UNIVERSITÄT AUGSBURG

Declarative Graph Algorithms via Knuth-Bendix Completion

Georg Struth

Report 2002-10 Juni 2002

INSTITUT FÜR INFORMATIK
D-86135 AUGSBURG

Declarative Graph Algorithms via Knuth-Bendix Completion

Georg Struth

Institut für Informatik, Universität Augsburg
Universitätsstr. 14, D-86135 Augsburg, Germany
struth@informatik.uni-augsburg.de

Abstract We propose combined and cooperating Knuth-Bendix completion procedures for equalities and inequalities. They serve as metaprocedures for developing rule-based declarative algorithms. Here, we present algorithms for memoization, cycle detection and strongly connected components. The specifications of algorithms are highly non-deterministic and generic. Refinements via evaluation strategies yield efficient implementations. Data structures and implementation details are largely hidden in the metaprocedure. Our algorithms easily adapt to dynamically changing environments. Main applications are constraint systems and the constraint-based analysis and verification of programs and finite and infinite state systems.
Keywords: term rewriting, Knuth-Bendix completion, graph algorithms, declarative programming, constraint systems.

1 Introduction

Declarative programming often means just writing a mathematical specification, since the specification *is* the program. Integrating efficiency, however, can be delicate, since control is not a main concern of mathematics. We propose Knuth-Bendix completion as a metaprocedure from rewriting and universal algebra for developing rule-based dynamic graph algorithms that are declarative and efficient. Our rule systems are abstract, non-deterministic and generic. They specify classes of algorithms rather than particular instances. Efficiency is provided by the control and implementation details of the metaprocedure and integrated by refinement via execution strategies. Algorithmic properties can be derived in a natural, precise and modular way from those of the metaprocedure.

We specify and prove soundness and correctness of a combined Knuth-Bendix completion procedure for (ground) inequalities and equalities. Knuth-Bendix procedures for inequalities (non-symmetric transitive relations and quasiorderings), even modulo associativity and commutativity and normalizing with respect to further axioms, have been proposed already in [21,23]. The combined procedure has been informally discussed and used in [23]. Intuitively, it constructs an equational and an inequational rewrite system in parallel, computing also critical pairs between equalities and inequalities and simplifying inequalities by equational rewrite rules. The resulting rewrite systems yield monotonic rewrite proofs for inequalities on (canonical representants of) equivalence classes of the canonical equational rewrite system.

The combined Knuth-Bendix procedure extends the scope of term rewriting in declarative programming and constraint systems. It provides a novel approach to developing efficient procedures and algorithms for graphs and state transition systems with an associated notion of equivalence on edges, vertices or states. We support this claim by two examples. We develop first a memoization algorithm and second algorithms for cycle detection and strongly connected components for (constraint) graphs. Algorithms for detecting (homeomorphic) embeddings in ground rewrite sequences are developed in [24]. In each case, simple, robust, non-deterministic and purely declarative specifications of algorithms arise immediately from mathematical definitions. Due to non-determinism, the algorithms support dynamic and on-line environments and are easy to parallelize. Modifications, heuristics and efficiency are obtained by refining via execution strategies (depth-first search for strongly connected components). Except mathematical concepts like tuples, sets, orderings and rewrite or inference rules, no data structures appear in programs. Just the opposite: our approach easily integrates simple and purely declarative specifications of the union by rank and path-compression heuristics of the union-find data structure to efficiently handle the equivalence classes of component graphs.

Our methods and algorithms are not only appropriate for rewrite-based programming environments like ELAN [1] or MAUDE [2]. There are further interesting applications. First, in the control of centralized reactive systems, for instance operation or data base systems. There, dynamic cycle detection helps avoiding deadlocks [19]. Second, in the constraint-based analysis and verification of programs and finite or infinite state systems. Rewriting techniques are at the core, for instance, of various successful approaches to cryptographic protocol analysis [8,14,12]. Collapsing cycles and strongly connected components may yield considerably smaller constraint stores and state spaces. Detecting such structures is also important for the acceptance analysis of Büchi automata.

The remainder of this text is organized as follows. Section 2 recalls some properties of terms, orderings, rewrite relations and graphs. Section 3 sketches the basics of non-symmetric rewriting. The combined completion procedure for inequalities and equalities is specified and proven sound and correct in section 4. It is further discussed in section 5. Section 6 gives the first application of the combined procedure: declarative dynamic memoization for constraint graphs. Section 7 gives the second application: simple and robust declarative dynamic algorithms for cycle detection and strongly connected components. These algorithms are further refined to efficient variants in section 8. Section 9 shows the strongly connected component algorithm at work. Section 10 contains a conclusion.

2 Preliminaries

Let $T_\Sigma(V)$ be a set of terms with signature Σ and variables in V. We write T_Σ, if $V = \emptyset$, that is for *ground terms*. We identify terms with $\Sigma \cup V$-labeled trees with nodes or *positions* in the monoid $\mathbb{N}^*$. ϵ denotes the root and ni the i-th child of node n. A variable is *linear* (*non-linear*) in a term, if it labels exactly one leaf (at least two leaves). *Substitutions*, that is mappings $\sigma : V \to T_\Sigma(V)$, are identified with their homomorphic extensions to $T_\Sigma(V)$. We write $s[t]_p$, if t is the subterm of s at position p and $r[s/t]_p$, if the subterm $s = r|_p$ of r at position p is replaced by t. We lift the bracket notation to other expressions. Apart from the leaves, substitutions preserve positions and labels of a tree. For a term t and a substitution σ, a node p of $t\sigma$ is a *skeleton position*, if it is an internal node of t. It is a *variable instance position* of $t\sigma$, if it is either a leaf or not a node of t.

Let $\to$ be a binary relation on a set A. We write $\leftarrow$ or $\to^\circ$ for its converse, $\leftrightarrow$ for its symmetric closure, $\to^+$ for its transitive closure and $\to^*$ for its reflexive transitive closure. Juxtaposition of relations denotes relational product. $\to$ is a *quasiordering*, if it is reflexive and transitive, a *partial ordering*, if it is also antisymmetric and an *equivalence*, if it is reflexive, transitive and symmetric. For every quasiordering $\preceq$, the relation $\sim\, = \,\preceq \cap \succeq$ is an equivalence and the relation $\leq$ on $A/\sim$ defined by $[x]_\sim \leq [y]_\sim$ iff $x' \preceq y'$ for some $x' \in [x]_\sim$ and $y' \in [y]_\sim$ is a partial ordering. When $\preceq$ is only transitive, $\sim$ and therefore $\leq$ need not be reflexive. $\to$ is *noetherian*, if all $\prec$-sequences are finite.

Let $\to$ be a binary relation on a term algebra A with associated set of terms $T_\Sigma(V)$. The operation f^A denoted by the n-ary operation symbol f is *monotone* (in each argument), if it satisfies the formula $s_1 \to t_1, \ldots, s_n \to t_n \Longrightarrow f(s_1, \ldots, s_n) \to f(t_1, \ldots, t_n)$, for all $s_1, \ldots, s_n, t_1, \ldots, t_n \in T_\Sigma(V)$. $\to$ is *compatible*, if every operation is monotone. For a quasiordering, this is the case iff $s \to t \Longrightarrow r[s]_p \to r[t]_p$ holds for all $r, s, t \in T_\Sigma(V)$ and positions p. $\to$ is *stable*, if $s \to t \Longrightarrow s\sigma \to t\sigma$ holds for all $s, t \in T_\Sigma(V)$ and endomorphisms σ of A. $\to$ is *fully invariant* or a *rewrite relation*, if it is compatible and stable. A rewrite relation is usually induced by a set of rewrite rules. A term s *rewrites* to a term t in one step at position p, if there is a rewrite rule $l \to r$ such that $s|_p = l\sigma$ for some position p and substitution σ and $t = s[r\sigma]_p$. This presumes that all predecessor nodes of p are labeled by names of monotonic functions. We also consider syntactic orderings on (ground) terms. A *reduction ordering* is a well-founded fully invariant quasiordering. For our purposes, in particular, we may assume that this reduction ordering contains the proper subterm relation.

Every finite binary relation $\to$ can be represented as a directed graph (digraph) $G = (V, \to)$ with set of vertices V and set of edges $\to$. A *k-path* from vertex u to vertex v in G is a sequence $v_0, \ldots, v_k$ of vertices such that $v_0 = u$, $v_k = v$ and $v_{i-1} \to v_i$ for all $1 \leq i \leq k$. k is called the *length* of the path. v is *reachable* from u, if $u \to^+ v$, that is, if for some k there is a k-path from u to v. A path is *simple*, if all its vertices are distinct. A *k-cycle* in G is a k-path with $v_0 = v_k$. A cycle is *simple*, if $v_1, \ldots, v_k$ are distinct. G is *acyclic*, if it contains no cycle. The *strongly connected components* (SCCs) of G are the equivalence classes of the relation $\approx\, = \,\leftarrow^* \cap \to^*$, which expresses mutual reachability. This definition includes that every vertex is reachable from itself. The *component graph* (SCC-graph) $G_{SCC} = G/\approx$ of G is defined on $\approx$-equivalence classes analogously to the definition of $\leq$ on $\preceq$. $[v]_\approx \to_\approx [w]_\approx$, iff $v' \to w'$ for some $v' \in [v]_\approx$ and $w' \in [w]_\approx$.

3 Non-Symmetric Rewriting

We presuppose the basic concepts and notation of equational rewriting (c.f. [13]). Introductions to non-symmetric rewriting can be found in [17,20]. Non-symmetric rewriting addresses reachability of binary relations. Let $\to_I$ be a binary relation and $\prec$ some noetherian syntactic ordering on a set A. Let $\to_I$ be partitioned into three sets $\to_R = \to_I \cap \succ$, $\to_S = \to_I \cap \prec$ and $\to_\Delta$ (non-orientable). R- and S-steps in paths can be separated, if the following local criteria are satisfied.

