

Basic Completion with E-cycle Simplification*

Christopher Lynch

Department of Mathematics and Computer Science
Clarkson University
Box 5815, Potsdam, NY 13699-5815, USA
Christopher.Lynch@clarkson.edu

Christelle Scharff

LORIA
BP 239, 54506 Vandoeuvre-les-Nancy Cedex, France
Christelle.Scharff@loria.fr

Abstract. We give a new simplification method, called E-cycle Simplification, for Basic Completion inference systems. We prove the completeness of Basic Completion with E-cycle Simplification. We prove that E-cycle Simplification is strictly stronger than the only previously known complete simplification method for Basic Completion, Basic Simplification, in the sense that every derivation involving Basic Simplification is a derivation involving E-cycle Simplification, but not vice versa. E-cycle Simplification is simple to perform, and does not use the reducibility-relative-to condition. *ECC* implements our method.

Keywords: Equational constraints, Basic Completion, Simplification Strategies.

1. Introduction

In automated theorem proving, it is important to know if an inference system is complete, because a complete inference system guarantees that a proof will be found if one exists and if it halts without a proof, then the theorem is false. However, in practice, incomplete inference systems are often used because complete ones are not efficient.

An example of this phenomenon is the case of Basic Completion [BGLS95, NR92]. This is a restriction on Knuth-Bendix Completion [KB70] such that the most general unifier is saved

*This work was supported by NSF grant number CCR-9712388 and partially done during a visit in the PROTHEO group in Nancy.

as a constraint, instead of being applied to the conclusion of an inference [KKR90a]. The effect of this restriction is that much of a term is stored in a constraint, and therefore the variable positions appear closer to the root than in the non-basic case, or else variable positions occur where there are no variable positions in the non-basic case. In Knuth-Bendix Completion, there is a restriction that inferences are not allowed at variable positions. This restriction then becomes much more powerful in Basic Completion. In [NR92, BGLS95], it was shown that Basic Completion is complete.

Simplification rules are crucial in any form of completion. However, in [NR92] it was shown that the combination of Basic Completion and Standard Simplification is not complete (see [BGLS95] for more incompleteness examples). In [BGLS95], a new form of simplification, called *Basic Simplification*, is shown to be complete in combination with Basic Completion. Unfortunately, Basic Simplification can only be performed under certain circumstances. So, to retain completeness, a theorem prover must either not simplify under these circumstances, or else apply the constraint of the simplifying equation before simplifying. The first solution is unsatisfactory because it does not allow as much simplification. The second solution is unsatisfactory because it removes the advantages of Basic Completion. There is a third solution, which is to allow the simplification without applying the constraint. Of course, this solution potentially causes a loss of completeness. In [BGLS92], a modification to OTTER [McC94] was made to make it perform Basic Paramodulation, and a series of experiments was run. In every case, the theorem was proved using the incomplete strategy of Basic Paramodulation with Standard Simplification. Basic Paramodulation (modulo AC) is also implemented in [Vig97]. More recently, McCune has developed a theorem prover called EQP [McC96], based on Basic Paramodulation, using an incomplete simplification strategy. He recently solved the Robbins Algebra problem [Kol96] [McC97b], and he believes that Basic Paramodulation was a key strategy in finding the proof [McC97a].

These results lead us to an analysis of simplification strategies for Basic Completion. The goal is to understand when simplification will destroy completeness and when it will not. We provide an abstract setting to develop and prove the completeness of a concrete simplification method for Basic Completion, called *E-cycle Simplification*, which does not use the reducibility-relative-to condition of Basic Simplification. We prove that E-cycle Simplification is complete and strictly stronger than Basic Simplification, in the sense that every derivation involving Basic Simplification is a derivation involving E-cycle Simplification, but not vice versa (see section 5). Also, there are many examples where E-cycle Simplification may be performed but Basic Simplification may not (see section 5 for an example) .

