SOUR Graphs for Efficient Completion

Christopher Lynch[†]& Polina Strogova[‡]

We introduce a data structure called SOUR Graphs and present an efficient Knuth-Bendix completion procedure based on it. SOUR Graphs allow for a maximal structure sharing of terms in rewriting systems. The term representation is a dag representation, except that edges are labelled with equational constraints and variable renamings. The rewrite rules correspond to rewrite edges, the unification problems to unification edges. The Critical Pair and Simplification inferences are recognized as patterns in the graph and are performed as local graph transformations. Our algorithm avoids duplicating term structure while performing inferences, which causes exponential behavior in the standard procedure. This approach gives a basis to design other completion algorithms, such as goal-oriented completion, concurrent completion and group completion procedures.

1 Introduction

Knuth Bendix Completion [KB70] is an efficient procedure for solving the word problem and equational unification problem. The objective of the procedure is to take a set of equations and convert it into an equivalent set where word problems and equational unification problems can be solved by applying equations in an ordered fashion. Completion does not always halt. But when it does, the word problem becomes decidable, since it is only necessary to reduce equations into a normal form and compare normal forms to solve the problem. In Completion, we define an ordering and unify a maximal side s of a renaming of one equation $s \approx t$ with a subterm of a maximal side of another subterm. Then we replace the subterm with t and apply a constraint representing the unification problem.

Knuth Bendix Completion is not as efficient as it could be. Many terms appearing in different places have the same ancestor. So it would be beneficial if one inference applied to all the occurrences at the same time. For instance. Suppose we have equations $f(a) = b$ and $a = c$. Completion unifies a in $f(a)$ with a in $a = c$, and then replaces a in $f(a)$ with c. The result is a new equation $f(c) = b$. This new equation is made up of pieces of the two old equations. b appears in both $f(a) = b$ and $f(c) = b$, etc. Further inferences spread these pieces around more and more. Note that every equation that is created is made up of pieces of the initial equations. If it was possible to keep track of all pieces of the same ancestor, an inference procedure could perform many inferences at the same time. In this paper, we introduce a new graph data structure called a SOUR graph, which allows us to do that. When terms contain variables, each inference involves a renaming and a constraint. It is still true that all created equations are formed from pieces of the initial ones, but some of the equations have renamings and constraints applied to them.

[†]Clarkson University, Department of Mathematics and Computer Science, Box 5815, Potsdam, NY, 13699-5815, USA, clynch@sun.mcs.clarkson.edu. (Note: Most of the work was done while the first author was visiting the PROTHEO group at INRIA Lorraine and CRIN.)

[‡]INRIA Lorraine and CRIN, Campus scientifique, BP 101, 54602 Villers-lès-Nancy Cedex, France, strogova@loria.fr.

1365–8050 © Chapman & Hall

We use the Basic Completion [BGLS95, NR92] inference rules. The idea of Basic Completion is that whenever we perform an inference, instead of applying the most general unifier to the conclusion of the inference, we save the unification problem as an equational constraint. This is the first idea necessary for the SOUR graph philosophy. When unifiers are applied new pieces are created. But with equational constraints, old pieces are just combined together in new ways and constrained.

Our work is based on [Lyn95b], where it was shown how to perform theorem proving, using a graph data structure. This is where it was first shown how new objects are created from existing pieces. When an inference is performed, we just add new relationships between the existing pieces. Instead of combining constraints as in Basic Completion, we keep them separate and associate them with the new relationships that are created. We also associate renamings created by an inference with these new relationships.

In this paper we instantiate the ideas from [Lyn95b] to create a new data structure for the Completion Procedure. A dag representation of a set of equations is created. Nodes in the graph initially represent subterms of the initial problem. However the completion procedure will make the nodes represent new terms. The dag representation has edges called *subterm edges* from a node representing a term to the nodes representing each of its subterms. It also has edges called *rewrite edges* between the nodes representing the terms on either side of an equation. We can also add a *unification edge* between two nodes representing unifiable terms, and an *orientation edge* from a node representing a term to a node representing a smaller term. We call the graphs SOUR graphs, for **S**ubterm, **O**rientation, **U**nification and **R**ewrite edges.

SOUR graphs are closer to an implementation than the Paramodulation without Duplication of [Lyn95b]. Inferences are performed by graph transformations. We look for certain patterns in the graph. Each pattern causes a new edge to be added to the graph, and an old edge to be possibly deleted. There are two kinds of patterns. An SUR pattern is a sequence of edges representing that a subterm of a term unifies with one side of an equation. In that case we can add a new equation where that subterm is replaced by the other side of the equation. We only need to add a new subterm edge to the graph to represent the new term, labelled with a unification constraint and a renaming associated with the inference. We also have RUR patterns, representing that two sides of equations unify. So we add a new rewrite edge to the graph, representing an equation between the other two sides of the equations. The completion procedure on SOUR graphs is then a series of graph transformations. The most important point is to know which constraints and renamings to label the new edges with.

The paper is organized as follows. In the next section we give the preliminaries. We follow that by giving some examples of the completion procedure on SOUR graphs and a definition of the syntax and the semantics of the SOUR graph. Then we give the graph transformations, which determine the inference procedure. After that, we prove the soundness and completeness of the system. We give some experimental results for an implementation of SOUR graph completion. We also show how this technique has led to the solution of other open problems dealing with completion. Finally, in the conclusion, we show the relationship of our work with other work.

2 Definitions

We use the graph formalism to express rewriting systems. A Graph $G = (V, E)$ is the pair composed of the set of *vertices* or *nodes* V and the set of *edges* E. Each edge $e \in E$ is associated with a pair of vertices by a surjective function $pair : E \to V \times V$. *Directed* edges are ordered, and *undirected* edges are not.

For each *directed* edge e the first component of $pair(e)$ denoted by $init(e)$ is called the *initial vertex* of e, the second component denoted by $fin(e)$ is called the *final vertex* of e. We say that e *goes from* $init(e)$

to $fin(e)$. If e is an undirected edge then $pair(e)$ can be viewed as a multiset $\{init(e), fin(e)\}$.

Let $\mathcal{F}$ be an enumerable set of function symbols and $\mathcal{X}$ be a set of variables. Let $arity$ be a function from $\mathcal{F}$ to the set of non-negative integers. Then the set of *terms* $\mathcal{T} = \mathcal{T}(\mathcal{F}, \mathcal{X})$ is the smaller set such that

- $\mathcal{X} \subset \mathcal{T}$,
- $\forall f \in \mathcal{F},\ arity(f) = n\ :\ f(t_1, \dots t_n) \in \mathcal{T}$ if $t_1, \dots t_n \in \mathcal{T}$.

Terms without variables are called *ground*. A *position* p is a sequence of integers. We define *the subterm* $t|_p$ of term t *at position* p such that

- $t|_\varepsilon = t$, where ε is the empty sequence,
- if $t = f(t_1, \dots t_n),\ p = i\ q$ for $1 \leq i \leq n$, then $t|_p = t_i|_q$.

We write $s[t]_p$ to indicate that s is a term containing t at position p, i.e., $s|_p = t$. After using that notation, we will write $s[t']_p$ to indicate the term obtained by replacement of t by t' in s at position p.

A *substitution* σ is a mapping from $\mathcal{X}$ to $\mathcal{T}$, given in postfix notation, which is the identity everywhere except on a finite number of variables. This mapping can be homomorphically extended to a mapping from $\mathcal{T}$ to $\mathcal{T}$. The set $Ran(\sigma) = \{x\sigma | x\sigma \neq x,\ x \in \mathcal{X}\} \subset \mathcal{T}$ is called the *range* of the substitution σ. A *composition* $\sigma\theta$ of substitutions σ and θ is the substitution such that $t\sigma\theta = (t\sigma)\theta$ for each $t \in \mathcal{T}$. A substitution σ is *more general* than a substitution θ if there exists a substitution τ such that $\sigma\tau = \theta$. A *renaming substitution* ρ is an injective substitution from $\mathcal{X}$ to $\mathcal{X}$. A renaming substitution ρ is *fresh* if the variables from the $Ran(\sigma)$ have not appeared somewhere else. We denote by id the substitution such that $Ran(id) = \emptyset$. A *ground substitution* is a substitution σ such that $Ran(\sigma)$ is a set of ground terms. If t is a term then $Ground(t)$ is the set of all terms $t\sigma$ such that σ is a ground substitution. If T is a set of terms, then $Ground(T)$ is the set of all terms t' such that t' is in $Ground(t)$ for some t in T.

An *equation* on $\mathcal{T}$ is an expression $s \approx t$, where $s, t \in \mathcal{T}$. The symbol $\approx$ is a binary predicate symbol, represented in infix notation. Let Eq be a set of equations. The *congruence closure* Eq^* of Eq is the smallest set such that $Eq^* \supset Eq$, Eq^* is reflexive, transitive, symmetric and $f(s_1, \dots, s_n) \approx f(t_1, \dots t_n)\ \in Eq^*$ whenever $arity(f) = n$ and $s_i \approx t_i\ \in Eq^*\ 1 \leq i \leq n$. We write $Eq \models\ s \approx t$ iff $s\sigma \approx t\sigma\ \in (Ground(Eq))^*$ for all ground σ.

Let $\doteq$ be a binary infix predicate. The expression $s \doteq t$, where $s, t \in \mathcal{T}$ is called a *unification problem* on $\mathcal{T}$. An *(unification) equational constraint* φ is a conjunction $s_1 \doteq t_1 \wedge \dots \wedge s_n \doteq t_n$. If $n = 0$ we write $\varphi = \top$. A pair $s[\![\, \varphi \,]\!]$ composed of a term s and an equational constraint φ is called a *constrained term*. A pair $s \approx t[\![\, \varphi \,]\!]$ composed of an equality $s \approx t$ and an equational constraint φ is called a *constrained equation*. A set of constrained equations is a *constrained equational system*.

A substitution σ is a *solution*

- of a unification problem $s \doteq t$, denoted by $\sigma \in Sol(s \doteq t)$, if $s\sigma = t\sigma$. We also say that σ is a *unifier* of s and t;
- of a constraint $s_1 \doteq t_1 \wedge \dots \wedge s_n \doteq t_n$ if $\sigma \in Sol(s_i \doteq t_i)\ 1 \leq i \leq n$.