Lemma 1. *Let $\to_R$ and $\to_S$ be binary relations on some set A. Let $\to_R \cup \leftarrow_S$ be noetherian.*

(i) $\to_S\to_R \subseteq \to_R^+\to_S^* \cup \to_S^+ \quad \Longrightarrow \quad (\to_R \cup \to_S)^+ \subseteq \to_R^+\to_S^* \cup \to_S^+$,
(ii) $\to_S\to_R \subseteq \to_R^*\to_S^* \quad \Longrightarrow \quad (\to_R \cup \to_S)^* \subseteq \to_R^*\to_S^*$.

Lemma 1 (i) is adequate for transitive relations, lemma 1 (ii) for quasiorderings. We call paths in $\to_R^+\to_S^* \cup \to_S^+$ or $\to_R^*\to_S^*$ *rewrite paths* or *valleys*. Paths in $\to_S\to_R$ are called *peaks*. We say that $\to_R$ *semicommutes* over $\to_S$, when every peak can be replaced by a valley. Lemma 1 (ii) generalizes Newman's lemma. Setting $\to_R = \to_T = \leftarrow_S$, the left-hand side expresses local confluence of $\to_T$ and the right-hand side the Church-Rosser property. If $\to_T$ is a rewrite relation that is noetherian and has the Church-Rosser property, then every term has a unique normal form representing its $\leftrightarrow_T^*$-equivalence class. In non-symmetric rewriting, term normal forms do not exist.

Let now $\prec$ be a reduction ordering on $T_\Sigma(X)$. Let I be a set of term inequalities. Let R and S be sets of term rewrite rules decreasing from left to right and from right to left respectively with respect to $\prec$. Let $\to_I$, $\to_R$ and $\to_S$ be the associated rewrite relations. Then, semicommutation depends of the relative positions in a term where consecutive rewrite rules apply. This is analyzed in *critical pair lemmata* [20]. Intuitively, a critical pair is an element of $\to_S\to_R$ that prevents semicommutation. As in equational rewriting, critical pairs arise when one rewrite step takes place at a prefix position of the other and the lower one at a skeleton position. Applicability of the second step then depends on the result of the first step, the order of application may not be convertible and therefore there may be no semicommutation. In analogy to equational rewriting, we define a *skeleton critical pair* of rewrite rules $l_1 \to_R r_1$ and $l_2 \to_S r_2$ as either $(l_1\sigma[l_2\sigma]_p, r_1\sigma)$ for a most general unifier σ of r_2 and $l_1|_p$ and a skeleton position p of l_1 or as $(l_2\sigma, r_2\sigma[r_1\sigma]_p)$ for a most general unifier σ of $r_2|_p$ and l_1 and a skeleton position p of r_2. But unlike equational rewriting, there are also *variable critical pairs* involving variable instance positions of non-linear variables. Variable critical pairs seem to involve context variables, higher-order unification [17] and higher-order reduction orderings. Here, we consider only the ground case, but also the linear-variable case, including non-symmetric rewriting and completion modulo associativity and commutativity [20,21,23] are still first-order.

We consider combined Knuth-Bendix completion procedures where an equational rewrite relation $\rightarrow_T$ occurs in addition to the non-symmetric rewrite relations $\rightarrow_R$ and $\rightarrow_S$. This yields peaks and (skeleton) critical pairs between $\leftarrow_T\rightarrow_R$, $\rightarrow_S\rightarrow_T$ and $\leftarrow_T\rightarrow_T$. We consider rewrite paths of the form $\rightarrow_T^+\rightarrow_R^*\rightarrow_S^*\leftarrow_T^* \cup \rightarrow_R^+\rightarrow_S^*\leftarrow_T^* \cup \rightarrow_S^+\leftarrow_T^* \cup \leftarrow_T^+$ for a transitive relation and $\rightarrow_T^*\rightarrow_R^*\rightarrow_S^*\leftarrow_T^*$ for a quasiordering. For confluent and terminating $\rightarrow_T$, that is, when $\rightarrow_T$ computes unique normal forms for equivalent terms, rewrite paths operate solely on T-normal forms.

In absence of critical pairs and when $\rightarrow_R$, $\leftarrow_S$ and $\rightarrow_T$ are noetherian, we obtain a search procedure for reachability along rewrite paths. For a non-symmetric transitive relation, $s \rightarrow_I^+ t$ holds, if either the T-normal form of s and the T-normal form of t are in the reflexive part of $\rightarrow_I^+$ or if some $\rightarrow_R$ path from the T-normal form of s and some $\leftarrow_S$-path from the T-normal form of t have a common vertex. This is a decision procedure, if $\rightarrow_R$ and $\leftarrow_S$ are finitely branching. For a quasiordering, the reflexive part of $\rightarrow_I$ need not be considered. When A is just a set of points, the decision procedure takes time linear in the size of $A \cup \rightarrow_R \cup \rightarrow_S$, for instance by depth-first search. When A carries a ground term structure, memoization yields polynomial time complexity [16,7]. In section 6, we model memoization declaratively as an extension of the congruence closure approach in [15,6]. The argument easily generalizes to combined systems.

We call a pair (R, S) of sets of rewrite rules enforcing the decision procedure for reachability of non-symmetric rewriting a *normal system*. By definition, all critical peaks can be joined by a rewrite proof, $\rightarrow_R$ and $\leftarrow_S$ are noetherian and no rule from $\rightarrow_R$ or $\rightarrow_S$ can be deleted. A set T of equational rewrite rules is *canonical*, if the induced term rewrite system is confluent and noetherian.

4 Ground Completion for Equalities and Inequalities

Equational Knuth-Bendix completion procedures (KB-procedures) belong to the basics of universal algebra, computer algebra and computational logic. They are presented in most introductions to term rewriting. KB-procedures for non-symmetric transitive relations and quasi-orderings have been proposed and discussed in [20,23] and extended to completion modulo associativity and commutativity (including normalization with respect to idempotence) in [20,21]. The combination of KB-procedures for inequalities and equalities allows an interleaved construction of a canonical term rewrite system for equalities and a normal system for inequalities. In opposition to normalized completion [18], the equational rewrite system is not statically prescribed. The combination is straightforward and intuitive and has already been informally introduced in [23]. Here, our applications to declarative algorithms require a more formal presentation.

A KB-procedure implements a state transition system together with a syntactic reduction ordering $\prec$ on terms, (oriented) equalities and inequalities and paths. States are tuples of sets of equalities, inequalities and rewrite rules. The transition relation is specified by transition rules of two kinds. First, deductive inference rules for adding certain consequences to a state, corresponding to critical pair computations. Second, simplification steps that combine deduction steps with deletions implementing an (approximative) notion of redundancy: An (oriented) inequality is *redundant*, if it can be replaced by a smaller path.

We denote the combined KB-procedure for a non-symmetric transitive relation by C_t and that for a quasiordering by C_q.

For C_t, states are of the form (S, Δ, P). Thereby, $S = I \cup E$, where I is a set of inequalities, E is a set of equalities on T_Σ, $\Delta \subseteq \{(t,t) : t \in T_\sigma\}$, $P = R \cup S \cup T$, where R, S and T are sets of rewrite rules for inequalities and equalities on T_Σ.

The intended meaning of S is *specification*, that of P is *program*. In the initial state or *initial specification*, P is empty. For, C_q, states are pairs (S, P), that is, Δ can be disregarded, since the reflexive part of the relation need not be stored.

A *run* of C_t or C_q is a (finite or infinite) sequence $q_0, q_1, q_2, \ldots$ of states such that q_0 is an initial state and consecutive states are related by transitions.

We also define *limit states* $(S_\infty, \Delta_\infty P_\infty)$ and (S_∞, P_∞) of C_t and C_q, where

$$S_\infty = \bigcup_{i=0}^{\infty}(\bigcap_{j=i}^{\infty} S_j), \qquad \Delta_\infty = \bigcup_{i=0}^{\infty} \Delta_i, \qquad P_\infty = \bigcup_{i=0}^{\infty}(\bigcap_{j=i}^{\infty} P_j).$$

We say that a run *fails*, if $S_\infty \neq \emptyset$. It *succeeds*, if it does not fail, if T_∞ yields a canonical system for E and the equational part of I and if R_∞ and S_∞ yield a normal system on E-equivalence classes. A run is *fair*, if every enabled transition is eventually executed. Then every relevant critical pair will eventually be computed. A completion procedure is *correct*, if every run that does not fail succeeds and the limit rewrite system is equivalent to the initial specification.

To obtain a more compact notation, we henceforth write $s \rightarrow_I t$, $s \rightarrow_E t$ and $s \rightarrow_\Delta t$ also for rewrite rules. We also write $\rightarrow_{R \cup S}$ instead of $\rightarrow_R \cup \rightarrow_S$, $\rightarrow_{R^\circ}$ instead of $\leftarrow_R$ and likewise. Finally, we use a function ϕ that maps R to I, S° to I° and T to E. ϕ thus forgets the orientation.