The idea behind E-cycle Simplification is simple. No equation may simplify one of its *ancestors*¹. In the inference procedure we build a *dependency (directed) graph*. The nodes of the dependency graph are labelled by the equations. When we deduce a new equation, we add a node to the graph labelled by the new equation to show dependencies and ancestors. A Basic Critical Pair inference adds an *Inference* edge to indicate that the conclusion depends on the

¹We define the notion of *ancestor* in the paper.

premises if it has an irreducible constraint. A Simplification adds *Simplification* edges. When the rule deduces a constrained equation, *Constraint* edges are added from the constrained equation to its original ancestors in E , the set to complete. These dependencies are only needed if the constraint of the equation is reducible, to be able to create a reduced version of the constrained equation. Edges are associated with reducibility constraints, which may conflict with each other. We define *E-paths* and *E-cycles* in the dependency graph to be paths and cycles with no conflict in reducibility constraints. E-cycles may only occur when an equation simplifies an ancestor. Whenever a simplification would create an E-cycle in the dependency graph, we disallow the simplification.

Our completeness proof is based on the model construction proof of [BG94], which is also used in the completeness proofs of Basic Completion in [NR92, BGLS95]. Like those proofs, we build a model of irreducible equations, based on an ordering of the equations. The difference is that we do not use the multiset extension of the term ordering but a different ordering \succ_g directly based on the dependency graph. If there is an edge from a node labelled with equation e_1 to a node labelled with equation e_2 , then e_1 is larger than e_2 in our ordering \succ_g and we write $e_1 \succ_g e_2$. The ordering \succ_g is well-founded, because the dependency graph does not have any E-cycles or infinite E-paths.

The paper is organized as follows. Section 2 contains some definitions and notions useful for the comprehension of the paper. Section 3 defines dependency graphs and E-cycles. Section 4 defines E-cycle Simplification and the construction of dependency graphs. Then, in section 5, we prove that Basic Completion with E-cycle Simplification is complete. In section 6, we show that E-cycle Simplification is strictly more powerful than Basic Simplification. Before the conclusion section, we show, in section 7, the incompleteness of some simplification methods. The interest of this section is to try to find the line that divides complete simplification strategies from incomplete ones. We think that E-cycle Simplification is “close to the line”.

The conference version of this paper is available in [LSc98a]. For lack of space, some proofs are only available in [LSc98b].

2. Preliminaries

Equational Logic: We assume the reader is familiar with the notation of equational logic and rewriting. A survey of rewriting is available in [DJ90]. We only define important notions for the comprehension of the paper and new notions and definitions we introduce.

An *equational constraint* φ is a conjunction $s_1 =? t_1 \wedge \dots \wedge s_n =? t_n$ of syntactic equations $s_i =? t_i$. \top is the true equational constraint and \perp is the false constraint. The symbol \approx is a binary symbol, written in infix notation, representing semantic equality. In this paper \succeq_t refers to an *ordering* on terms (\succ_t in its strict version), which is a well-founded reduction ordering total on ground terms. \approx is symmetric and, when we write the equality $s \approx t$, we assume that $s \not\prec_t t$. We extend the ordering \succeq_t to ground equations and we call the new ordering \succeq_e (\succ_e in its strict version). Let $s \approx t$ and $u \approx v$ be two ground equations. We define the ordering \succ_e such that

$s \approx t \succ_e u \approx v$ if either $s \succ_t u$ or, $s = u$ and $t \succ_t v$ ². A pair $s \approx t \llbracket \varphi \rrbracket$ composed of an equation $s \approx t$ and an equational constraint φ is called a *constrained equation*. An equation $s\sigma \approx t\sigma$ is a ground *instance* of a constrained equation $s \approx t \llbracket \varphi \rrbracket$ if σ is a ground substitution solution of φ . We say that $\sigma \in \text{Sol}_G^{s \approx t}(\varphi)$. We denote by $\text{Gr}(e \llbracket \varphi \rrbracket)$ the set of ground instances of an equation $e \llbracket \varphi \rrbracket$. This is extended to a set E by $\text{Gr}(E) = \bigcup_{e \in E} \text{Gr}(e)$. We call $e\sigma_1 \llbracket \varphi_2 \rrbracket$ a retract form of a constrained equation $e \llbracket \varphi \rrbracket$ if $\sigma = \text{mgu}(\varphi)$, $\sigma_2 = \text{mgu}(\varphi_2)$ and $\forall x \in \text{Dom}(\sigma), x\sigma = x\sigma_1\sigma_2$. For example, $g(f(y)) \approx b \llbracket y = ? a \rrbracket$ is a retract of $g(x) \approx b \llbracket x = ? f(a) \rrbracket$.

