: \quad \frac{(K, E? \cup \{c \approx d\}, E, S)}{(K, E? \cup \{c_1 \approx d_1, \ldots, c_k \approx d_k\}, E \cup \{c \approx d\}, S)}$$

if $c, d \in K$ and $f(c_1, \ldots, c_k) \approx c$ and $f(d_1, \ldots, d_k) \approx d$ are equations in E.

This suggests that the distinction between the $E?$- and E-components is not fundamental, and hence we can merge these two components into one. All transition rules remain the same, except decomposition and clash, where the condition $c \approx d \in E?$ is replaced by $c \leftrightarrow^*_{E^g} d$.

The linear-time algorithm of Paterson and Wegman [7] uses a term dag representation for terms (with a unique node for every variable). Construction of the term dag is the only use of congruence closure rule. Subsequently, decomposition steps are applied using a strategy that makes any use of the congruence closure steps unnecessary. All deductive inferences coming out of congruence closure are, of course, known a-priori through the $E?$-equations. The linear time algorithm described by Martelli and Montanari [5] achieves the same effect too.

The proposed view of integrating congruence closure with unification is useful in at least two other different contexts. First, it is easily extended, using additional transition rules, to describe rigid E-unification algorithms, see [10]. Secondly, it serves as a starting point to study Shostak's combination of congruence closure with certain E-unification algorithms [8].

References

[1] F. Baader and W. Snyder. Unification theory. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier Science Publisher B.V., 1999.

[2] L. Bachmair and A. Tiwari. Abstract congruence closure and specializations. In D. McAllester, editor, *17th Intl Conf on Automated Deduction, CADE-17*, 2000.

[3] P. C. Kanellakis and P. Z. Revesz. On the relationship of congruence closure and unification. *J. of Symbolic Computation*, 7(3/4):427–444, 1989.

[4] J-L. Lassez, M.J. Maher, and K. Marriott. Unification revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann Publishers, Inc., 1987.

[5] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Trans on programming languages and systems*, 4(2):258–282, 1982.

[6] G. Nelson and D. Oppen. Fast decision procedures based on congruence closure. *Journal of the Association for Computing Machinery*, 27(2):356–364, April 1980.

[7] M. S. Paterson and M. N. Wegman. Linear unification. *J. of computer and system sciences*, 16:158–167, 1978.

[8] R. E. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31(1):1–12, 1984.

[9] R. Socher-Ambrosius and P. Johann. *Deduction systems*. Springer-Verlag, New York, 1997.

[10] A. Tiwari, L. Bachmair, and H. Ruess. Rigid E-unification revisited. In D. McAllester, editor, *17th Intl Conf on Automated Deduction, CADE-17*, 2000.