We now define the deduction and simplification rules of C_t and C_q. This can be done in parallel, because the rules are very similar.

$$\frac{(S, \Delta, P)}{S \cup \{s \rightarrow_I t\}, \Delta, P)}, \qquad \text{(Deduce)}$$

if (s, t) is a critical pair involving one of R or S. It is well-known from equational completion that critical pairs from T are subsumed by simplification rules. Deduce can be written as a pair of inference rules on equalities and inequalities.

$$\frac{l_1[r_2] \rightarrow_{R \cup T} r_1 \qquad l_2 \rightarrow_S r_2}{l_1[r_2/l_2] \rightarrow_I r_1}, \qquad \frac{l_1[l_2] \rightarrow_{S^{-1} \cup T} r_1 \qquad l_2 \rightarrow_R r_2}{r_1 \rightarrow_I l_1[l_2/r_2]}.$$

Thereby rewriting into T-rules occurs at strict subterms.

$$\frac{(S \cup \{s \rightarrow_{\phi(X)} t\}, \Delta, P)}{(S, \Delta, P \cup \{s \rightarrow_X t)}, \qquad \text{(Orient)}$$

where $s \succ t$. In C_t, we store the reflexive part of $\rightarrow^*_{I\cup E}$ in Δ.

$$\frac{(S \cup \{s \rightarrow_{I\cup E} s\}, \Delta, P)}{(S, \Delta \cup \{s \rightarrow_\Delta s\}, P)}, \qquad \text{(Diagonal)}$$

Diagonal can be restricted to un-oriented expressions, when it is eagerly applied. Diagonal is of course not used in C_q. In case of a quasiordering or of equalities, we delete reflexive parts instead.

$$\frac{(S \cup \{s \rightarrow_{I\cup E} s\}, P)}{(S, P)}. \qquad \text{(Delete)}$$

The inequational simplification rules are based on search for smaller proofs.

$$\frac{(S, \Delta, P \cup \{s \rightarrow_{I\cup R\cup S} t\})}{(S, \Delta, P)}, \qquad \text{(ISimplify)}$$

if $s \rightarrow_I t$, $s \rightarrow_R t$ or $s \rightarrow_S t$ can be replaced by a smaller proof (also using E and T). ISimplifiy should only be used if none of the following rule applies.

The equational simplification rules use equational rewrite rules to simplify oriented and un-oriented equalities and inequalities.

$$\frac{(S[l], \Delta, P \cup \{l \rightarrow_T r\})}{(S[l/r], \Delta, P \cup \{l \rightarrow_T r\})}. \qquad \text{(Simplify)}$$

$$\frac{(S, \Delta \cup \{s[l] \rightarrow_\Delta s[l]\}, P \cup \{l \rightarrow_T r\})}{(S, \Delta \cup \{s[l/r] \rightarrow_\Delta s[l/r]\}, P \cup \{l \rightarrow_T r\})}. \qquad \text{(DSimplify)}$$

DSimplify could be restricted to one side. The result of simplification should then be stored in I and be shifted back into Δ after applying Simplify. The rule need not be considered in case of a quasiordering.

$$\frac{(S, \Delta, P \cup \{s[l] \rightarrow_X t, l \rightarrow_T r\})}{(S \cup \{s[l/r] \rightarrow_{\phi(X)} t\}, \Delta, P \cup \{l \rightarrow_T r\})}. \qquad \text{(Collapse)}$$

For $X = T$, either rewriting occurs at a strict subterm, or else $s = l$ and $t \succ r$.

$$\frac{(S, P \cup \{s \rightarrow_X t[l], l \rightarrow_T r\})}{(S, P \cup \{s \rightarrow_{\phi(X)} t[l/r], l \rightarrow_T r\})}. \qquad \text{(Compose)}$$

C_t and C_q specialize to an equational KB-procedure [13] by forgetting all rules involving I, R and S. C_t specializes to a KB-procedure for non-symmetric transitive relations [20,23], forgetting all rules involving E and T. C_q specializes to a procedure for quasiorderings [20,23], forgetting all rules involving E and T

and discarding Δ. Conversely, as an extension of these procedures, the combined procedures add just a few more simple cases to well-known rules.

The same holds for the proofs of soundness and correctness of C_t and C_q with respect to those of equational and non-symmetric KB-completion. We define $S \uparrow = \bigcup_{i \geq 1} S_i$ and $P \uparrow = \bigcup_{i \geq 1} P_i$. A proof of an identity $s \rightarrow_E t$ or an inequality $s \rightarrow_I t$ in $S \uparrow \cup P \uparrow$ is a finite sequence $(s_0, \ldots, s_n)$ of length greater than one such that $s = s_0$, $t = s_n$ and for all $1 \leq i \leq n$,

1. $s_{i-1} \rightarrow_{E\uparrow \cup I\uparrow} s_i$,
2. $s_{i-1} \rightarrow_{R\uparrow \cup S\uparrow \cup T\uparrow \cup (T\uparrow)^\circ} s_i$.

Pairs (s_{i-1}, s_i) are called proof steps. Two proofs are called *equivalent*, if they prove the same pair (identity or inequality). A proof $(s_0, \ldots, s_n)$ in $S \uparrow \cup P \uparrow$ is a *rewrite proof* in P_∞, iff there exists some $0 \leq k_1 \leq k_2 \leq k_3 \leq n$, such that $s_{i-1} \rightarrow_T s_i$ for all $1 \leq i \leq k_1$ and $s_i \rightarrow_R s_{i+1}$ for all $k_1 \leq i < k_2$, $s_i \rightarrow_S s_{i+1}$ for all $k_2 \leq i < k_3$ and $s_i \leftarrow_T s_{i+1}$ for all $k_3 \leq i < n$,

Let M be the set of finite multisets of terms from T_Σ. Let $\prec$ be some reduction ordering on T_Σ and $\lhd$ the (strict) subterm ordering. We compare tuples in $M \times T_\Sigma \times T_\Sigma$ by a proof ordering $\prec'$ which is the lexicographic combination of the multiset extension of $\prec$ (also denoted by $\prec$) for the first component, $\lhd$ for the second component and $\prec$ for the third component. A proof step measure is a mapping $\mu : T_\Sigma \times T_\Sigma \rightarrow A$ defined for a proof step $w = (s_{i-1}, s_i)$ by

$$\mu : (w) \mapsto \begin{cases} (\{s_{i-1}, s_i\}, \_, \_), & \text{if } s_{i-1} \rightarrow_{E\uparrow \cup I\uparrow} s_i, \text{ (\_ denotes an arbitrary term)}, \\ (\{s_{i-1}\}, l, s_i), & \text{if } s_{i-1} \rightarrow_{R\uparrow \cup T\uparrow} s_i, \text{ by rule } l \rightarrow r, \\ (\{s_i\}, \_, \_), & \text{if } s_{i-1} \rightarrow_\Delta s_i, \\ (\{s_i\}, l, s_{i-1}), & \text{if } s_{i-1} \rightarrow_{S\uparrow \cup (T\uparrow)^\circ} s_i, \text{ by rule } l \rightarrow r, \end{cases}$$

This is the definition for C_t. For C_q, Δ disappears. Proofs are measured as multisets of proof steps. $\prec'$ is extended to a proof ordering $\prec''$ by multiset extension. Since we consider only (bounded) lexicographic combinations and multiset extensions of well-founded orderings, both $\prec'$ and $\prec''$ are well-founded. We will denote them by $\prec$ if no confusion may arise.

Lemma 2. *Every run of C_t and C_q has the following invariants.*

(i) R, S° and T are contained in $\succ$.
(ii) The (in)equational theory of S_0 is preserved (completion soundness).

Proof. (ad i) Trivial inspection of the inference rules.

(ad ii) By inspection of the inference rules. Orient, Diagonal and Delete are trivial cases. Deduce rules are restricted applications of transitivity and monotonicity. This is sound. ISimplify rules are sound by definition. When an inequality $s \rightarrow_I t$ (or a respective rewrite rule) can be replaced by some smaller proof, then $s \rightarrow_I t$ is redundant; removing it does not alter the theory.

In case of the Simplify rules, we consider the case

$$\frac{(S \cup \{s[l] \rightarrow_I t\}, P \cup \{l \rightarrow_T r\})}{(S \cup \{s[r] \rightarrow_I t\}, P \cup \{l \rightarrow_T r\})}$$

First, we can use transitivity to derive

$$\frac{(S \cup \{s[l] \to_I t\}, P \cup \{l \to_T r\})}{(S \cup \{s[l] \to_I t, s[r] \to_I t\}, P \cup \{l \to_T r\})}$$

from the premise without altering the theory. But now $s[l] \to_t$ is entailed by the smaller expressions $s[r] \to_I t$ and $l \to_T r$. It is therefore redundant and can be deleted without changing the theory.

The remaining rules are handled in a similar way. $\square$

Lemma 3. *Assume that* C_t *and* C_q *is fair and does not fail. There is a smaller equivalent proof for every proof in* $S \uparrow \cup P \uparrow$ *that is not a rewrite proof in* P_∞*.*

Proof. There are three reasons for not being a rewrite proof.

1. The proof contains a step in $S \uparrow$.
2. It contains a step in $P \uparrow - P_\infty$.
3. It contains a peak in P_∞.

We show that each of these cases can be transformed away.

(ad 1) Since the procedure does not fail, $S_\infty = \emptyset$. Thus the step in $S \uparrow$ has been deleted at some stage of the run. This can happen either by applying Diagonal (in case of C_t), Delete (in case of C_q), Orient or by simplifying it. The first and second case obviously yield a smaller proof. In the second case this is also the case, because the first component of μ decreases. In the third case, for ISimplify, the decreases by definition. Otherwise, with the Simplify, the initial proof steps can be replaced by two new ones, which are both smaller by the first component of the measure.

(ad 2) Assume without loss of generality that there is a proof step $s_{i-1} \to_R s_i$ with a rule $l \to_R r$ such that $l = s_{i-1}|_p$. For the cases of S, T or T° the situation is similar, by definition of μ. Then the rule is eventually discarded at the run, either by ISimplify, by Collapse or by Compose.

In case of ISimplify we argue like in (1).

In case of Collapse, assume that $l \to_R r$ has been replaced by the inequality $l' \to_I r$, using the rule $s \to_T t$. Hence $l|_q = s$ and $l' = l[t]_q$ and q is not the root position. Then the initial proof step can be replaced by two new ones. The measure of the new inequality is smaller by the fist component (because a subterm is replaced by a smaller one) of the measure and the new rule is smaller by the subterm ordering.