Reducibility Constraints: We define a predicate symbol *Red* which is applied to two parameters, a term t , and a ground equation e or \top . The second parameter of *Red* is only useful in section 5. Let E be a set of equations.

A *reducibility constraint* is (1) \top denoting the empty conjunction and the true reducibility constraint or (2) \perp denoting the false reducibility constraint or (3) of the form $\varphi_{r_1} \wedge \cdots \wedge \varphi_{r_n}$, where φ_{r_i} is of the form $(\bigvee_j \text{Red}(t_j, e_j))$ or $\neg \text{Red}(t_j, e_j)$ or \top where $t_j \in \mathcal{T}$ and $e_j \in \text{Gr}(E) \cup \{\top\}$. First instances of reducibility constraints can be found in [Pet94] and in [LS95].

A *ground reducibility constraint* is a reducibility constraint such that the first parameter of the predicate *Red* is a ground term. Let φ_r be a ground reducibility constraint, and R be a ground rewrite system. Then φ_r is *satisfiable in R* , if and only (1) $\varphi_r = \top$, or (2) $\varphi_r = \text{Red}(t, \top)$ and t is reducible in R , or (3) $\varphi_r = \text{Red}(t, e)$ and t is reducible by an equation e' of R such that $e' \prec_e e$, or (4) $\varphi_r = \neg \varphi'_r$ and φ'_r is not satisfiable in R , or (5) $\varphi_r = \varphi'_r \wedge \varphi''_r$ and φ'_r and φ''_r are satisfiable in R , or (6) $\varphi_r = \varphi'_r \vee \varphi''_r$ and φ'_r is satisfiable in R or φ''_r is satisfiable in R .

Notice that for any ground rewrite system R and for any ground equation e with t smaller than the maximal term in e , $\text{Red}(t, e)$ is satisfiable in R if and only if $\text{Red}(t, \top)$ is satisfiable in R . Therefore, in this case, we abbreviate $\text{Red}(t, e)$ by $\text{Red}(t)$. Also, we abbreviate $\text{Red}(t, \top)$ by $\text{Red}(t)$.

A reducibility constraint φ_r is *satisfiable* iff there exists a ground rewrite system R and a ground substitution σ such that $\varphi_r\sigma$ is satisfiable in R . Satisfiability is a semantic notion. In our completion procedure, we deal with syntactic objects. For that, we need the notion of *consistency*. A reducibility constraint φ_r is *inconsistent* if and only if (1) $\varphi_r = \perp$ or (2) there exist terms $u_1[t_1\sigma_1], \dots, u_n[t_n\sigma_n]$, equations e_1, \dots, e_n and e'_1, \dots, e'_n such that, for $i \in \{1, \dots, n\}$, σ_i are substitutions and $(\bigvee_{i \in \{1, \dots, n\}} \text{Red}(t_i, e_i))$ appears in φ_r and $\neg \text{Red}(u_i[t_i\sigma_i], e'_i)$ appear in φ_r and $e_i \preceq_e e'_i$, or (3) there exist $u_1[t_1\sigma_1], \dots, u_n[t_n\sigma_n]$, equations e_1, \dots, e_n and $v_1 \approx w_1, \dots, v_n \approx w_n$ such that, for $i \in \{1, \dots, n\}$, σ_i are substitutions and $(\bigvee_{i \in \{1, \dots, n\}} \text{Red}(t_i, e_i))$ appears in φ_r and $\neg \text{Red}(u_i[t_i\sigma_i], v_i \approx w_i)$ appear in φ_r and $v_i \succ_t t_i$. A reducibility constraint is *consistent* if and only if it is not inconsistent.

Note that it is simple to test if a reducibility constraint is consistent using this definition. There is a close relationship between satisfiability and consistency. We prove that a satisfiable reducibility constraint is consistent.

Theorem 2.1. *Let φ_r be a reducibility constraint. If φ_r is satisfiable, then φ_r is consistent.*

²Recall that $s \not\prec_t t$ and $u \not\prec_t v$

Definition 2.1. Let φ be an equational constraint. We define $RedCon(\varphi)$ as the reducibility constraint $\bigvee\{Red(x\sigma) \mid x \in Dom(\sigma) \text{ and } \sigma = mgu(\varphi)\}$. In particular, $RedCon(\top) = \perp$.