σ is a *unifier* of two constrained terms $s[\![\, \varphi \,]\!]$ and $t[\![\, \psi \,]\!]$ if it is a solution of $s \doteq t \wedge \varphi \wedge \psi$. σ is a *most general unifier (solution)*, written mgu (resp. mgs), if σ is a unifier (resp. solution) and for each unifier (resp. solution) θ we have that σ is more general than θ. A constraint is *satisfiable* if it has a solution. An mgu and an mgs is unique up to composition with a variable renaming.

It is quite important to understand our use of renamings in this paper. To begin with, there are individual renamings. We will always represent these using the Greek letter τ or ρ. It is helpful to think of these as syntactic objects, such that $x\tau_i$ is the same variable as $x\tau_j$ if and only if $i = j$, and $x\tau_i$ is never the same variable as $y\tau_j$ if $x \neq y$. In general, a renaming is a list of individual renamings. We always represent these using the Greek letter η. We also think of these as syntactic objects. In other words, $x\eta_i$ is the same variable as $x\eta_j$ if and only if η_i and η_j are the same list of individual constraints, and $x\eta_i$ is never the same variable as $y\eta_j$ if $x \neq y$. When we talk about renamings, one should visualize such a list of individual renamings.

A term $t\sigma$ is an *instance* of a constrained term $t[\![\,\varphi\,]\!]$ if $\sigma \in Sol(\varphi)$. An equation $s\sigma \approx t\sigma$ is an *instance* of a constrained equation $s \approx t[\![\,\varphi\,]\!]$ if $\sigma \in Sol(\varphi)$.

A constrained term $s[\![\,\varphi\,]\!]$ *basic matches* a constrained term $t[\![\,\psi\,]\!]$ if there is a renaming ρ such that $s\rho = t$ and $mgs(\varphi)\rho = mgs(\psi)$.

An irreflexive antisymmetric transitive infix binary relation on $\mathcal{T}$ is called an *ordering*. An ordering $>$ is *total* if for every pair of distinct terms s and t, either $s > t$, or $t > s$. The ordering $>$ is *well founded* if there is no infinite decreasing sequence $t_1 > t_2 > \ldots > t_i > \ldots$ in $\mathcal{T}$. It is *monotonic with respect to substitution* if for all substitutions σ and terms s and t, $s > t$ implies $s\sigma > t\sigma$. The ordering $>$ is *monotonic with respect to context* if for all terms s and t and contexts u, $s > t$ implies $u[s]_p > u[t]_p$ for every position p. An ordering is a *reduction ordering* if it is well-founded, and monotonic with respect to substitution and context. In this paper we assume $>$ is a reduction ordering total on ground terms. We compare equations by considering an equation $s \approx t$ as the multiset $\{s, t\}$ and identifying $<$ with its multiset extension. For instance, if s_1, t_1, s_2, t_2 are ground terms, such that $s > t$ and $u > v$, then $(s \approx t) > (u \approx v)$ if and only if (i) $s > u$ or (ii) $s = u$ and $t > v$. In the non-ground case, if C_1 and C_2 are equations then $C_1 > C_2$ if and only if $C_1\theta > C_2\theta$ for all θ.

If $u[s']$ is a term, and Eq is a set of equations, we write $u[s'] \to u[t\sigma]$ and say that $u[s']$ *rewrites in one step* to $u[t\sigma]$ if there is an equation $s \approx t \in Eq$ and a substitution σ such that $s\sigma > t\sigma$, and $s' = s\sigma$. We write $u_0 \to^* u_n$ and say that u_0 *rewrites* to u_n if there is a a set of terms $\{u_1, \cdots, u_{n-1}\}$ such that for all i, $1 \leq i \leq n$, $u_{i-1} \to u_i$. A set of equations Eq is *canonical* if $>$ is a well-founded ordering and for every term s, t and u, if s rewrites to t and s rewrites to u, then there is a term v such that t rewrites to v and u rewrites to v.

The inference rule that we are imitating with SOUR Graphs is the *basic critical pair* inference rule.

Basic Critical Pair

$$\frac{(s \approx t[\![\,\theta_1\,]\!])\tau \qquad u[s'] \approx v[\![\,\theta_2\,]\!]}{u[t\tau] \approx v[\![\,s\tau \doteq s' \,\wedge\, \theta_1\tau \,\wedge\, \theta_2\,]\!]}$$

where s' is not a variable, τ is a fresh renaming substitution, and there is no $\sigma \in Sol(s\tau \doteq s' \wedge \theta_1\tau \wedge \theta_2)$ such that $s\sigma < t\sigma$ or $u[s']\sigma < v\sigma$.

This inference is called *basic* because instead of applying the substitution, it is saved as a constraint and inherited in future inferences. Standard presentations assume that at least one of the premises is renamed before the inference is performed. We assume that the left premise is renamed, and we explicitly show the renaming in the inference rule. The reason we have chosen to give the renaming explicitly, is because it helps to understand what is happening in the SOUR Graphs. We call the left premise a *from equation*, and the right premise is called an *into equation*. In certain cases, the from equation simplifies all instances of the into equation, and the inference is called a *simplification*. We give known completeness results for this inference rule in section 6.

3 Examples

In this section, we give an informal description of a SOUR graph and its associated inference rules, to give an idea of how they work. In the next section, we give a more formal presentation.

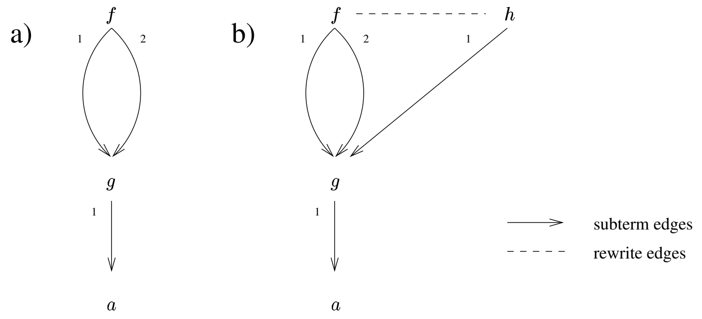

Fig. 1: SOUR representation of a) $f(g(a), g(a))$ and b) $f(g(a), g(a)) \approx h(g(a))$

Every term can be represented with a dag representation. For instance, Figure 1a is the dag representation of the term $f(g(a), g(a))$. As usual in dag representations, the same tree is used to represent like subterms, i.e., $g(a)$ is the same tree both times it appears. The edges used in the dag representation of a term are called *subterm edges*, and are labelled by the index number. They are considered to be directed from a term to its subterm. Equations are represented by a *rewrite edge* between the dag representing the term on the left side of the equation to the dag representing the term on the right side of the equation. Rewrite edges are undirected. [§] See Figure 1b for the representation of $f(g(a), g(a)) \approx h(g(a))$. Like subterms on different sides of the equations are represented by the same tree. Like terms in different rewrite rules are also represented by the same tree. In this fashion, a set of equations is represented by a SOUR graph with subterm edges and rewrite edges.

Necessary inferences are found by detecting patterns in the graph, and performed by adding (and sometimes removing) an edge to (from) the graph. For instance consider the equations $f(a) \approx b$ and $a \approx c$ as in Figure 2a. There is a critical pair inference between the two rules, because the subterm a of $f(a)$ is identical to the left hand side of the equation $a \approx c$. To perform the inference, we must create the new rule $f(c) \approx b$ by adding a new rule identical to $f(a) \approx b$, except that the subterm a has been replaced by c. To detect this inference in the SOUR graph, we need to detect that a is a subterm of $f(a)$, and a is also the left hand side of an equation. In other words, to detect an inference, we notice that there is a node in the graph which has an incoming subterm edge and an outgoing rewrite edge. To perform the inference, we

[§] It is possible to present SOUR graphs with directed edges representing rewrite rules, but that creates more complexity. We have chosen this format, because it allows a simpler representation.

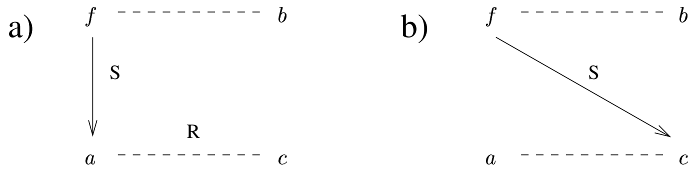

Fig. 2: Inference between $f(a) \approx b$ and $a \approx c$

must add a new subterm edge from the source of that subterm edge to the target of that rewrite edge. Furthermore, since this is a simplification, we may remove the subterm edge from $f(a)$ to a (see Figure 2b). In the ground case, every critical pair is a simplification. Since we need to find a pattern containing a Subterm and a Rewrite edge to perform the inference, we call the pattern an SR configuration.

We just gave an example of a ground inference. What happens in the non-ground case is more complicated. Consider the equations $f(g(x)) \approx h(x)$ and $g(k(x)) \approx k(x)$ as in Figure 3a. To perform a critical pair, we could apply a renaming ρ to $g(k(x)) \approx k(x)$. We get $g(k(x\rho)) \approx k(x\rho)$, and we can perform the following inference:

$$\frac{g(k(x\rho)) \approx k(x\rho) \qquad f(g(x)) \approx h(x)}{f(k(x\rho)) \approx h(x) \,[\![\, g(k(x\rho)) \doteq g(x) \,]\!]}$$

This inference has the same structure as the ground inference, except that

- we unify the subterm with the left hand side of an equation, instead of just finding identical terms,
- we must rename the left premise before performing the inference, and
- the conclusion is constrained by the unification problem

These three points make a non-ground inference different from a ground inference. Let's see how that affects what happens in a SOUR graph. The first point means that in order to detect an inference we must find a subterm of a term that is unifiable with the renamed version of the left hand side of a rule. In the SOUR Graph, we will have an undirected *unification edge* between two terms that are unifiable when renamed. So, in the example, there will be a unification edge between $g(x)$ and $g(k(x))$ (see Figure 3a). Therefore, the pattern we must detect is an SUR configuration. There must be nodes v_1, v_2, v_3, and v_4 in the graph, such that there is a Subterm edge from v_1 to v_2, a Unification edge between v_2 and v_3, and a Rewrite edge from v_3 to v_4. Then we need to add a new equation which is the same as the old one except that the term represented by v_2 is replaced by the term represented by v_4. Therefore, we add a new subterm edge from v_1 to v_4 (see Figure 3). This is analogous to what we did in the ground situation. The subterm edge from v_1 to v_2 may be removed only if this inference is a simplification. But this is not enough. We must also handle the second and third point mentioned above. The new edge that is added must be labelled with a renaming and a constraint as given in the inference. In the example presented in Figure 3b, we label the new edge with the constraint $g(k(x\rho)) \doteq g(x)$ and the renaming ρ. This means that any term using that subterm edge must rename everything underneath that edge by ρ, and must apply the constraint given in the edge to the whole term. See the next section for the formal definition. Notice that the ground case is just a special case of this. If there are unification edges between every node and itself, then in the ground case we also have SUR configurations representing SR inferences. Renamings and constraints do not affect the ground case. Renamings apply only to variables, and the only constraints needed in the ground case are those that evaluate to true.