In case of Compose, assume that $l \to_R r$ has been replaced by $l \to_R r'$ using the rule $s \to_T t$, such that $r|_p = s$ and $r' = r[t]_p$. Then the initial proof step can be replaced by two new ones. The new R-step is smaller than the initial one by the third component of the measure, the S-step is smaller by the first component (and replacement of subterms).

(ad 3) We apply the critical pair lemma. For disjoint positions we can commute the rewrite steps to turn the peak into a valley. This yields of course a smaller proof. Or else we apply Deduce and again obtain a smaller proof. $\square$

Lemma 4. *Consider an unfailing fair run of C_t or C_q.*

(i) Every proof in $S \uparrow \cup P \uparrow$ is equivalent to a rewrite proof in P_∞.
(ii) T_∞ is convergent. (R_∞, S_∞) is a normal system on E-equivalence classes.

Proof. (ad i) By well-founded induction on the size of proofs, using the proof ordering and lemma 3.

(ad ii) A special case of (i). □

The statements of lemma 4 can also be expressed as follows.

Theorem 1. *Every fair implementation of C_t and C_q is correct.*

5 Extensions and Limitations

There are three main differences between non-symmetric and equational KB-procedures. First, even in the ground case, Deduce is not subsumed by ISimplify. Second, ISimplify is based on search and cannot, as in the equational case, be refined by one-step rules. The third difference is even more interesting.

Lemma 5.

(i) Straightforward implementations of C_t and C_q admit infinite runs.
(ii) All runs of C_t and C_q terminate, if Σ is a (finite) set of constants.
(iii) There is a transformation on the initial expressions that forces termination of C_t and C_q.

Proof. (ad i) See [23] for an example.

(ad ii) Let Σ be a finite set of constants. Then the C_t- and C_q-rules correspond to addition, relabeling and deletion of edges in a graph. Since no edge that has been deleted must ever be added again (because it is redundant) and every edge is relabeled at most once, the procedure terminates after at most $3|\Sigma|^2$ steps.

(ad iii) See [24] for a definition of the transformation (essentially currying and flattening of terms) and a proof of termination in polynomial running time with respect to the size of the initial expressions. In particular, flattening of inequalities can be achieved as a preprocessing, using the memoization procedure $\mathsf{C}_\mathsf{t}^\mathsf{M}$ from section 6. □

All these phenomena (except termination) are further discussed in [23]. Due to variable critical pairs, KB-procedures for the non-linear non-ground case require context variables. The associated context unification problem [9] is still open (unifiers could however be computed in practical cases by higher-order unification). Defining appropriate reduction orderings seems difficult.

There are three interesting cases that can still easily be handled. First the non-compatible case (no function is monotonic), second the linear case and third the ground case modulo associativity and commutativity.

The non-compatible case is interesting as the basis for ordered chaining calculi for transitive relations and quasiorderings [4,22].

The KB-procedure of the linear case is obvious. Linearity is an invariant of the process. The transition rules combine the equational non-ground with the non-symmetric ground case. One of the DEDUCE rules, for instance, has the form

$$\frac{l_1[r_2] \rightarrow_{R \cup T} r_1 \qquad l_2 \rightarrow_S r_2'}{l_1\sigma[l_2\sigma] \rightarrow_I r_1\sigma}.$$

Thereby σ is a most general unifier of r_2 and r_2'. Now all possible combinations between rules in R, S and T must be considered.

A particularly interesting instance is a ground non-symmetric KB-procedure modulo associativity and commutativity (AC), since the associativity law and the commutativity law, which are the only non-ground rules, are linear. Here one uses an AC-compatible reduction ordering [5,11], AC-unification and extended rules. A Knuth-Bendix procedure for ground quasiorderings modulo AC has already been specified in [21]. The extension to the combined case is obvious. For an AC-operation symbol f, a rule $f(r,s) \rightarrow_{R \cup T} t$ or $r \rightarrow_S f(t,s)$ is extended to $f(f(r,s),x) \rightarrow_{R \cup T} f(t,x)$ or $f(r,x) \rightarrow_S f(f(t,s),x)$, where x is a fresh extension variable. Hence also extensions preserve linearity. Now, extension rules are added to the completion procedure. The rules

$$\frac{(I,R,S)}{(I,R \cup \{s \rightarrow_{R \cup T} t\},S)}, \qquad \frac{(I,R,S)}{(I,R,S \cup \{s \rightarrow_S t\})}, \qquad \text{(EXTEND)}$$

if $s \rightarrow_{R \cup T} t$ ($s \rightarrow_S t$) extends a non-extended rule in R or T (in S), are the extension rules for a ground KB-procedure for a quasiordering.

Variable critical pairs also occur for the interaction of inequational and equational rewrite rules. It immediately follows from the critical pair lemma of non-symmetric rewriting that there are also variable critical pairs of R or S with T, when T is non-linear and the R- or S-rule is applied at a variable-instance position. Conversely, when R or S is non-linear and T is applied at a variable-instance position, there is semicommutation and therefore no variable critical pair.

6 Memoization in Constraint Graphs

We now present a first application of C_t. A declarative memoization procedure for digraphs, where vertices are labeled by terms. This simple adaption of the declarative congruence closure algorithms in [15,6] allows efficient representations of constraint graphs. The main idea is to use the equational rewrite system $\rightarrow_T$ for renaming all subterms in the inequalities presented by I. Since C_t and C_q are non-deterministic, memoization can be performed on the fly during the completion process, although it is preferably done in a bottom-up way. Ambiguous assignments of names can be resolved by the equational COMPOSE and COLLAPSE rules. The memoization algorithms can therefore easily be adapted to dynamically changing environments. Renaming techniques are also standard for obtaining polynomial time decision procedures [16,7].

We now define the state transition system C_t^M for memoization as an extension of C_t. An adaption to a system C_q^M is straightforward. Renaming introduces new names. We therefore use a set C of constants disjoint from Σ and consider (ground) terms over $T_{\Sigma \cup C} = T_\Sigma(C)$. Following [15] we call a rule of the form $f(c_1, \dots, c_n) \to_T c_0$, where $f \in \Sigma$, $c_0, \dots, c_k \in C$, a *D-rule* and rules of the form $c_0 \to_X c_1$, where X is one of E, I, R, S, or T and $c_0, c_1 \in C$, *C-rules.* D-rules represent the renaming definitions, hence equivalences and thereby the structure of terms. Equational C-rules represent equivalence or congruence classes. Inequational C-rules represent the original digraph.

States of C_t^M are now tuples (K, S, Δ, P), where in addition to the states of C_t, K is a set of constants, which in the initial state is empty and E is a set of C- and D-rules, which is also initially empty.

In presence of new names, the ordering $\prec$ looses importance. It should only be total on C and all elements of C should be smaller than the elements of Σ. It can be constructed on the fly during completion.

Most of the transition rules of C_t^M are restrictions of those in C_t. But we must also add a rule that does the renaming with constants from C.

$$\frac{(K, S[f(c_1, \dots, c_n)], \Delta, P)}{(K \cup \{c_0\}, S[f(c_1, \dots, c_n)/c_0], \Delta, P \cup \{f(c_1, \dots, c_n) \to_T c_0\})} \qquad (\textsc{Rename})$$

where $c_1, \dots, c_n \in K$, $c_0 \in C$ and $c_0 \notin \Sigma \cup K$. The rule entering T is a D-rule. Rename forces renaming in a bottom up way.

After renaming, we can restrict Orient to equalities and inequalities with one side a constant. Simplify and DSimplify are restricted such that the T-rules replace (sub-)terms by constants. In Deduce, Collapse and Compose, both rules are restricted to have a constant at least at one side. In Compose, the T-rule consists solely of constants, when it is applied at a strict subterm. All expressions introduced by these rules have again a constant at least at one side.

There is only one obvious difference to declarative equational congruence closure. In C_t^M, like in C_t and C_q, one cannot dispense with Deduce. Soundness and correctness of C_t^M are almost immediately inherited from C_t.

Theorem 2. *C_t^M is sound.*

Proof. Soundness with respect to all inference rules of C_t^M except Rename follows immediately from soundness of C_t in lemma 2 (ii). Introducing a new name c_0 for a term $f(c_1, \dots, c_n)$ in Rename means a definitional extension of the original theory. Obviously, every theorem of the original theory is a theorem of the extension. Moreover, for every formula ϕ in the extended language there is a formula ψ of the original language that is equivalent in the extended theory. ψ can be obtained from ϕ by replacing every occurance of c_0 by $f(c_1, \dots, c_n)$. □

Theorem 3. *Every fair implementation of C_t^M is correct.*

Proof. The proof is again by well-founded induction on the size of proofs. It goes along the lines of lemma 3 and lemma 4. Again we must show that for every non-rewrite proof in $S\uparrow \cup P\uparrow$ there is a smaller proof in P_∞. Most cases are covered by C_t. In addition, it may happen that some step in $S\uparrow$ must be renamed before ORIENT is applicable. Obviously, RENAME yields a smaller equivalent proof. Renaming, all rules can be eventually oriented and for all oriented rules, at least one side is a constant from C. Therefore, the restrictions of simplification and DEDUCE rules of C_t are satisfied a forteriori. In particular, by COMPOSE, T_∞ consists solely of D-rules. □

Like already for C_t and C_q, there is one crucial difference to the equational congruence closure algorithms.