Inference Systems: Our inference system is based on Basic Completion. The main inference rule of Basic Completion is the *Basic Critical Pair* inference rule:

Basic Critical Pair

$$\frac{u[s'] \approx v[\varphi_1] \quad s \approx t[\varphi_2]}{u[t] \approx v[s = ? s' \wedge \varphi_1 \wedge \varphi_2]} \quad \text{if :}$$

- s' is not a variable,
- there exists a substitution σ such that $\sigma \in Sol((s = ? s') \wedge \varphi_1 \wedge \varphi_2)$, $s\sigma \not\approx_t t\sigma$ and $u[s']\sigma \not\approx_t v\sigma$.

The *Critical Pair* inference rule is the rule where $\varphi_1 = \varphi_2 = \top$ and the conclusion equation is $u[t\sigma] \approx v$ where $\sigma = mgu(s = ? s')$. The Basic Critical Pair rule saves the mgu as a constraint instead of applying it.

The *Standard Simplification* deletion rule is the following:

Standard Simplification

$$\frac{u[s'] \approx v[\varphi_1] \quad s \approx t[\varphi_2]}{u[t\sigma_2\mu] \approx v[\varphi_1 \wedge \varphi_2\mu]} \quad \text{if :}$$

- s' is not a variable,
- there exists substitutions μ , $\sigma_1 = mgu(\varphi_1)$ and $\sigma_2 = mgu(\varphi_2)$ such that $s'\sigma_1 = s\sigma_2\mu$, $s\sigma_2\mu \succ_t t\sigma_2\mu$, and $v\sigma_1 \succ_t t\sigma_2\mu$ if $u = s'$.

Basic Simplification is based on the notion of reducibility-relative-to and is described in [BGLS95]. We call this property *substitution-reducibility-relative-to*.

Definition 2.2. Let $s \approx t$ be a ground instance of an equation $s' \approx t'[\varphi]$. $s \approx t$ is of the form $s'\sigma \approx t'\sigma$ where $\sigma \in Sol_G^{s' \approx t'}(\varphi)$. We say that $s \approx t$ is *substitution reduced* with respect to a rewrite system R , if for all x in $Dom(\sigma) \cap Var(s' \approx t')$, $x\sigma$ is irreducible with respect to R . A non ground equation $s \approx t[\varphi]$ is *substitution reduced* with respect to a rewrite system R , if all its ground instances $s\sigma \approx t\sigma$ where $\sigma \in Sol_G^{s \approx t}(\varphi)$ are substitution reduced with respect to R .

$e_1[\varphi_1]$ is *substitution reduced relative to* $e_2[\varphi_2]$ modulo a substitution ρ if, for all rewrite systems R and for all ground instances of $e_2[\varphi_2]$, $e_2\sigma_2\sigma$ where $\sigma_2 = mgu(\varphi_2)$ and $\sigma_2\sigma \in Sol_G^{e_2}(\varphi_2)$, such that $e_2\sigma_2\sigma$ is substitution reduced with respect to R , then $e_1\sigma_1\rho\sigma$, where $\sigma_1 = mgu(\varphi_1)$ and $\sigma_1\rho\sigma \in Sol_G^{e_1}(\varphi_1)$, is substitution reduced with respect to R .

The property of substitution-reducibility-relative-to-modulo is quite general but the following simple sufficient condition implies substitution-reducibility-relative-to-modulo. An equation $e_1[\varphi_1]$ is *substitution reduced relative to* $e_2[\varphi_2]$ modulo ρ , if for all term $t \in Ran(\sigma_1)$, where $\sigma_1 = mgu(\varphi_1)$, there exists $y \in Dom(\sigma_2) \cap e_2$ such that $t\rho$ is a subterm of $y\sigma_2$, where $\sigma_2 = mgu(\varphi_2)$.