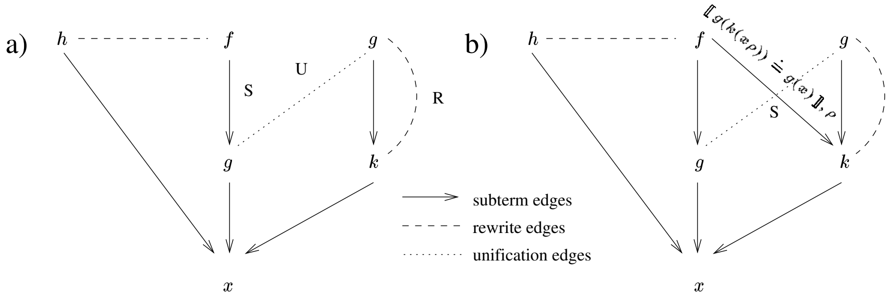

Fig. 3: Inference between $f(g(x)) \approx h(x)$ and $g(k(x)) \approx k(x)$

Below we explain what happens when we perform an inference at the root of one side of an equation. Inferences at the root are represented by a different kind of configuration. Consider rules $f(x) \approx g(x)$ and $f(a) \approx h(b)$ as in Figure 4a. We could perform the inference:

$$\frac{f(a) \approx h(b) \qquad f(x) \approx g(x)}{f(x) \approx h(b) \llbracket\, f(a) \doteq f(x) \,\rrbracket}$$

This inference is detected by noticing that the left hand side of one equation unifies with the left hand side of another equation. In the SOUR Graph, this means that there is a set of nodes v_1, v_2, v_3 and v_4 such that there is a Rewrite edge from v_1 to v_2, a Rewrite edge from v_3 to v_4, and a Unification edge between v_1 and v_3. To perform the inference, we need to add a new equation between the right hand sides of the previous equations. So, in the SOUR Graph, we add a rewrite edge from v_2 to v_4 as in Figure 4b. This is called an RUR configuration. As in the SUR inference, we must label the new edge with a constraint and a renaming. In the example, the renaming is not useful, but the constraint will be the constraint $f(a) \doteq f(x)$, as given in the inference rule. In certain cases, this inference is a simplification, and the equation represented by the rewrite edge between v_1 to v_2 may be removed.

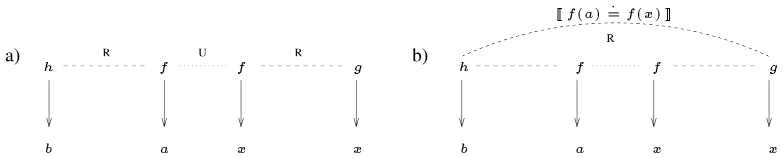

Fig. 4: Inference between $f(x) \approx g(x)$ and $f(a) \approx h(b)$

For one more motivating example, consider the equation $f(f(x)) \approx g(f(x))$ (see Figure 5a). Completion of this rule gives an infinite set of equations. The equations are represented by finitely many edges of the SOUR graph. In other cases, completion creates a SOUR graph containing infinitely many edges; in this case there are infinitely many edges of the same kind between a pair of vertices, but they have different labels. First, note that there is a unification edge between $f(x)$ and $f(f(x))$, because they are unifiable after renaming. Therefore, there is an SUR configuration between the nodes representing $f(f(x))$, $f(x)$, $f(f(x))$, and $g(f(x))$ (as in Figure 5a). This results in adding a new subterm edge from the node representing $f(x)$ to the node representing $g(f(x))$ as in Figure 5b. It corresponds to the inference:

$$\frac{f(f(x\rho_1)) \approx g(f(x\rho_1)) \qquad f(f(x)) \approx g(f(x))}{f(g(f(x\rho_1))) \approx g(f(x)) \llbracket f(f(x\rho_1)) \doteq f(x) \rrbracket}$$

The new subterm edge will have the constraint $f(f(x\rho_1)) \doteq f(x)$ and the renaming ρ_1. This gives us the equation in the conclusion of the inference. An equation in the graph is represented by a rewrite edges. The terms are formed from subterm edges rooted the two ends of the rewrite edge. Constraints on the edges are combined. Renamings on the edges are applied to anything that appears underneath it in the tree. See the next section for a formal definition.

Next we find the SUR configuration from the node representing $g(f(x))$ through the nodes representing $f(x)$ and $f(f(x))$ to the node representing $g(f(x))$ (see Figure 5b). This means we must add a new subterm edge from the node representing $g(f(x))$ to itself, as in Figure 5c. This corresponds to the inference:

$$\frac{f(f(x\tau)) \approx g(f(x\tau)) \qquad f(g(f(x\rho_1))) \approx g(f(x)) \llbracket f(f(x\rho_1)) \doteq f(x) \rrbracket}{f(g(g(f(x\tau)))) \approx g(f(x)) \llbracket f(f(x\tau)) \doteq f(x\rho_1) \wedge f(f(x\rho_1)) \doteq f(x) \rrbracket}$$

The new subterm edge will have the constraint $f(f(x\rho_2)) \doteq f(x)$ and the renaming ρ_2. The new rule which we read from the graph, containing the two new subterm edges, is $f(g(g(f(x\rho_2\rho_1)))) \approx g(f(x)) \llbracket f(f(x\rho_2\rho_1)) \doteq f(x\rho_1) \wedge f(f(x\rho_1)) \doteq f(x) \rrbracket$. Notice that this is the same as the conclusion of the inference, except that τ is replaced by $\rho_2\rho_1$. Therefore, it is a renamed version of the conclusion of the inference.

4 Model

In this section, we will give the syntax and semantics of the SOUR Graph.

Let $G = (V, E)$ represent the SOUR Graph, with V being the set of nodes and E the set of edges. The nodes and edges are labelled as described below. It is possible to have more than one edge between nodes of a SOUR Graph, as long as the edges are labelled differently. A SOUR Graph may contain cycles and loops. Each node v is labelled by a function symbol, a constant, or a variable. We write $Symbol(v) = f$ if f is the label of node v.

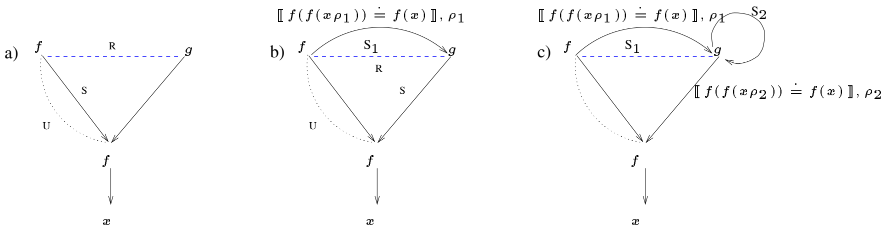

Fig. 5: Completion of $f(f(x)) \approx g(f(x))$

E is the disjoint union of sets E_S, E_R, E_U and E_O. If $e \in E_S$ then e is a *Subterm edge*. If $e \in E_O$ then e is an *Orientation edge*. If $e \in E_U$ then e is a *Unification edge*. If $e \in E_R$ then e is a *Rewrite edge*. For short, we call them S, O, U, and R edges, respectively. Each subterm edge e is labelled by a constraint, a renaming, and an index. We write $Con(e)$ to indicate the parameterized constraint of edge e, $Ren(e)$ to indicate its parameterized renaming, and $Ind(e)$ to indicate the index of a subterm edge. Each rewrite edge e is labelled by a constraint and two renamings. We write $Con(e)$ to indicate its parameterized constraint, and $Ren_l(e)$ and $Ren_r(e)$ to indicate its parameterized renamings. The S and O edges are directed, but the U and R edges are undirected.

We define *S-trees* as an intermediate level for defining the semantics of a SOUR Graph. An S-tree is a tree whose nodes are labelled with function symbols, constants and variables. Its edges are labelled in the same way as the subterm edges of a SOUR graph. In other words, each edge e of an S-tree will have $Con(e)$ denoting its constraint, $Ren(e)$ denoting its renaming, and $Ind(e)$ denoting its index. Furthermore S-trees will have the following properties. Each leaf of an S-tree will be labelled with a constant or a variable. Interior nodes will be labelled with function symbols. If v is a node such that $Symbol(v) = f$ and $arity(f) = n$, then v will have n edges $e_1, \cdots, e_n$ leading to its children, such that $Ind(e_i) = i$ for each i. $S-trees$ are the standard tree representation of terms, with the addition of parameterized constraints and renamings on its edges. S-trees are unfoldings of dags composed of S-edges in the SOUR Graph. The meaning of nodes and edges in the SOUR Graph will be defined by mapping them to a set of S-trees, which is in turn mapped to a set of terms.

Now we describe the semantics of S-trees and SOUR Graphs.