Theorem 4. *Runs of C_t^M need not terminate.*

Proof. We adapt the counterexample from [23] that shows non-termination of non-symmetric KB-procedures. Consider the rewrite rules $f(b) \to_I b$ and $f(a) \to_I b$ and assume names $c_0, c_1, \dots \in C$ such that $c_i \succ c_j$ for all $i < j$ and $c_i \prec f$ for all $c_i \in C$ and $f \in \Sigma$. Then RENAME yields

$$a \to_T c_0, \qquad b \to_T c_1, \qquad f(a) \to_T c_2, \qquad f(b) \to_T c_3.$$

RENAME, ORIENT and COLLAPSE and COMPOSE, then yield for all $i \geq 0$

$$f(c_{2i}) \to_T c_{2i+2}, \qquad f(c_{2i+1}) \to_T c_{2i+3}, \qquad c_{2i+2} \to_S c_{2i}, \qquad c_{2i+3} \to_S c_{2i+1}$$

and in the next step by DEDUCE

$$f(c_{2i+2}) \to_R c_{2i+2}, \qquad f(c_{2i+3}) \to_R c_{2i+3}.$$

This means non-termination. □

However, C_t^M terminates in many cases.

Example 1. We show that the rules $a \to_I f(f(f(a)))$ and $f(f(f(f(f(a))))) \to_I a$ imply $f(a) \to_I a$. Renaming yields

$$a \to_T c_0, \quad f^1(a) \to_T c_1, \qquad f^2(a) \to_T c_2, \qquad f^3(a) \to_T c_3,$$
$$f^4(a) \to_T c_4, \qquad f^5(a) \to_T c_5.$$

We order the constants according to the size of their indices and obtain

$$c_0 \to_S c_3, \qquad c_5 \to_R c_0, \qquad f(c_0) \to_T c_1, \qquad f(c_1) \to_T c_2,$$
$$f(c_2) \to_T c_3, \qquad f(c_3) \to_T c_4, \qquad f(c_4) \to_T c_5.$$

In the next step we obtain

$$f(c_0) \to_R c_4, \qquad c_1 \to_S c_4, \qquad f(c_1) \to_R c_5, \qquad c_2 \to_S c_5, \qquad c_2 \to_R c_0.$$

This corresponds to $f^2(a) < a$. In the next step we obtain

$$c_3 \to_S f(c_0), \qquad c_3 \to_R c_1, \qquad c_0 \to c_1.$$

and therefore $a \to_I f(a)$, as expected.

However, in accordance with lemma 5, a variant of C_t^M terminates, where RENAME is used as a preprocessing and is not intertwined with DEDUCE-steps, as in the previous example. This is the case, since without DEDUCE-steps, no new terms, that would require new names, are introduced.

7 Strongly Connected Components

The memoization example of section 6 shows the combination of the equational and the inequational procedure. Here we address the far more interesting example of cycle detection and strongly connected component algorithms that shows the *cooperation* of the procedures.

Consider a digraph $G = (V, \rightarrow_I)$ together with a total well-founded ordering $\prec$ on V. Thus there is no term structure for vertices. Every equality and inequality can be oriented. A path is *oriented*, if it contains only edges labeled with R or S. We have defined SCCs and the mutual reachability relation $\approx\, = \leftarrow_I^* \cup \rightarrow_I^*$ in section 2. We call an SCC *trivial*, if it contains one single vertex and *non-trivial* else. We use $\rightarrow_T$ (or more precisely $\leftrightarrow_T^*$) for representing $\approx$. The definition of $\approx$ is procedurally insufficient, since mutual reachability is a global property of $\rightarrow_I$. We therefore refine it and define the local variant $\approx_1\, =\, \leftarrow_I \cap \rightarrow_I$. Obviously, $\approx_1\, \subseteq\, \approx$. We model $\approx_1$ by the rule

$$\frac{(S, \Delta, P \cup \{l \rightarrow_R r, r \rightarrow_S l\})}{(S, \Delta \cup \{r \rightarrow_\Delta r\}, P \cup \{l \rightarrow_T r\})}. \qquad \text{(SCC)}$$

Adding the rule SCC to C_t yields the state transition system C_{SCC}. In the initial state, we assume that the input digraph is presented by inequalities in I. E, Δ and P are assumed to be empty. We also assume that all Orient and Diagonal steps are carried out implicitly together with the other rules.

We now show that adding just the single rule SCC to C_t suffices for detecting all SCCs and constructing the component graph G_{SCC} of G.

Lemma 6. *All runs of C_{SCC} terminate.*

Proof. A simple adaption of the argument in the proof of lemma 5 (ii). □

For soundness and correctness we must include the rule SCC into the proof measure μ. It is easy to modify μ such that also the conclusion of SCC is smaller than its premise. Just make inequalities bigger than equalities by adding their terms twice to the multiset.

Lemma 7. *C_{SCC} is sound.*

Proof. Like for C_t in lemma 2, we show that all runs of C_{SCC} preserve the initial theory, that is reachability and mutual reachability defined by $\rightarrow_I^+$ and $\approx$. By lemma 2, all runs of C_t preserve reachability and therefore also mutual reachability. It therefore remains to consider SCC.

SCC adds an equational rule $l \rightarrow_T r$ in presence of $l \rightarrow_R r$ and $r \rightarrow_S l$. Since therefore $(l, r) \in\, \approx_1$ and $(l, r) \in\, \approx$, adding the conclusion is sound. Since moreover the inequalities in the premise are entailed by the smaller conclusion they are redundant and can be deleted after adding the conclusion. Thus reachability and mutual reachability are preserved in every run of C_{SCC}. □

We now address correctness of C_{SCC}. Before presenting formal arguments, we try to give the underlying intuition. Roughly, the non-symmetric rewrite system is used to detect cyclic structures, the equational rewrite system to collapse cycles and represent equivalence classes. The SCC rule couples the two KB-procedure, hence establishes their cooperation.

Let us now take a closer look at the two main phases of the SCC algorithm. They are also illustrated by the example in section 9. For the first phase, it is important to observe that by definition of $\approx$, a SCC can be understood as a cluster of simple cycles (c.f. lemma 8 (i)). Moreover, every simple cycle must contain at least one R- and one S-step (c.f. lemma 8 (i)). This holds, since pictorially spoken, one can neither walk up forever or walk down forever on an oriented cycle. So consider a simple oriented k-cycle. As a consequence of lemma 8, it must contain a critical pair $\rightarrow_S\rightarrow_R$. Thus DEDUCE adds a new edge to the cycle and thereby generates a $k-1$-cycle (c.f. lemma 9 (i)). Iterating this argument, every fair run of C_{SCC} will eventually generate a 2-cycle, for which SCC introduces a T-rule (c.f. proof of theorem 5 (i)). This alone suffices for cycle detection.

We now turn to the second phase of the SCC algorithm. Once a T-rule $l \rightarrow_T r$ has been introduced, SIMPLIFY and COLLAPSE allow replacing all occurances of l in an I- R- or S-edge by r (c.f. lemma 9 (ii)). Thereby every i-cycle, $3 \leq i \leq k$, obtained from DEDUCE-steps on a k-cycle is collapsed to an $i-1$-cycle. In particular, a new 2-cycle appears and consequently, a new T-rule is introduced by SCC. Iterating this argument, the initial k-cycle is eventually collapsed into a 2-cycle and then to a single vertex by SCC. All in- and outgoing I- R- and S- edges of the initial k-cycle are now in- and outgoing edges to this surviving vertex. The remaining vertices of the cycle do no longer contribute to I- R- and S-rules. They are connected by T-rules to the surviving vertex instead (c.f. proof of theorem 5 (i)). Algebraically, C_{SCC} collapses the cycle into an equivalence class of the component graph, which is now represented by T-rules pointing from each member of the class to its canonical representant. The I- R- and S-rules apply solely at the canonical representant and thereby rewrite modulo the equivalence class.

Since a SCC is a cluster of simple cycles, it is eventually entirely collapsed to a canonical representant and represented by T. The equational part of C_{SCC}, in particular COMPOSE, transforms T eventually into a canonical term rewrite system whose term normal forms are the canonical representants of the equivalence classes. Remind that C_{SCC} does all these operations automatically, just by adding the rule SCC to C_{SCC}.

We now formally reconstruct these arguments.

Lemma 8. *Let $v_1, v_2 \in V$.*

(i) $v_1 \approx v_2$ iff there is a cycle through v_1 and v_2.
(ii) Every oriented cycle through v_1 and v_2 contains both $R-$ and S-edges.

Proof. (ad i) obvious.

(ad ii) By induction on the length of the cycle. For a 2-cycle with nodes v_0 and v_1, we have that $v_0 \rightarrow_S v_1$ and $v_1 \rightarrow_R v_0$, if $v_0 \prec v_1$ and $v_0 \rightarrow_R v_1$ and $v_1 \rightarrow_S v_0$, if $v_1 \prec v_0$. With an n-cycle, we associate an $n-1$-cycle by arbitrarily selecting three nodes v_i, v_{i+1} and v_{i+2} such that $v_i \rightarrow_I v_{i+1} \rightarrow_I v_{i+2}$, adding an edge $e = v_i \rightarrow_I v_{i+2}$. By the induction hypothesis, the $n-1$-cycle contains at least one R- and one S-edge. If the selected R and S-edge are not e, then these edges are also in the n-cycle. Else assume that e is in S and there is another R-edge in the $n-1$-cycle and therefore also in the n-cycle. The case where e is in R is dual. Now by the assumption, $v_i \prec v_{i+2}$. There are three cases for v_{i+1}.

– $v_{i+1} \prec v_i$. Then $v_i \rightarrow_R v_{i+1} \rightarrow_S v_{i+2}$.
– $v_i \prec v_{i+1} \prec v_{i+2}$. Then $v_i \rightarrow_S v_{i+1} \rightarrow_S v_{i+2}$.
– $v_{i+1} \succ v_{i+2}$. Then $v_i \rightarrow_S v_{i+1} \rightarrow_R v_{i+2}$.

So in any case there is one edge labeled with S. □

Lemma 9.