We define the semantics of S-trees inductively. Let T be an S-tree, then we will define $Term(T)$ to be the constrained term that T represents. If T is an S-tree with one node, labelled by symbol c, then $Term(T) = c$. Let T be an S-tree with root v_0 and edge i leading from v_0 to v_i for each i, $1 \leq i \leq n$. Let $Symbol(v_0) = f$, $Con(e_i) = \varphi_i$, $Ren(e_i) = \eta_i$ and $Term(v_i) = t_i \llbracket \Psi_i \rrbracket$ for each i. For each i, let τ_i be a fresh renaming. Then $Term(v_0) = f(t_1\eta_1\tau_1, \cdots, t_n\eta_n\tau_n) \llbracket \bigwedge_{1\leq i\leq n}(\varphi_i\tau_i \wedge \Psi_i\eta_i)\tau_i) \rrbracket$. For the S-tree shown in Figure 6, $Symbol(v_0) = f$, $Term(v_1) = Term(v_2) = h(g(z)) \llbracket z\rho = y \rrbracket$ et

$$Term(v_0) = f(h(g(z\tau_1)), h(g(z\tau_2))) \llbracket z\tau_1 = y \wedge z\tau_2 = y \rrbracket.$$

We inductively define how a node in the SOUR Graph represents a set of trees, and therefore also represents a set of terms. We will define $Trees(v)$ to be the set of trees represented by a node and $Terms(v)$ to be the set of terms represented by the node. If v is labelled by a constant or variable c, then $Trees(v) = \{v\}$. Let v be labelled by an n-ary function symbol f. For each i from 1 to n let e_i be an S-edge leading from v to some vertex v_i in the SOUR Graph, such that $Index(e_i) = i$. Let T_i be an element of $Trees(v_i)$. Then $Trees(v)$ contains a tree whose root is labelled with f and which has n

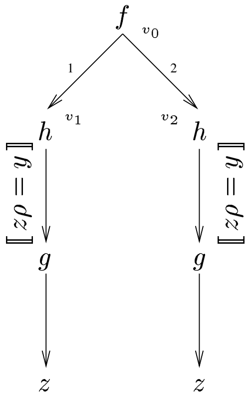

Fig. 6: An S-tree

edges leading to its children. For each i, one of these edges is labelled the same way as e_i and leads to the root of the subtree T_i. For all $v \in G$, we define $Terms(v) = \{Term(T) \mid T \in Trees(v)\}$.

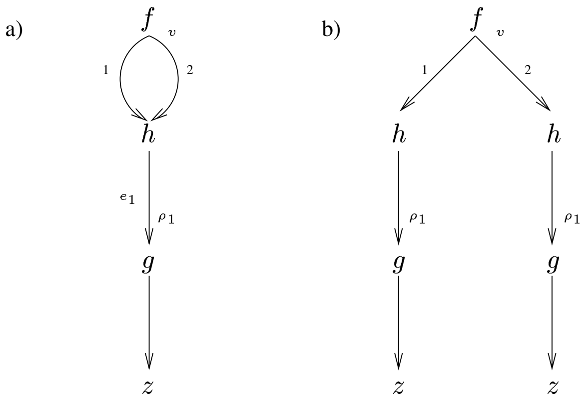

Fig. 7: Trees represented by a SOUR Graph

For SOUR Graph shown in Figure 7a with $Ren(e_1) = \rho_1$ the set $Trees(v)$ is represented in Figure 7b. We have

$$Terms(v) = \{f(h(g(z\rho_1\tau_1)), h(g(z\rho_1\tau_2)))\},$$

where τ_1 and τ_2 are fresh instances of renaming ρ_1.

Next we define how a rewrite edge in the SOUR Graph represents a set of equations. Let e be a rewrite edge between node v_1 and node v_2. Let $t_1 [\![\varphi_1]\!] \in Terms(v_1)$ and $t_2 [\![\varphi_2]\!] \in Terms(v_2)$. Let $Con(e) = \varphi$, $Ren_l(e) = \eta_1$ and $Ren_r(e) = \eta_2$. Then $Equations(e)$ contains the equation $(t_1\eta_1 \approx t_2\eta_2 [\![\varphi_1\eta_1 \wedge \varphi_2\eta_2 \wedge \varphi]\!])\tau$. If G is a SOUR Graph, then $Equations(G) = \bigcup_{e \in E_R} Equations(e)$. Therefore a SOUR Graph represents a constrained rewrite system.

5 Inference System

Completion is performed on SOUR Graphs by representing a set of equations as a graph, and then saturating the set with respect to a set of graph transformations (inference rules) given below.

The set of equations is transformed into a graph in the normal way that a term is represented as a dag. It can be defined inductively. If a term t is a constant or variable c, then $Graph(t)$ is the dag with one node, which is labelled with c. If a term t is of the form $f(t_1, \cdots, t_n)$, then $Graph(t)$ is the dag such that one node v_0 is labelled with f, and there are n subterm edges $e_1, \cdots, e_n$ leading out of v_0, such that for each e_i, $Ind(e_i) = i$, $Con(e_i) = \top$, and $Ren(e_i) = id$. The destination of each e_i is the root of $Graph(t_i)$. As defined, for a term t, $Graph(t)$ is not a unique dag. It may be a tree. However, if some identical trees are shared, then it is no longer a tree. The most efficient representation is when a maximum amount of structure sharing is performed, as in graph shown in Figure 8 representing term $f(h(g(z)), h(g(z)))$.

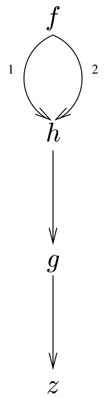

Fig. 8: $Graph(f(h(g(z)), h(g(z))))$

For our application, any amount of structure sharing is allowed. Even if the initial graph is created with no structure sharing at all, the advantages of the SOUR graph are still enjoyed. The reason for that is that the structure sharing is performed automatically by the inference rules.

For an equation $t_1 \approx t_2$, let v_1 be the root of $Graph(t_1)$ and v_2 be the root of $Graph(t_2)$. Then $Graph(t_1 \approx t_2)$ is the graph containing $Graph(t_1)$ and $Graph(t_2)$ such that there is a rewrite edge e between v_1 and v_2, labelled such that $Ren_l(e) = id$, $Ren_r(e) = id$ and $Con(e) = \top$. We may also perform structure sharing between the subterms of t_1 and t_2.

If $\{r_1, \cdots, r_n\}$ is a set of equations, then $Graph(\{r_1, \cdots, r_n\})$ is the graph which is the union of graphs representing $Graph(r_1), \cdots, Graph(r_n)$. There may be structure sharing between the subterms of each r_i. In addition, there is a unification edge between every pair of vertices $v_1, v_2 \in Graph(\{r_1, \cdots, r_n\})$ such that $Symbol(v_1) = Symbol(v_2)$ or $Symbol(v_1) \in Var$ or $Symbol(v_2) \in Var$. Note that v_1 and v_2 are not necessarily distinct nodes.

Now we describe the inference rules. They all correspond to a particular case of the critical pair rule, simplification being a special case:

Basic Critical Pair

$$\frac{(s \approx t\,[\![\,\theta_1\,]\!])\tau \qquad u[s']_p \approx v\,[\![\,\theta_2\,]\!]}{u[t\tau]_p \approx v\,[\![\,s\tau \doteq s' \,\wedge\, \theta_1\tau \,\wedge\, \theta_2\,]\!]}$$

where τ is a fresh renaming substitution

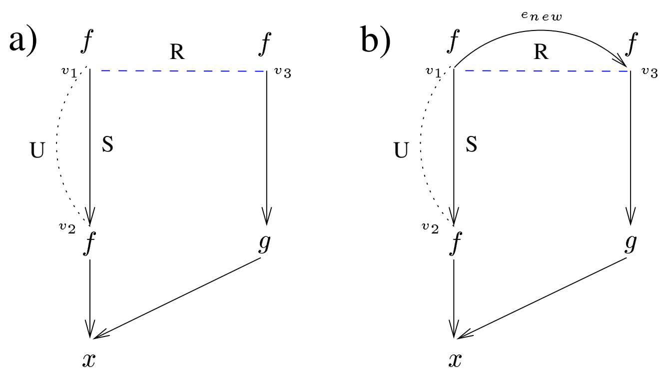

Fig. 9: Example of SUR Transformation

There are two inference rules. They are instances of the Basic Critical Pair inference rule, depending on whether or not the inference is at a root position. These inferences are necessary for completeness. The first inference rule is called an *SUR transformation* and corresponds to a critical pair where p is not the root position of u.

Let us give an example of an SUR transformation. Suppose G is a SOUR graph shown in Figure 9a. Since G contains a subterm edge from v_1 to v_2, a unification edge between v_1 and v_2 and a rewrite edge from v_1 to v_3, we add a new subterm edge e_{new} from v_1 to v_3 as shown in Figure 9b. This SUR transformation corresponds to the Basic Critical Pair inference

$$\frac{f(f(x\tau)) \approx f(g(x\tau)) \qquad f(f(x)) \approx f(g(x))}{f(f(g(x\tau))) \approx f(g(x))\,[\![\,f(x) \doteq f(f(x\tau))\,]\!]}.$$

Now we define the SUR inference rule by referring to a figure. The definition is as follows : Suppose that edges e_s, e_u and e_r exist, such that $Ren(e_s) = \eta_1$, $Ren_l(e_r) = \eta_l$, $Ren_r(e_r) = \eta_r$, $Con(e_s) = \Psi_1$ and $Con(e_r) = \Psi_2$, as in Figure 10. Further suppose that terms $t_1\,[\![\,\varphi_1\,]\!] \in Terms(v_1)$, $t_2\,[\![\,\varphi_2\,]\!] \in Terms(v_2)$, $t_3\,[\![\,\varphi_3\,]\!] \in Terms(v_3)$ and $t_4\,[\![\,\varphi_4\,]\!] \in Terms(v_4)$ exist as in Figure 10. Then the subterm edge e_{new} is added to the graph from v_3 to v_4 as in Figure 10, if Ψ is satisfiable, where

$$\Psi = (t_1\eta_1 \doteq t_2\eta_l\rho) \,\wedge\, \Psi_1 \,\wedge\, \varphi_1\eta_1 \,\wedge\, (\Psi_2 \,\wedge\, \varphi_2\eta_l)\rho,$$

and ρ is a fresh renaming.

The labels on e_{new} will be as in Figure 10. In other words, $Con(e_{new}) = \Psi$, $Ren(e_{new}) = \eta_r\rho$ and $Ind(e_{new}) = Ind(e_s)$.

Let us analyze the inference that was performed. A critical pair inference was performed below the root of some term in an equation.¶ The term that the inference went into is represented by $t_1\,[\![\,\varphi_1\,]\!] \in$

¶ It may not be strictly a critical pair inference, because that term is used for inferences into left hand sides of equations. These inferences may also take place in right hand sides.

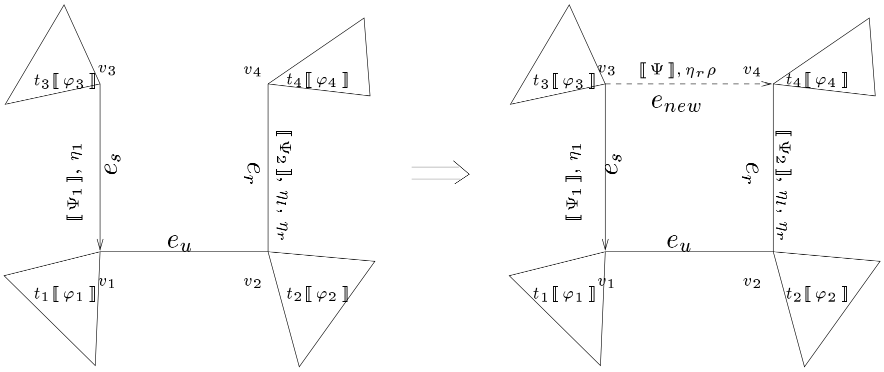

Fig. 10: SUR Transformation

$Terms(v_1)$. Suppose that $Symbol(v_3) = f$ and $Ind(e_s) = k$, then $t_1 [\![\varphi_1]\!]$ is the k^{th} subterm of $f(\cdots, t_1\eta_1\tau, \cdots) [\![\Psi_1\tau \wedge \varphi_1\eta_1\tau]\!] \in Terms(v_3)$, where τ is a fresh renaming. This is straightforward from the definitions. In turn, this term is the subterm of a constrained equation

$$C[f(\cdots, t_1\eta_1\tau, \cdots)\eta] [\![\Psi_1\tau\eta \wedge \varphi_1\eta_1\tau\eta \wedge \varphi]\!]$$

for some equation C, constraint φ and renaming η, occuring at some node v of G. This is understood by noticing that in order to form the equation C containing this term, we add edges above v_3. These edges will further constrain the equation by conjoining a constraint φ, and further rename with some renaming η.

The from equation that is used in this inference is the one represented by edge e_r. It is $t_2\eta_l \approx t_4\eta_r [\![\varphi_2\eta_l \wedge \varphi_4\eta_r \wedge \Psi_2]\!]$. This follows directly from the definition. We need to do an inference between this equation and the one from the previous paragraph. We need to consider a renaming of this equation, such that no variables are in common with the other one. Consider the renaming $\rho\tau\eta$. Therefore, the renamed equation is $(t_2\eta_l \approx t_4\eta_r [\![\varphi_2\eta_l \wedge \varphi_4\eta_r \wedge \Psi_2]\!])\rho\tau\eta$. This has no variables in common with the other equation, because of the fresh renaming ρ, which the other equation does not use. It may not seem obvious where this renaming comes from, but we will see that is the renaming that arises in the SOUR Graph.

Therefore the inference is the following.

$$\frac{(t_2\eta_l \approx t_4\eta_r [\![\varphi_2\eta_l \wedge \varphi_4\eta_r \wedge \Psi_2]\!])\rho\tau\eta \qquad C[f(\cdots, t_1\eta_1\tau, \cdots)\eta] [\![\Psi_1\tau\eta \wedge \varphi_1\eta_1\tau\eta \wedge \varphi]\!]}{C[f(\cdots, t_4\eta_r\rho\tau, \cdots)\eta] [\![\Psi\tau\eta \wedge \varphi_4\eta_r\tau\eta \wedge \varphi]\!]}$$

Note that before the edge was added, the node v_3 represented the right premise of the inference. After the new edge is added, the node v_3 represents the right premise of the inference and the conclusion of the inference.

We show that the conclusion of this inference is represented in the graph by considering the dag representing the into equation, such that the original subtree at v_3 is replaced by a new subtree, consisting of e_{new} and the subtree at v_4. We know that $t_4 [\![\varphi_4]\!] \in Terms(v_4)$. This is the k^{th} subterm of

$f(\cdots, t_4\eta_r\tau, \cdots)\,[\![\, \Psi\tau \,\wedge\, \varphi_4\eta_r\rho\tau \,]\!] \in Terms(v_3)$, where τ is the same τ as above. In turn this term is the subterm of an equation

$$C[f(\cdots, t_4\eta_r\tau, \cdots)\eta]\,[\![\, \Psi\tau\eta \,\wedge\, \varphi_4\eta_r\rho\tau\eta \,\wedge\, \varphi \,]\!]$$

which is exactly the equation that is the conclusion of the inference.

The above argument leads to the following lemmas. The first lemma will be used in our soundness proof in the next section.

Lemma 1 *Let G be a graph and let G' be the result of performing an SUR inference on G. Then each equation in $Equations(G')$ is implied by $Equations(G)$.*

Proof: Let eq be an equation in $Equations(G')$. We want to show that eq is implied by $Equations(G)$. If $eq \in Equations(G)$ then eq is obviously implied by $Equations(G)$. So suppose that $eq \notin Equations(G)$. Then eq must be an equation formed using the edge created by the SUR transformation. The above argument shows that each such eq is a result of an inference between two equations in $Equations(G)$. Therefore eq is implied by $Equations(G)$. $\square$

The second lemma will be used in the completeness proof in the next section.

Lemma 2 *Suppose that eq_1 and eq_2 are equations in $Equations(G)$ such that there is an inference between eq_1 and eq_2 below the root. Let eq_3 be the conclusion of that inference. Then there is an SUR transition from G to some G' such that $eq_3 \in Equations(G')$.*

Proof: Let eq_1 be the into equation and eq_2 be the from equation. Then there is a subterm t of eq_1 which unifies with one side of eq_2. In the graph there must be a unification edge between the node representing t and the node representing the side of eq_2 which unifies with t. There must be a subterm edge between the node representing the immediate superterm of t and the node representing t. There also must be a rewrite edge between the nodes representing the two sides of eq_2. Therefore, there is an SUR configuration. By the above argument, we see that the new edge added creates the conclusion of the inference between eq_1 and eq_2. $\square$

The inference must be performed for every possible value of $Terms(v_1)$ and $Terms(v_2)$. The values of $Terms(v_1)$ and $Terms(v_2)$ are modified when new subterm edges are added to the graph. So the inference needs to be re-performed for these new values.

The second inference rule is called an *RUR transformation* and corresponds to a critical pair where p is the root position of u in the critical pair rule.

This is an example of an RUR transformation. Let G be a SOUR graph shown in Figure 11a (everything is the same as in Figure 9b except the unification edge).

Since G contains a rewrite edge from v_1 to v_3, a unification edge between v_1 and itself, we add a new rewrite edge e_{new} from v_3 to itself as shown in Figure 11b.

This RUR transformation corresponds to the Basic Critical Pair inference

$$\frac{f(f(x\tau'))\approx f(g(x\tau')) \qquad f(f(g(x\tau)))\approx f(g(x))}{f(g(x\tau'))\approx f(g(x))\,[\![\, f(x)\doteq f(f(x\tau)) \wedge f(f(x\tau'))\doteq f(f(g(x\tau))) \,]\!]}.$$

The definition of the RUR inference rule, referring to Figure 12, is as follows. Let e_{r_1} be a rewrite edge between nodes v_1 and v_3, e_{r_2} be a rewrite edge between v_2 and v_4 and e_u be a unification edge

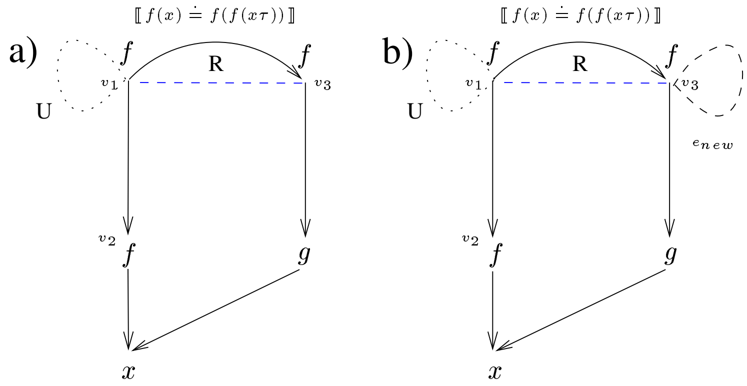

Fig. 11: Example of RUR Transformation

between nodes v_1 and v_2 such that $Ren_l(e_{r_1}) = \eta_1$, $Ren_r(e_{r_1}) = \eta_3$, $Ren_l(e_{r_2}) = \eta_2$, $Ren_r(e_{r_2}) = \eta_4$, $Con(e_{r_1}) = \Psi_1$ and $Con(e_{r_2}) = \Psi_2$. Suppose that terms $t_1 [\![\varphi_1]\!] \in Terms(v_1)$, $t_2 [\![\varphi_2]\!] \in Terms(v_2)$, $t_3 [\![\varphi_3]\!] \in Terms(v_3)$ and $t_4 [\![\varphi_4]\!] \in Terms(v_4)$ exist. Then the rewrite edge e_{new} is added to the graph between v_3 and v_4 as in Figure 12, if Ψ is satisfiable, where

$$\Psi = (t_1\eta_1 \doteq t_2\eta_2\rho) \wedge \Psi_1 \wedge \varphi_1\eta_1 \wedge (\Psi_2 \wedge \varphi_2\eta_2)\rho,$$

and ρ is a fresh renaming.

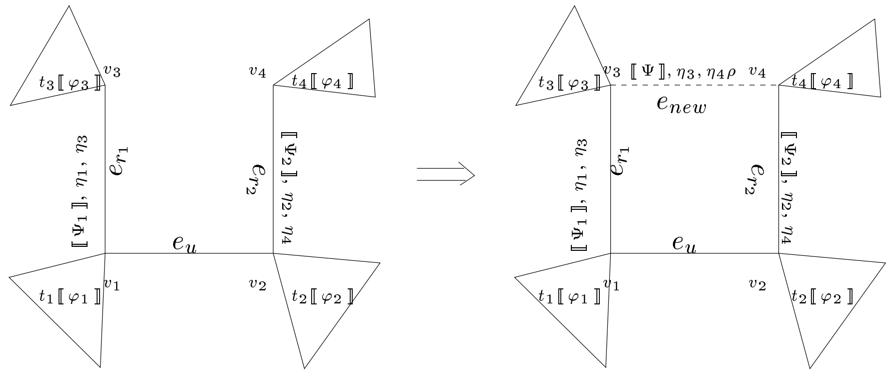

Fig. 12: RUR Transformation

The labels on e_{new} will be $Con(e_{new}) = \Psi$, $Ren_l(e_{new}) = \eta_3$, and $Ren_r(e_{new}) = \eta_4\rho$.