(i) For every oriented k-cycle, $k > 1$, DEDUCE *constructs a $k-1$-cycle.*
(ii) Let v_i and v_{i+1} be adjacent vertices on an oriented k-cycle c, $k > 1$. Let $v_{i+1} \rightarrow_T v_i$, $1 < i < k-1$. Then COMPOSE *and* COLLAPSE *replace c by a $k-1$-cycle, in which only v_{i+1} has been discarded and all in- and outgoing edges of v_{i+1} have been replaced by in- and outgoing edges of v_i.*

Proof. (ad i) Let c be a k-cycle. By lemma (ii), c contains at least one edge labeled with R and one edge labeled with S and therefore a peak, $v_1 \rightarrow_S \rightarrow_R v_2$ say . DEDUCE then derives $v_1 \rightarrow_I v_2$.

(ad ii) This is straightforward. COLLAPSE reduces all bigger and COMPOSE all smaller vertices in the respective edges. □

Theorem 5. *Every fair implementation of C_{SCC} is correct.*

(i) Let v be the $\prec$-minimal element of $[v]_\approx$. Then there exists a rule $v \rightarrow_{\Delta_\infty} v$ and for all $v' \neq v$ in $[v]_\approx$ there exists a rewrite rule $v' \rightarrow_{T_\infty} v$. Thus $\rightarrow_{T_\infty}$ computes canonical elements for and represents every non-trivial SCC of G.
(ii) $\leftrightarrow^+_{T_\infty} \cup id_V = \approx$.
(iii) For all $v, v' \in V$, v' is reachable from v in the graph $G' = (V, \rightarrow_{R_\infty}, \rightarrow_{S_\infty}, \rightarrow_{T_\infty})$ by a path in $\rightarrow^{\leq 1}_{T\infty} (\rightarrow^+_{R_\infty} \rightarrow^*_{S_\infty} \cup \rightarrow^+_{S_\infty}) \leftarrow^{\leq 1}_{T_\infty}$, iff $[v]_\approx \rightarrow^+_\approx [v']_\approx$ in $G/\approx$. Thereby $\rightarrow^{\leq 1}$ denotes that there is at most one such rewrite step.

Proof. (ad i) Let $v \approx v'$. Then v and v' are on an oriented k-cycle c_k for some $k \geq 2$ by lemma 8 (i). By lemma 8 (ii), this cycle contains a peak and by lemma 9 (i), DEDUCE constructs a $k-1$ cycle c_{k-1} which can again be oriented. Iteration leads to a sequence $c_k, c_{k-1}, \ldots, c_2$ of oriented i cycles. Every i-cycle is identical to the $i+1$-cycle, except that some path $v_k \rightarrow_S \rightarrow_R v_{k+2}$ on the $i+1$-cycle has been replaced by and edge $v_k \rightarrow_I v_{k+2}$. This iteration finishes after at most $|V|^2 - |E|$ steps. Now consider the oriented 2-cycle c_2. Assume it has the vertices v_0 and v_1 with $v_0 \prec v_1$. By construction, these two vertices are on all other cycles

c_1. We obtain the rewrite rules $v_0 \rightarrow_S v_1$ and $v_1 \rightarrow_R v_0$. Then SCC is applicable. The two rules can be replaced by $v_1 \rightarrow_T v_0$ and $v_0 \rightarrow_\Delta v_0$. By lemma 9 (ii) we can now use Collapse and Compose to simplify each oriented i-cycle, $3 \leq i \leq k$, to an $i-1$-cycle, which can again be oriented. This yields a new sequence of oriented cycles $c'_{k-1}, \ldots c'_2$, on which SCC and the simplification can be applied. After $k-2$ steps, the original sequence of cycles has been completely simplified. There is exactly one rewrite rule $v_i \rightarrow_T v_j$ for every vertex v_i of the original cycle, except the minimal vertex. Using Collapse, they can be transformed into $k-1$ T-rules pointing from each vertex (except the minimal one) to the minimal one. This can be done in $k-1$ steps, starting from the minimal element.

(ad ii) By (i), $\rightarrow_{T_\infty}$ computes canonical representants for every non-trivial equivalence class of $\approx$. Thus $\leftrightarrow_{T_\infty}$ connects exactly the elements in every non-trivial SCC. $\leftrightarrow_{T_\infty}$ is contained in $\approx$ and becomes identical to $\approx$, when self-edges for all vertices in V are added.

(ad iii) Let v' be reachable from v by a path in $\rightarrow^{\leq 1}_{T_\infty} (\rightarrow^+_{R_\infty} \rightarrow^*_{S_\infty} \cup \rightarrow^+_{S_\infty}) \leftarrow^{\leq 1}_{T_\infty}$. Then, by soundness of C_{SCC} (lemma 7), v' is reachable from v in G. Moreover, all vertices on the $\rightarrow^+_{R_\infty} \rightarrow^*_{S_\infty} \cup \rightarrow^+_{S_\infty}$ path are in T-normal form by lemma 4. Thus for every pair v_i and v_j on this paths there are $v'_i \in [v_i]$ and $v'_j \in [v_j]$ such that there is an edge $v'_i \rightarrow_{R \cup S} v'_j$, setting $v'_i = v_i$ and $v'_j = v_j$. Hence $[v_i]_\approx \rightarrow_\approx [v_j]_\approx$ and therefore $[v]_\approx \rightarrow^+_\approx [v']_\approx$ hold by definition of $\rightarrow_\approx$.

Conversely, let $[v]_\approx \rightarrow^+_\approx [v']_\approx$. Then there are elements in $[v]_\approx$, $[v']_\approx$ and all intermediate equivalence classes that yield an equivalent proof $v_0 \rightarrow^+ v_k$. By correctness of C_t (theorem 1), there is a rewrite proof from v_O to v_k. By (i), canonical representants of $[v]_\approx$ and $[v']_\approx$ (which contain v_0 and v_k) are T-reachable in one step. □

The statement in theorem 5 (iii) is somewhat more complex than one might expect. It does not say that G' *is* the component graph of G, but that G' behaves like G_{SCC} with respect to reachability. The reason is that the Deduce steps in C_{SCC} add new edges to the graph that may persist also in the final state. Thus the edges of G' will be in general different from the edges of G_{SCC}. Therefore additional work is required to recover G_{SCC} from G', but in many applications, reachability equivalence suffices. We will discuss this issue below.

The proof of correctness of C_{SCC} has the following implications.

Corollary 1. *Graph G contains a cycle, iff C_{SCC} generates a T-rule.*

Corollary 2. *C_{SCC} eliminates an isolated simple k-cycle in $O(k)$ steps.*

Proof. There are $k-2$ applications of Deduce, $k-1$ applications both of SCC and Collapse and $2k-3$ applications of Orient. So there are $5k-7$ steps in the algorithm. □

The proof of lemma 5 (ii) yields an upper bound for the running time of C_{SCC}.

Corollary 3. *All runs of C_{SCC} terminate after $O(|V|^2)$ steps.*

This bound is not optimal, since the standard SCC algorithms based on depth-first search have $O(|V| + |\to|)$ running time [3,10]. It is also very loose, since the number of steps in C_{SCC} depends strongly on the clustering of cycles in SCCs. C_{SCC} is a declarative, rule-based and non-deterministic state transition system. It therefore specifies a class of SCC algorithms rather than a particular instance. Efficient algorithms can be obtained from C_{SCC} by eliminating non-determinism via execution strategies, as we will see below. For running time analysis we assume—following arguments in [15] for similar procedures—that all implementations of rules in C_t and C_{SCC} can be executed in constant time.

The fact that C_{SCC} is declarative, rule-based and non-deterministic, makes the procedure also flexible enough to support various different applications, implementations and modifications. An obvious application are constraint systems in general and constraint-based analysis of program and finite and infinite state systems in particular. There, the reduction to a component graph may yield a more concise representation of the constraint system; the size of the constraint store or the state space may be drastically deduced. The fact that G' is not precisely G_{SCC} is not important here, since constraint solving requires computation of the transitive closure anyway. This can now be done using the search-based decision procedure of non-symmetric rewriting. SCC-algorithms are also useful for the acceptance analysis of Büchi automata, with interesting applications in model checking. One could therefore take the extreme point of using C_{SCC} for a constraint-based approach to model checking. Since C_t and C_q can be extended to some non-ground cases, our approach should be applicable to the analysis of infinite state systems.

8 Strategic Refinements of the Strongly Connected Component Algorithm

We now show that refinements via specification of execution strategies allow the rational reconstruction of efficient SCC algorithms. To this end, two further questions must be answered. How to translate the output graph G' of C_{SCC} into the component graph G_{SCC}? How to improve the running time of C_{SCC}?

A naive answer to the first question is as follows.

Lemma 10. *The following procedure yields the component graph G_{SCC} of a graph G.*

1. *Run C_{SCC} on G, but store G separately.*
2. *Run* SIMPLIFY *of C_{SCC} on G and T_∞.*

Proof. The procedure obviously reduces each vertex of G to its canonical representant in G_{SCC} and each edge of G to an edge of G_{SCC}, that is an edge that connects canonical representants. □

An answer to the second question requires suppressing DEDUCE steps that do not contribute to cycle elimination. We now present a strategy that restricts DEDUCE steps completely to cycles, thereby also answering the first question. Let C^s_{SCC} be the state transition system C_{SCC} together with the following strategy.

– Orient inequalities using depth-first search on I_0,
– Construct $\prec$ on vertices on-the-fly, using the discovery times of vertices.
– Give successor vertices a smaller weight with respect to $\prec$ as long as possible.

This strategy orients inequalities into R-rules as long as possible. We call *R_f-step* an R-step and *S_f-step* an S-step whose right-hand vertex is finished by depth-first search. We call *R_d-step* an R-step and *S_d-step* an S-step whose right-hand vertex is discovered but not finished by depth-first search.