Let us analyze the inference that was performed here. An inference was performed at the root of some term in an equation. The term that the inference went into is $t_1 [\![\varphi_1]\!] \in Terms(v_1)$. This is a part of the constrained equation

$$(t_1\eta_1 \approx t_3\eta_3 [\![\Psi_1 \wedge \varphi_1\eta_1 \wedge \varphi_3\eta_3]\!])\tau$$

which is represented by edge e_{r_1}.

The from equation that is used in this inference is the one represented by edge e_{r_2}. It is $t_2\eta_2 \approx t_4\eta_4 [\![\varphi_2\eta_2 \wedge \varphi_4\eta_4 \wedge \Psi_2]\!]$. This follows directly from the definition. We need to do an inference between this equation and the one from the previous paragraph. We need to consider a renaming of this equation, such that no variables are in common with the other one. Consider the renaming $\rho\tau$. Therefore, the renamed equation is $(t_2\eta_2 \approx t_4\eta_4 [\![\varphi_2\eta_2 \wedge \varphi_4\eta_4 \wedge \Psi_2]\!])\rho\tau$. This has no variables in common with the other equation, because of the fresh renaming ρ, which the other equation does not use.

Therefore the inference is the following.

$$\frac{(t_2\eta_2 \approx t_4\eta_4 [\![\varphi_2\eta_2 \wedge \varphi_4\eta_4 \wedge \Psi_2]\!])\rho\tau \qquad (t_1\eta_1 \approx t_3\eta_3 [\![\varphi_1\eta_1 \wedge \varphi_3\eta_3 \wedge \Psi_1]\!])\tau}{(t_3\eta_3 \approx t_4\eta_4\rho [\![\Psi\rho \wedge \varphi_3\eta_3 \wedge \varphi_4\eta_4\rho]\!])\tau}$$

We show that the conclusion of this inference is represented in the graph by considering the equation represented by e_{new}. We know that $t_3 [\![\varphi_3]\!] \in Terms(v_3)$ and $t_4 [\![\varphi_4]\!] \in Terms(v_4)$. Therefore, e_{new} represents the equation

$$(t_3\eta_3 \approx t_4\eta_4\rho [\![\Psi\rho \wedge \varphi_3\eta_3 \wedge \varphi_4\eta_4\rho]\!])\tau$$

which is exactly the equation that is the conclusion of the inference.

The above argument leads to the following lemmas. The first lemma will be used in our soundness proof in the next section.

Lemma 3 *Let G be a graph and let G' be the result of performing an RUR inference on G. Then each equation in $Equations(G')$ is implied by $Equations(G)$.*

Proof: Let eq be an equation in $Equations(G')$. We want to show that eq is implied by $Equations(G)$. If $eq \in Equations(G)$ then eq is obviously implied by $Equations(G)$. So suppose that $eq \notin Equations(G)$. Then eq must be an equation formed using the edge created by the RUR transformation. The above argument shows that each such eq is a result of an inference between two equations in $Equations(G)$. Therefore eq is implied by $Equations(G)$. □

The second lemma will be used in the completeness proof in the next section.

Lemma 4 *Suppose that eq_1 and eq_2 are equations in $Equations(G)$ such that there is an inference between eq_1 and eq_2 at the root. Let eq_3 be the conclusion of that inference. Then there is an RUR transition from G to some G' such that $eq_3 \in Equations(G')$.*

Proof: Let eq_1 be the into equation and eq_2 be the from equation. Then one side t_1 of eq_1 unifies with one side t_2 of eq_2. In the graph there must be a unification edge between the node representing t_1 and the node representing t_2. There must be a rewrite edge between the nodes representing the two sides of eq_1, and another rewrite edge between the nodes representing the two sides of eq_2. Therefore, there is an RUR configuration. By the above argument, we see that the new edge added creates the conclusion of the inference between eq_1 and eq_2. □

The inference must be performed for every possible value of $Terms(v_1)$ and $Terms(v_3)$. The values of $Terms(v_1)$ and $Terms(v_3)$ are modified when new subterm edges are added to the graph. So the inference needs to be re-performed for these new values.

6 Completeness and Soundness

In this section, we show that SOUR Completion is sound and complete. Let R be a set of equations. We show that saturating $Graph(R)$ with respect to the graph transformation rules results in a graph G^* which represents the saturation of R with respect to the Basic Completion inference rules.

We need to model what happens in the completion process. To do that, we must first define the notion of *redundancy*. A *redundant* equation is one that is implied by smaller equations, therefore it may be removed. For instance, a simplified equation becomes redundant. More formally, if C is a ground equation and S is a set of ground equations, then C is *redundant in* S if there exist equations $C_1, \cdots, C_n \in S$ such that

1. for all i, $C_i < C$,
2. C is true whenever $C_1, \cdots, C_n$ are true.

For a non-ground term, we give a sufficient definition of redundancy, which is an instance of the one defined by [BGLS95]. If $C\llbracket \varphi \rrbracket$ is a non-ground term, we say that $C\llbracket \varphi \rrbracket$ is *redundant in* S if for all instances $C\sigma$ of $C\llbracket \varphi \rrbracket$ where $\sigma = mgs(\varphi)$, there exist $C_1\llbracket \varphi_1 \rrbracket, \cdots, C_n\llbracket \varphi_n \rrbracket \in S$ and instances $C_i\sigma_i$ of $C_i\llbracket \varphi_i \rrbracket$ where $\sigma_i\ mgs(\varphi_i)$ for each i such that

1. for all i, $C_i\sigma_i < C\sigma$,
2. $C\sigma$ is true whenever $C_1\sigma_1, \cdots, C_n\sigma_n$ are true, and
3. for all i, every term $t_i \in Ran(\sigma_i)$ is a subterm of some term $t \in Ran(\sigma)$.

In this paper, we do not actually need the notion of redundancy. Normally, it is needed in a completion procedure to deal with simplification. But we have not presented simplification in this paper. Simplification is based on orderings, which we also have not presented in this paper. It is possible to use orderings and simplification in SOUR Graphs. However, the notation then becomes even more complex than it is now. We chose not to include them in order to simplify the paper. But we still include redundancy to have the framework so it is possible to add simplification later. Also, this has become the standard way to present Knuth-Bendix Completion.

We define a *completion derivation* as in [BGLS95, NR92]. A *completion derivation from* R_0 is a (possibly infinite) sequence of equations $R_0, R_1, \cdots$ such that for $n \geq 0$, R_{n+1} is formed by adding an equation $s \approx t\llbracket \varphi \rrbracket$ such that $R_n \models s \approx t\llbracket \varphi \rrbracket$ or removing an equation $s \approx t\llbracket \varphi \rrbracket$ such that $s \approx t\llbracket \varphi \rrbracket$ is redundant in R_n. Let $R_\infty = \bigcup_{i \geq 0} \bigcap_{j \geq i} R_j$. An equation $s \approx t\llbracket \varphi \rrbracket \in R_\infty$ is defined to be *persistent*. A completion derivation is *fair* if all inferences among persistent equations are redundant in R_∞. A sufficient condition for fairness is that all non-redundant inferences among persisting equations are eventually performed. A completion derivation is *sound* if each R_n is logically equivalent to R_0. A completion derivation is *complete* if R_∞ is canonical.

A practical completion procedure is an example of a completion derivation, because critical pair inferences add equations which follow from existing equations and simplifications add equations which follows from existing equations and remove redundant equations.

Let I^{basic} be an inference system which performs the basic critical pair rule and basic simplification rule under some strategy. We have the following result from [BGLS95, NR92]

Theorem 1 *Let $R_0, R_1, \cdots$ be a completion derivation from R_0, with the inference rules I^{basic}. The derivation is sound. Furthermore, if the derivation is fair, and all constraints in R_0 are $\top$, then the derivation is complete.*

SOUR Completion according to some strategy also determines a theorem proving derivation. Given a set of equations R, we transform R into the graph $Graph(R)$. Then continually add edges to the graph as a result of an SUR or RUR, transformation. Call this inference system I^{SOUR}. It determines a completion derivation because each graph created can be seen as a set of equations using the $Rules$ function. An edge in the SOUR graph is persistent if it is added and never removed. We will define I^{SOUR} to be *fair* if every transformation among persistent edges is eventually performed.

We start with a result to indicate how the set of equations changes when an inference is performed on the graph. The lemma shows that every transformation on the graph corresponds to a series of steps in a completion derivation. This implies that our inference rules are sound.

Lemma 5 *Let G be a graph and let G' be the result of performing an SUR or RUR transformation on G. Then each equation in $Equations(G')$ is implied by $Equations(G)$.*

Proof: This is immediately implied by Lemmas 1 and 3. □

The next lemma is used to show that a fair SOUR graph inference strategy is also a fair completion derivation. This implies that our inference rules are complete.

Lemma 6 *Suppose that eq_1 and eq_2 are equations in $Equations(G)$ such that there is an inference between eq_1 and eq_2. Let eq_3 be the conclusion of that inference. Then there is an SUR transition from G to some G' such that $eq_3 \in Equations(G')$.*

Proof: This follows immediately from Lemmas 2 and 4. □

We can define a *SOUR derivation* from a graph $G_0 = (V, E)$ to be a (possibly infinite) sequence of graphs $G_0, G_1, \cdots$, such that for $n \geq 0$, G_{n+1} is formed from G_n by performing an SUR or RUR transformation. Let each $G_i = (V, E_i)$, and define $G_\infty = (V_0, E_\infty)$, where $E_\infty = \bigcup_{i \geq 0} \bigcap_{j \geq i} E_j$. A SOUR derivation is *fair* if all SUR and RUR transformations in G_∞ have been performed in the derivation. A SOUR derivation is *sound* if $Equations(G_n)$ is logically equivalent to $Equations(G_0)$ for all n. A SOUR derivation is *complete* if $Equations(G_\infty)$ is canonical.

The completeness theorem follows immediately from the previous two lemmas.

Theorem 2 *Let $G_0, G_1, \cdots$ be a SOUR derivation from G_0. The derivation is sound. Furthermore, if the derivation is fair, and in G_0 all edges have a constraint $\top$, then the derivation is complete.*

Proof: Lemma 5 implies that the SOUR derivation corresponds to a completion derivation. Therefore the SOUR derivation is sound. Lemma 6 ensures that if the SOUR derivation is fair then all non-redundant critical pairs in $Equations(G_\infty)$ are performed. Therefore the SOUR derivation is complete. □

7 Preliminary Results on an Implementation

We are in the process of implementing the SOUR graph inference system. The implementation is written in C++, using the LEDA package of the Max Planck Institute [Nah93] to implement the graphical data structures. The implementation is now working for the completion of ground terms. We give some implementation details and experimental results for ground completion for our implementation.

In the implementation, we store the SOUR Graph in a graph structure provided by LEDA. The terms are read in and stored in the graph using the maximum amount of structure sharing. Unification edges are added as the terms are created. All this is done in a bottom up fashion. We perform SOUR completion by making several passes through the graph. In each pass, we perform some transformations. We stop when all the transformations have been done. A pass through the graph consists of choosing rewrite edges one by one. For each rewrite edge, we first try to perform RUR inferences involving that edge. Then all SUR inferences using that edge are performed. As new subterm edges are added, new unification edges may also be added. In the ground case, things are much simpler, because we do not need constraints or renamings. Also, simplifications are easy to perform. Each SUR and RUR transformation corresponds to a simplification, and an edge is deleted in each transformation. This also means that we keep orientation information, which is created and updated at the same time as the unification edges are. We can also collapse two nodes to one when they represent exactly the same term.

First, we note that ground completion using SOUR graphs always needs only a polynomial space to store the system. This is easily seen, because in the ground case, without constraints and renamings, there can only be $\frac{n(n+1)}{2}$ edges of any type, where n is the number of nodes. This is true, because there are never two edges of the same type between any two nodes v_1 and v_2. However, the standard completion algorithm may use an exponential amount of space. Consider the set of rewrite rules $a_0 \rightarrow f(a_1, a_1), \cdots, a_{n-1} \rightarrow f(a_n, a_n), f(a_0, a_0) \rightarrow b$, with the ordering $a_0 > \cdots > a_n > f > b$. Under this ordering, the completed system contains an exponentially large equation. [‖] Of course, it could be avoided by structure sharing. But that is what is so nice about the SOUR graph technique. The structure sharing is automatic.

In the following table, we give some experimental results of our implementation compared with OTTER[McC94]. All the experiments were performed on a Sun4 Sparc station. We used version 2.2 of OTTER. All the results are listed in seconds. The first column gives the name of the problem. The second column gives the number of seconds needed for completion using our system. The third column gives the number of seconds OTTER requires for completion. It is difficult to find examples in the literature for ground completion. Therefore, some of our examples are taken mostly at random. This is the case for problems *dc2* and *data3*. Problem *s4* is adapted from a problem of group theory, for the symmetric group of degree 4. Problem *gcd_4_8_10* is the set of equations $\{f^4(a) = a, f^8(a) = a, f^{10}(a) = a\}$. Completing this system is equivalent to finding the greatest common divisor of 4, 8, and 10, where a is zero and f is the successor function. Problem *gnprs* is taken from [GNP+93], where they used it to illustrate their congruence closure algorithm. Problem *counter5* is taken from [GNP+93, PSK96], where it is used to illustrate that completion can take exponential time even with structure sharing, if a proper strategy is not used. The strategy OTTER uses does not have this bad behavior. Finally, problem *expf6* is an adaptation of the example given above showing that completion can generate exponentially large terms.

‖ This example is also given in [PSK96].

Problem	SOUR	OTTER
gcd_4_8_10	0.43	0.67
gnprs	0.45	0.91
data3	1.04	1.42
counter5	0.45	1.81
dc2	1.04	2.46
s4	2.12	6.31
expf6	2.98	*

In many other examples, SOUR completion and OTTER had similar running times. Sometimes, OTTER was a little faster because of its efficient implementation.

We do not claim these results as conclusive evidence of our method. But we are encouraged by the fact that our implementation compares favorably to OTTER. The "*" in the last row of the chart indicates that OTTER was unable to prove this theorem because it ran out of memory. This confirms the theoretical result that OTTER generates exponentially large terms in this example. Our conclusion is that our results compare favorably with OTTER in many cases, and that in some cases it performs much better. Since our non-ground method is an extension of the ground method, we expect that the results will continue to be good in the non-ground case.

8 Perspectives

One of our claims is that SOUR Graphs implement completion in a more fundamental way, thereby allowing us to examine the completion procedure more closely. In this section, we give evidence to that claim by showing how some important open problems in the field of completion can be solved using SOUR Graphs. This includes the problem of constructing a fine-grained concurrent completion procedure, and the problem of constructing a goal-directed completion algorithm. We also explain how to use SOUR graph-based structures to complete finitely presented groups.

8.1 Fine-Grained Concurrent Completion

Parallelism is an important problem in completion, and in theorem proving in general. The goal is to parallelize the completion procedure, so the work is divided among different processors, making the procedure more efficient. Simplification is useful for keeping the search space of a completion procedure small. In order to handle simplification, parallel completion procedures need to have some kind of global store or global control. In other words, there must be some global location which all processors access in order for an equation at one processor to simplify an equation at another processor. Or, there must be a master who controls each of its slave processors. In a true concurrent procedure, each processor would be able to act on its own, with only some message passing between it and some neighbor processors.

Another question about parallelism is the granularity of the pieces into which the problem is divided. Other completion procedures use coarse grain procedures, where a processor controls a whole equation. A fine-grained procedure would be able to divide up the search space further so that each processor controls related subterms, rather than a whole equation.

SOUR graphs can be used to create a truly fine-grained completion procedure, with no global control or store, and with the search space broken down into fine-grained pieces. The idea is very simple, and is presented in [KLS95]. In the SOUR graph, nodes are processes and edges are communication links. This allows for a truly local procedure. Each process has only a local view of the graph. It only knows about which nodes are connected to it by an edge in the graph. The graph transformations that are described in this paper are local graph transformations. In other words, only local information is necessary to detect a configuration. It is not necessary to know which equations are involved, only that local patterns are formed in the graph.

In the concurrent procedure, there are three kinds of operations that must be performed.

- Unification must be detected.
- Orientation must be detected.
- SUR and RUR configurations must be detected and processed.

These operations can all be performed in parallel by message passing between nodes of the graph, along the edges of the graph. To detect that two terms are unifiable, it is necessary to know that the subterms are unifiable. Therefore, information must be passed from the final node of a subterm edge to the initial node, if the unification of the subterms is possible. The initial node uses the information from all its subterms to decide if the top terms are unifiable. Orientation is similar. We use the lexicographic path ordering. Suppose we have a node u such that the set of subterm edges with initial node u have a final node in the set $\{u_1, \cdots, u_n\}$, and another node v such that the set of subterm edges with initial node v have a final node in the set $\{v_1, \cdots, v_m\}$. Then the ordering between two terms at nodes v_1 and v_2 is a function of the orderings between the terms at (i) u and $\{v_1, \cdots, v_m\}$, (ii) v and $\{u_1, \cdots, u_n\}$, and (iii) $\{u_1, \cdots, u_n\}$ and $\{v_1, \cdots, v_m\}$. Nodes can determine their orientation in relation to other edges and pass that information up along a subterm edge to its parent edge to determine the orientation of the whole term. Finally, configurations can be detected by passing information among the nodes of the configurations, through the subterm, unification and rewrite edges that exist in the configuration. When a new edge is created, this information is passed from the creator of the new edge to the node at the other end.

The completion procedure is just a matter of performing these three operations in parallel everywhere such an operation exists in the graph. All this would not be possible without the fundamental view of SOUR graphs.

8.2 Goal directed Completion

The main benefit of the completion procedure is that it is based on orderings. The orderings are used to direct the inferences, so that the conclusion of an inference is smaller than one of the premises. In this way, many unnecessary inferences are avoided. This results in a big reduction of the search space, such that in many cases the procedure will halt. In those cases, the completed system forms a decision procedure for solving the word problem of the equational theory which was completed.

The main disadvantage of the completion procedure is that if there is a particular equation which we are trying to solve (the *goal*), then there is no way to use the goal to determine which critical pairs need to be created. The completion procedure ignores the goal, and creates the same critical pairs no matter which goal is trying to be solved. Nobody has ever been able to define a procedure which uses the goal in any way, much less define a procedure which starts with the goal and works backwards to initial equations.

The problem of defining a goal directed completion procedure is the problem of modifying the completion procedure to be goal directed, such that the orderings are still used to restrict the search space. There is a goal directed procedure for solving E-unification problems [Sny91] (called *General E-Unification*), that does not use orderings. That procedure starts with the goal, and then uses equations to create another equivalent goal, until an identity is reached. But the fact that it cannot use orderings limits its effectiveness. Suppose the equational theory is $\{f(a) \approx a\}$, and the goal is $a \approx b$. This procedure would create a new goal $f(a) \approx b$ because a equals $f(a)$, then from the new goal it would create another goal $f(f(a)) \approx b$, etc. It would create infinitely many subgoals, and it would never halt, even thought the goal is obviously not true in the theory. We can also consider an example where completion would not perform well. Suppose we had the equational theory $\{f(f(x)) \approx g(f(x))\}$ and the goal $a \approx b$. The completion of $\{f(f(x)) \approx g(f(x))\}$ creates infinitely many equations, even though the goal $a \approx b$ is obviously not true in that theory. These two examples illustrate the need for a goal-directed completion procedure.