Lemma 11. *C^s_{SCC} has the following properties.*

(i) Every S-step is either in S_f or S_d.
(iii) Every SCC contains a simple cycle with exactly one S_d-step.

Proof. The proofs use simple properties of depth-first search.
(ad i) Obvious.
(ad ii) Assume that we depth-first search on a SCC. By lemma 8 (i), the SCC contains a cycle. By lemma 8 (ii), this cycle contains at least one S-step. We can assume that the cycle is simple, since every cycle can be decomposed into simple cycles. Now assume that we have so far not detected any cycle. Let the S-step on the cycle be an S_f-step $v \rightarrow_{S_f} v'$. Then v is discovered, but not finished and v' is finished. Since we assume that we are on a cycle, some predecessor u from which v has been discovered must be reachable from v'. Since v' is finished, u must also be finished. But then v must also be finished, since it is reachable from u, a contradiction. Thus the S-step must be an S_d-step. Moreover, this S_d-step must be the step closing the cycle, since v' must at least have been discovered.□

We can therefore refine the strategy in C^s_{SCC} further by restricting DEDUCE step between an R-rule and an S-rule to R_d and S_d-steps. These DEDUCE steps must be eagerly applied as soon as an S_d-step has been discovered. Also the succeeding simplification steps must be eagerly applied. Vertices that have been discovered or finished must keep their labels under the simplifications. This guarantees that precisely those critical pairs are computed that collapse cycles. But all these critical pairs finally disappear in the T-rules that characterize the SCCs. Outside the SCCs, no new edge is added to the graph. Moreover the outside edges are simplified by T-rules and finally become edges of the component graph. This argument implies the following result.

Theorem 6. *Every fair implementation of C^s_{SCC} is correct. For every input graph G, the algorithm gives out the component graph G_{SCC}. The SCCs of G are represented by T_∞. The edges of G_{SCC} are represented by R_∞ and S_∞.*

Theorem 6 has the following immediate consequence.

Corollary 4. *Cycle detection with C^s_{SCC} means detection of S_d-steps. It is possible with* ORIENT *steps only, that is by pure depth-first search.*

We have thus reconstructed the standard algorithm for cycle detection in digraphs, using solely a strategy that refines a purely non-deterministic declarative specification. Thereby the main work in the algorithm is shifted from the

KB-procedure almost entirely to the strategy. As a consequence, the running time of cycle detection is determined by that of depth-first search.

Also C^s_{SCC} can be further refined.

Lemma 12. *For C^s_{SCC}, every T-tree for a simple k-cycle has k branches, that is one T-rule per vertex to the minimal one (except from the latter itself).*

By lemma 11 (iii), every simple cycle in a SCC contains exactly one S-step. The particular T-rules are computed backwards from the smallest node to the entry vertex of the SCC. One can therefore collect the vertices of the T-tree as follows: Search backwards (using depth-first search) all those vertices that are discovered, but not finished, starting with right-hand sides of S-rules and stopping at their left-hand sides. This saves some intermediate DEDUCE steps and yields a variant of the standard strongly connected component algorithm [3,10]. Again, most of the work has been shifted from the KB-procedure to the strategy. The running time is then of course again determined by that of depth-first search.

In opposition to the standard algorithm, however, which performs two global searches on the input-graph, C^s_{SCC} eliminates cycles locally one by one. This may be an advantage in practice. Moreover, the construction of the whole component graph is included in C^s_{SCC}, whereas this is not the case for the standard algorithm. As a further benefit from the locality and non-determinism of C_{SCC} and C^s_{SCC}, edges can easily be added to a graph at run time.

In the control of centralized reactive systems like operation or database systems, vertices of a graph represent tasks and edges the relation of waiting that some resource is released. Cycles correspond to deadlocks [19]. With C^s_{SCC}, dynamic cycle detection is quite simple, since, by lemma 11 (iii) and corollary 4, it suffices to check for cycles, when the new edge is in S. We now determine, whether an S-step is in S_d or S_f by recording the discovery and finishing times of all vertices. If it is in S_d (that is, when the finishing time of its right-hand vertex is greater than the discovery time of its left-hand vertex), it must be closing a cycle. If it is in S_f (that is, when the finishing time of its right-hand vertex is smaller than the discovery time of its left-hand vertex), it could also be the case that two paths are joined that do not form a cycle. One can then use the depth-first search to test for cyclicity along a rewrite path. Note that we cannot use the search procedure of non-symmetric rewriting, since we did not compute critical pairs involving S_f. This reconstructs the best known cycle detection technique in [19]. With C_{SCC}, the search procedure of non-symmetric rewriting allows cycle detection. This may be preferable for constraint graphs. Here, one can easily implement (incomplete) heuristics, for instance only search for cycles up to some fixed size. Of course, besides dynamic cycle detection, C_{SCC} also allows the dynamic and on-line construction of component graphs.

We now discuss further variants of C_{SCC}. First, the procedure supports approximations. 2-cycles, for instance, can be eliminated with C_{SCC}, prohibiting DEDUCE and using SCC solely. Also, successive applications of DEDUCE can be triggered by strategies that detect all cycles up to a fixed size.

Second, the running time of C_{SCC} can further be improved by disregarding equational COLLAPSE steps. A representation of SCCs without COLLAPSE corresponds to a disjoint-set forest in a union-find data structure [10]. The effect of COMPOSE is precisely path-compression. Modeling the other heuristics of union-find, namely union by rank, is more involved. Choosing the representant of the bigger partial equivalence class as a new representant when merging two partial classes may violate the ordering constraints. The solution is to introduce a rank function and assign a weight $(r(c), o(c))$ to each vertex c, where $r(c)$, $r : C \to \mathbb{N}$, is the rank of c and $o(c)$ its weight according to the usual precedence. The components of this weight function are compared lexicographically. $r(c)$ is initialized with 0 and incremented, whenever it becomes the smaller vertex of a new T-rule. We use the modified variant

$$\frac{(S, \Delta, P \cup \{l \to_R r, r \to_S l\})}{(S, \Delta \cup \{r \to_\Delta r\}, P \cup \{l \to_E r\})}$$

and the following variant of ORIENT for E-rules

$$\frac{(S \cup \{s \to_E t\}, \Delta, P)}{(S, \Delta, P \cup \{s \to_T t)},$$

if $r(s) \leq r(t)$. Then $r(t)$ is incremented by one. Otherwise, we use $t \to_T s$ and increment $r(s)$ by one. If this modification is used together with eager application of SIMPLIFY, then we can guarantee that only the canonical representants of equivalence classes are compared. It is well-known that $r(s)$ approximates the logarithm of size of the T-subtree rooted at s and is also an upper bound for the height of each vertex in the T-tree. Therefore for every vertex there are only logarithmically many smaller vertices [10]. This fact has considerable impact on the running time of C_{SCC} and associated search procedures.

Third, C_{SCC} and its variants can easily be adapted to quasiorderings. The elimination of SCCs then means the construction of the associated partial ordering from a given quasiordering. Now, of course, the information stored in Δ is redundant and SCC can be simplified to

$$\frac{(S \cup \{l \to_R r, r \to_S l\}, P)}{(S \cup \{l \to_T r\}, P)}.$$

Fourth, C_{SCC} and its variants can be extended to vertices with internal structure including variables. Without monotonicity, unification has to be used, maybe modulo some axioms, like associativity, monotonicity or idempotence. This is in particular interesting for the analysis of state transition systems. With monotonicity, one should in the ground case use the variant

$$\frac{(S \cup \{s[l] \to_R s[r], r \to_S l\}, \Delta, P)}{(S \cup \{s[l] \to_T s[r], r \to_R l\}, \Delta \cup \{s[r] \to_\Delta s[r]\}, P)}$$

of (SCC). By lemma 5, these extensions might not terminate, unless additional coding is used. In the non-ground case monotonic case, the aforementioned notorious difficulties with variable critical pairs arise.

9 The Strongly Connected Component Algorithm at Work

For further illustration of our previous arguments, we consider the following example. We eliminate the simple isolated oriented 4-cycle in the first diagram of figure 1. We use the precedence defined by $c_i \succ c_j$ iff $i < j$. Obviously, there

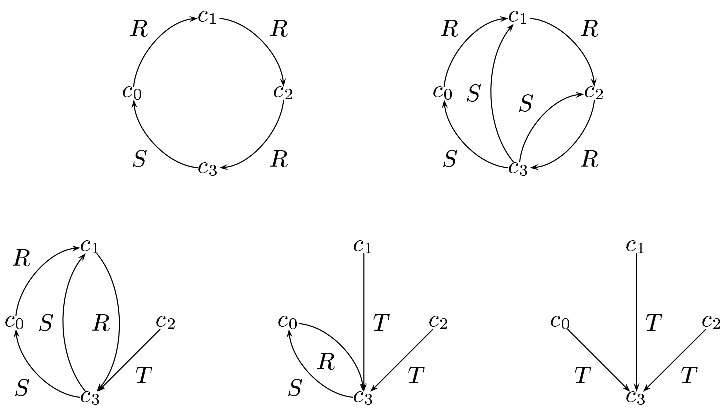

Figure1. Eliminating a simple cycle

is the critical pair $c_3 \rightarrow_S \rightarrow_R c_1$, for which Deduce and Orient compute the new edge $c_3 \rightarrow_S c_1$. This yields the new 3-cycle $c_1 \rightarrow_R c_2 \rightarrow_R c_3 \rightarrow_S c_1$ and the critical pair $c_3 \rightarrow_S \rightarrow_R c_2$. Deduce and Orient compute the new edge $c_3 \rightarrow_S c_2$. This yields the new 2-cycle $c_2 \rightarrow_R c_3 \rightarrow_S c_2$. This situation is shown in the second diagram of figure 1.