In [Lyn95a], a goal directed completion procedure is given, based on SOUR graphs. This procedure uses the idea that the completion procedure using SOUR graphs sometimes produces infinitely many edges, but the number of nodes never expands, so there are only finitely many nodes in the graph (actually, no more than the size of the initial problem). This means that if we ignore the constraints and renamings, then there are only finitely many edges in the graph. It is possible to modify the SOUR graph inference system to have a kind of recursive constraints. The constraint on an edge is defined in terms of which edges the constraint is inherited from, but the actual value of the constraint is not calculated. It is not until all the edges have been generated, that the procedure tries to calculate the actual value of all the constraints. The constraints may be recursive, because the value of the constraint on edge e_1 may be defined in terms of the constraint of edge e_2, which is in turn defined in terms of the constraint of edge e_1, for example.

The goal directed completion procedure has two phases. The first phase is the compilation phase. In this phase, all the edges and the recursive constraints labelling each edge are created. This phase also takes into account the goal to be solved. Importantly, this phase takes only polynomial time, because there are only polynomially many edges in the graph. The result of this phase is a constrained tree automaton representing a schematized version of the completed system, and a set of constraints representing potential solutions to the goal. The constraints that are generated are the equational constraints representing the unification problems, and ordering constraints arising from the critical pair inferences.

The second phase is the goal solving (or constraint solving) phase. In this phase, the potential solutions to the goal are solved in order to determine if they are actual solutions of the goal. This phase can take infinitely long, since the constraints are recursive. Step by step, a constraint is rolled back, based on which edges it is created from, and the equational and ordering constraints are solved along the way. In some cases, the ordering constraints cause the recursion to halt, and therefore the constraints are completely solved.

The procedure is truly goal directed, because only a polynomial amount of time is spent compiling the set of equations. The rest of the time is spent working backwards from the goal to solve the constraints. If the procedure is examined more closely, we see that the second phase of the procedure is exactly a backwards process of completion. A schematization of an equation in the completed system is applied to the goal, step by step until it rewrites to an identity. At the same time, the schematized equation that is selected is worked backwards until we reach the original equations from which it is formed.

8.3 Group Completion

The problem of defining specialized procedure for completion in particular equational theories is important. Much attention has been given to the theory of associativity and commutativity, to give special unification procedures, with the intention of avoiding critical pairs using the associativity or commutativity axiom. Group theory is an especially important equational theory, because it has so many natural applications. In particular the problem of the completion of ground equations modulo group theory is important. A finite set of ground equations on group generators is called a presentation of a group. A group is finitely presented if it has a finite presentation and finitely many generators. Finitely presented groups are used for problems like network routing. The word problem is undecidable for finitely presented groups. Finite groups are a subclass of finitely presented groups, for which the word problem is decidable, but the completion algorithm is very inefficient.

The work of [LC86] is significant, because it shows a way to complete finitely presented groups, while avoiding critical pairs involving group axioms. The work of [LC86] can be implemented by adding a new set of inference rules to the critical pair axiom, which are not as prolific as critical pairs involving group axioms. But the search space is still very big.

In [LS96], we give a solution to the finitely presented group completion problem that is inspired by SOUR graphs, while using a different data structure. The idea of the paper is to develop a special graphical data structure for finitely presented groups, that allows the inference rules to be performed by local modifications of that graph, exactly as in the SOUR graph approach. The result is a new graph structure, called Patch graphs.

The main idea behind Patch graphs is that the tree (or dag) data structure is not ideal for groups. Group elements can be represented as strings on a generating set, but that is not efficient enough, because they have more structure than that. The associativity lets them be represented as strings. But to take advantage of group properties, it is better to represent them as cycles. An equation $s \approx t$ is equivalent to the equation $st^{-1} = \epsilon$. This can be shifted to represent many different equations, each represented by the same cycle. For Patch graphs, we take the inverse approach to SOUR graphs. In other words, edges are labelled by symbols, instead of nodes. We read a term by concatenating the symbols on the edges. Therefore the edges are called concatenation edges instead of subterm edges. Inference rules add new edges, labelled by the empty word, to the graph. In figure 13a, we show a cycle representing the equation $ab \approx cd$. Because of the group structure, the equation $ab \approx cd$ is equivalent to equations $b^{-1}a^{-1} \approx d^{-1}c^{-1}$, $a^{-1}c \approx bd^{-1}$, and $c^{-1}a \approx db^{-1}$. All of these equations are represented by the cycle given in Figure 13a. Furthermore, these cycles make the basic inference rules for finitely presented group completion very easy to perform.

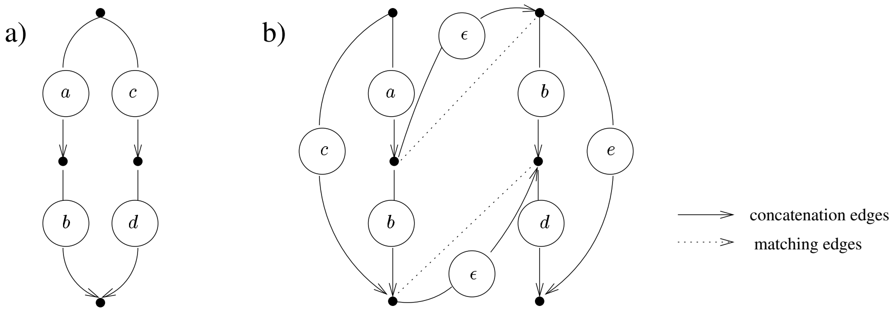

Fig. 13: Patch Graph Inference

The critical pair rule consists of matching a path in one cycle with a path in another cycle, and creating a new cycle by adding an edge between the beginning nodes of the matching path and the ending nodes of the matching path. This simultaneously computes critical pairs between equivalent equations represented by the two cycles. In Figure 13b, we show a critical pair between $ab \approx c$ and $bd \approx e$, resulting in the new cycle representing $ae \approx cd$. The new concatenation edges that are added are labelled by the empty word ϵ.

9 Conclusion

In this paper, we have introduced a new graphical data structure to use for performing completion. It provides maximal structure sharing, so that all terms with the same ancestor are stored together. Therefore, an inference on one term automatically performs the same inference on all terms with the same ancestor. We do this by saving equations in a SOUR graph, and performing local graph transformations on the SOUR graph as inference rules. Each transformation adds a new edge to the graph, and possibly deletes an old one. The new edges are labelled with unification constraints and variable renamings.

The earliest related work on efficient methods of completion is the thesis of Dexter Kozen [Koz77]. In his thesis, he gave a method based on congruence closure to solve the word problem for ground equations in polynomial time. His method was extended in [GNP+93], which gave a completion algorithm which runs in $O(n^3)$ time. This was further extended in [Sny93], which improved the running time of the algorithm to $O(nlog(n))$. However, these methods only work for the ground case, and were not extended to the general case. Our work can be seen as an extension of the congruence closure method to the non-ground case. It is based on the result of [Lyn95b], which gives a general theorem proving method, that is polynomial when reduced to the case of ground completion. This paper was based on the recent completeness results for Basic Paramodulation [BGLS95, NR92]. Our current paper is the result of tailoring the method of [Lyn95b] for the case of completion. We have developed a system which can easily be implemented, because the inference rules are specified as graph transformations. The experimental results we give suggest that the method does work well in practice. Recently, Plaisted and Sattler-Klein [PSK96] have shown that completion can be performed in polynomial time if structure sharing is used and the inferences are performed with a certain strategy. We haven't seen any experimental results of their idea, but we expect it will also perform well.

We believe that the SOUR graph data structure will result in more efficient completion procedures, and we are currently in the progress of implementing it. Furthermore, we believe that SOUR graphs provide a method to examine the Completion Procedure more closely and will be useful in other applications.

Acknowledgements

We are indebted to Claude Kirchner and Dominique Fortin for their support in the development of these ideas, and for many stimulating discussions. We also would like to thank the anonymous referees for providing suggestions to improve the presentation of this paper.

References

[BGLS95] L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic paramodulation. *Information and Computation*, 121(2):172–192, 1995.

[GNP+93] J. Gallier, P. Narandran, D. Plaisted, S. Raatz, and W. Snyder. Finding canonical rewriting systems equivalent to a finite set of ground equations in polynomial time. *Journal of Association for Computing Machinery*, 40(1):1–16, 1993.

[KB70] Donald E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, Oxford, 1970.

[KLS95] Claude Kirchner, Christopher Lynch, and Christelle Scharff. A fine-grained concurrent completion procedure. In *Proceedings of the RTA'96 Conference*, pages 3–17, Lecture Notes in Computer Science 1103, 1996.

[Koz77] Dexter Kozen. *Complexity of Finitely Presented Algebras*. PhD thesis, Cornell University, 1977.

[LC86] Ph. Le Chenadec. *Canonical Forms in Finitely Presented Algebras*. John Wiley & Sons, 1986.

[LS96] Christopher Lynch and Polina Strogova. Patch graphs: an efficient data structure for completion of finitely presented groups. In J. Pfalzgraf, editor, *Proc. Third International Conference on Artificial Intelligence and Symbolic Mathematical Computation*, Lecture Notes in Computer Science. Springer-Verlag, 1996.

[Lyn95a] Christopher Lynch. Goal directed completion using SOUR graphs. In *Proceedings of the RTA'97 Conference*, pages 8–22, Lecture Notes in Computer Science 1232, 1997.

[Lyn95b] Christopher Lynch. Paramodulation without duplication. In Dexter Kozen, editor, *Proceedings 10th IEEE Symposium on Logic in Computer Science, San Diego (Ca., USA)*, pages 167–177, San Diego, June 1995. IEEE.

[McC94] W. W. McCune. Otter 3.0: Reference manual and guide. Technical Report 6, Argonne National Laboratory, 1994.

[Nah93] Stefan Naher. Leda manual. Technical Report MPI-I-93-109, mpi, 1993.

[NR92] R. Nieuwenhuis and A. Rubio. Basic superposition is complete. In B. Krieg-Brückner, editor, *Proceedings of ESOP'92*, volume 582 of *Lecture Notes in Computer Science*, pages 371–389. Springer-Verlag, 1992.

[PSK96] David A. Plaisted and Andrea Sattler-Klein. Proof lengths for equational completion. *Information and Computation*, 125(2):154–170, 15 March 1996.

[Sny91] Wayne Snyder. *A proof theory for general unification*, volume 11 of *Progress in computer science and applied logic*. Birkhäuser, 1991.

[Sny93] W. Snyder. A fast algorithm for generating reduced ground rewriting systems from a set of ground equations. *Journal of Symbolic Computation*, 15:415–450, 1993.