Now SCC comes into play. It adds the edges $c_2 \rightarrow_E c_3$ (which is turned into $c_2 \rightarrow_T c_3$ by Orient) and $c_3 \rightarrow_\Delta c_3$ and discards $c_2 \rightarrow_R c_3$ and $c_3 \rightarrow_S c_2$. Compose then replaces $c_1 \rightarrow_R c_2$ by $c_1 \rightarrow_R c_3$. All k-cycles are now collapsed into $k-1$-cycles. In particular, there is the new 2-cycle $c_1 \rightarrow_R c_3 \rightarrow_S c_1$. This situation is shown in the third diagram of figure 1. After a mere repetition of the previous graph transformation leading to the fourth diagram of figure 1, there is a final 2-cycle $c_0 \rightarrow_R c_3 \rightarrow_S c_0$. After its elimination by SCC, c_3 is the canonical representant of the cycle; the equivalence class is completely described by T.

Let us add the edge $c_4 \rightarrow_S c_1$ to the cycle. This situation is shown in the first diagram of figure 2. Then Deduce and Orient also add $c_4 \rightarrow_S c_2$ and $c_4 \rightarrow_S c_3$ to the graph. This situation is shown in the second diagram of figure 2. This critical pair computation is unnecessary for cycle elimination. $c_4 \rightarrow_S c_1$ and $c_4 \rightarrow_S c_2$ are replaced by $c_4 \rightarrow_S c_3$—an edge on the equivalence class—in

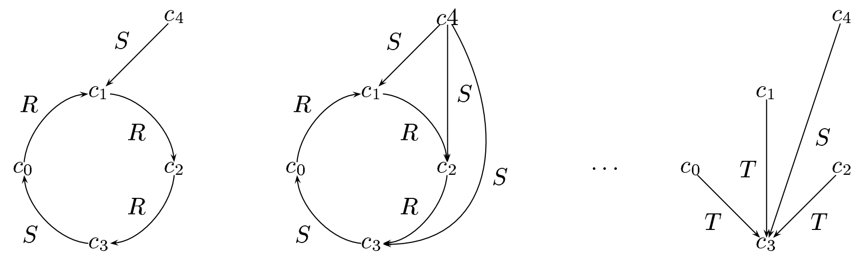

Figure2. Performing Unnecessary DEDUCE-Steps

the final graph, using COLLAPSE. This situation is shown in the third diagram of figure 2.

Let us now add the edge $c_1 \to c_5$ to the graph. This situation is shown in the first diagram of figure 3. Then DEDUCE and ORIENT also add $c_4 \to_R c_5$ to

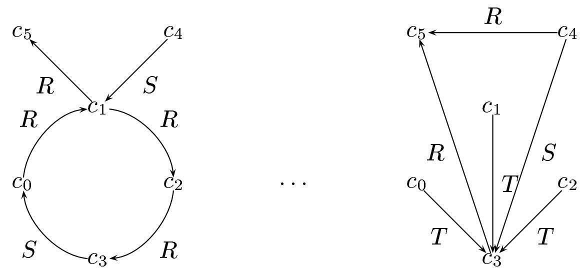

Figure3. A New Edge in the Component Graph

the graph. It does not belong to the component graph. This situation is shown in the second diagram of figure 3. Note that it does not matter at which stage of the process we a new edge to the graph.

Now assume that we orient the nodes of the graph on the fly with C^s_{SCC}. Assume that we start with c_0 and c_1, but then continue with c_5. Then, when we continue with c_2, c_5 is already finished, hence need not be considered by DEDUCE. We thereby disregard all critical pairs that are unnecessary for cycle elimination. All nodes on the cycle are discovered, but not finished and all rules on the cycle are R_d- and S_d-rules. We have detected the cycle (by mere depth-first search) as soon as we have oriented $c_3 \to_I c_1$ into S_d. For collecting all nodes on the cycle into one equivalence class, it suffices to depth-first search backwards from c_0 all the discovered, but non-finished nodes, until we return to c_1, the right-hand side of the S-rule. This yields the component graph via the traditional SCC algorithm by strategic refinement from C_{SCC}.

10 Conclusion

We have specified a combined Knuth-Bendix completion procedure for inequalities and equalities. The completion constructs inequational rewrite paths modulo the equivalence classes defined by the equations. Main applications are therefore graph traversals modulo equivalence relations defined on vertices or edges. By its non-deterministic and rule-based nature, Knuth-Bendix completion can be used as a metaprocedure for the development of declarative procedures and algorithms for graphs and constraint systems.

Here, we have presented two examples for such a development. The integration of memoization into constraint graphs and the construction of cycle detection and strongly connected component algorithms. Both procedures are declarative and generic. Efficiency can be easily integrated by refining with strategies. There is much space for approximations and other heuristics. In opposition to the standard algorithms, our algorithms immediately apply to dynamically changing environments. The approach should be flexible enough to cover other examples.

Knuth-Bendix completion not only supports mathematical program construction from general principles of universal algebra. It also provides generic data structures and implementation techniques that can be optimized once and for all. At least for prototyping, they can be completely hidden to the programmer. A rewrite-based environment for rule or strategy specification, like ELAN [1] or MAUDE [2], should be used instead.

In the present text, we could only superficially treat complexity issues. For C_{SCC}, in particular, we gave only rough bounds. In practice, the complexity strongly depends on the clustering of cyclic structures in a graph and thereby on its sparsity. One can expect that in a sparse graph the number of simple cycles as a function of the cycle size looks like a Gaussian curve with a maximum for medium cycles. Moreover there are only few cycles that have more than a few vertices in common. Therefore approximation heuristics that disregard very big cycles should yield acceptable results in practice. The overall complexity of cycle elimination with C^{s}_{SCC} should roughly be the sum of the complexities of simple cycles. For C_{SCC} an important factor for the running time is the expected number of DEDUCE steps. We leave a detailed probabilistic analysis with random graphs to future work. In the end, practical experiments are the ultimate measure for applicability of these procedures and algorithms.

References

1. http://www.loria.fr/equipes/protheo/SOFTWARES/ELAN.
2. http://www.maude.csl.sri.com.

3. A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.
4. L. Bachmair and H. Ganzinger. Rewrite techniques for transitive relations. In *Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 384–393. IEEE Computer Society Press, 1994.
5. L. Bachmair and D. Plaisted. Termination orderings for associative-commutative rewriting systems. *J. Symbolic Computation*, 1(4):329–349, 1985.
6. L. Bachmair, C. R. Ramakrishnan, I. V. Ramakrishnan, and A. Tiwari. Mormalization via rewrite closures. In P. Narendran and M. Rusinowitch, editors, *Rewriting Techniques and Applications, 10th International Conference, RTA-99*, volume 1631 of *LNCS*, pages 190–204. Springer-Verlag, 1999.
7. S. Burris. Polynomial time uniform word problems. *Math. Logic Quarterly*, 41:173–182, 1995.
8. I. Cervesato, N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In R. Gorrieri, editor, *12th IEEE Computer Security Foundations Workshop, CSFW'99*, pages 55–69. IEEE Computer Society Press, 1999.
9. H. Comon. Completion of rewrite systems with membership constraints. In *Int. Coll. on Automata, Languages and Programming, ICALP'92*, volume 632 of *LNCS*. Springer-Verlag, 1992.
10. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.
11. C. Delor and L. Puel. Extension of the associative path ordering to a chain of associative-commutative symbols. In C. Kirchner, editor, *5th Conference on Rewrite Techniques and Applications*, volume 690 of *LNCS*, pages 389–404. Springer-Verlag, 1993.
12. G. Denker and J. Millen. The CAPSL integrated protocol environment. Technical Report SRI-CSL-2000-02, SRI International, 2000.
13. Baader F. and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.
14. F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and verifying security protocols. In M. Parigot and A. Voronkov, editors, *7th International Conference on Logic for Programming and Automated Reasoning (LPAR 2000)*, volume 1955 of *LNCS*, pages 131–160. Springer-Verlag, 2000.
15. D. Kapur. Shostak's congruence closure as completion. In H. Comon, editor, *Rewrite Techniques and Applications, 8th International Conference, RTA-97*, volume 1232 of *LNCS*, pages 23–37. Springer-Verlag, 1997.
16. D. Kozen. Complexity of finitely presented algebras. Technical Report TR-76-294, Department of Computer Science, Cornell University, 1979.
17. J. Levy and J. Agustí. Bi-rewrite systems. *J. Symbolic Comput.*, 22:279–314, 1996.
18. C. Marché. Normalised rewriting: an alternative to rewriting modulo a set of equations. *J. Symbolic Computation*, 21:253–288, 1996.
19. O. Shmueli. Dynamic cycle detection. *Information Processing Letters*, 17(4):185–188, 1983.
20. G. Struth. *Canonical Transformations in Algebra, Universal Algebra and Logic.* PhD thesis, Institut für Informatik, Universität des Saarlandes, 1998.
21. G. Struth. An algebra of resolution. In L. Bachmair, editor, *Rewriting Techniques and Applications, 11th International Conference*, volume 1833 of *LNCS*, pages 214–228. Springer-Verlag, 2000.
22. G. Struth. Deriving focused calculi for transitive relations. In A. Middeldorp, editor, *Rewriting Techniques and Applications, 12th International Conference*, volume 2051 of *LNCS*, pages 291–305. Springer-Verlag, 2001.

23. G. Struth. Knuth-Bendix completion for non-symmetric transitive relations. In M. van den Brand and R. Verma, editors, *Second International Workshop on Rule-Based Programming (RULE2001)*, volume 59 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2001.
24. G. Struth. Termination of ground non-symmetric Knuth-Bendix completion. Technical Report 2002-9, Institut für Informatik, Universität Augsburg, 2002.