The *Basic Simplification* deletion rule is the following:

Basic Simplification

$$\frac{u[s'] \approx v[\varphi_1] \quad s \approx t[\varphi_2]}{u[t\sigma_2\mu] \approx v[\varphi_1 \wedge \varphi_2\mu]} \quad \text{if :}$$

- s' is not a variable,
- there exists a match μ , $\sigma_1 = mgu(\varphi_1)$ and $\sigma_2 = mgu(\varphi_2)$ such that $s'\sigma_1 = s\sigma_2\mu$, $s\sigma_2\mu \succ t\sigma_2\mu$, and $v\sigma_1 \succ t\sigma_2\mu$ if $u = s'$, and
- $s \approx t[\varphi_2]$ is substitution reduced relative to $u[s'] \approx v[\varphi_1]$ modulo μ .

There are two optional but useful rules. If the conditions for the application of Basic Simplification are not true, it is possible to apply the *Retraction* rule which consists of retracting the "from" equation of the rule to make the application of Basic Simplification possible. *Basic Blocking* is a deletion rule based on the substitution-reducibility-relative-to condition that deletes an equation with a reducible constraint.

The *Retraction* rule is the following:

Retraction

$$\{u[s] \approx v[\varphi]\} \cup \Gamma \rightarrow \{u[s] \approx v\}\sigma[\varphi'] \cup \Gamma \text{ if:}$$

- Γ is a set of equations, and $(u[s] \approx v)\sigma[\varphi']$ is a retract form of $u[s] \approx v[\varphi]$.

The *Basic Blocking* deletion rule is the following:

Basic Blocking

$$\{u \approx v[\varphi_1[s']], s \approx t[\varphi_2]\} \cup \Gamma \rightarrow \{s \approx t[\varphi_2]\} \cup \Gamma \text{ if:}$$

- Γ is a set of equations,
- s' is not a variable,
- there exists a match μ and $\sigma_2 = mgu(\varphi_2)$ such that $s' = s\sigma_2\mu$, and
- $s \approx t[\varphi_2]$ is substitution reduced relative to $u \approx v[\varphi_1[s']]$ modulo μ .

Standard Completion is the inference system designed using the Critical Pair inference rule and the Standard Simplification deletion rule. In that case, there are no constraints, so $\varphi_1 = \varphi_2 = \top$. *Standard Completion* is complete. We call *BCBS* the Basic Completion inference system consisting of the Basic Critical Pair inference rule, the Basic Simplification deletion rule plus the Basic Blocking deletion rule and the Retraction rule. *BCBS* is complete. In this paper, we give a new Basic Completion inference system **BCES** which uses the Basic Critical Pair inference rule, a restricted version of the **Standard Simplification** deletion rule, that we call the *E-cycle Simplification* deletion rule plus the *E-cycle Blocking* deletion rule. *Retraction* is added to *BCES* in [LSc98b].

3. The dependency graph and E-cycles

In this section, we describe the framework used in the paper. We first define the *dependency graph* of a set of unconstrained equations E to complete and then we give the definition of an E-cycle, a cycle that can be detected in the dependency graph. A graph G is $G = (V, ED)$, where V is a set of vertices and ED is a set of edges.

The dependency graph is a directed graph. The vertices of the dependency graph are labelled by equations. We associate a set of vertices $C_{ancestor}(v)$ to each vertex. There are three kinds

of edges in the dependency graph: C edges, I edges and S edges. C stands for Constraint, I stands for Inference and S stands for Simplification. Each edge has a reducibility constraint associated with it determined by the type of the edge (C , I and S) and the constraints of equations labelling the vertices at the extremities of the edge.

Let e_d be an edge from a vertex v_1 labelled by an equation $e_1[\varphi_1]$ to a vertex v_2 labelled by an equation $e_2[\varphi_2]$ in the dependency graph. If e_d is a C edge, then the constraint associated with e_d is $RedCon(\varphi_1)$. e_d is denoted by (v_1, v_2, C) . If e_d is an I edge, the constraint associated with e_d is $\neg RedCon(\varphi_2)$. e_d is denoted by (v_1, v_2, I) . If e_d is an S edge, then the constraint associated with e_d is \top . e_d is denoted by (v_1, v_2, S) .

An E -path is a path of C , I and S edges in the dependency graph such that the conjunction of the reducibility constraints associated to the edges is consistent. An E -cycle is an E -path which begins and ends at the same vertex and which contains at least a C and an S edge. The problem of finding an E -path and so an E -cycle in the dependency graph is NP-complete [HM98].

We can notice that E -paths and also E -cycles do not contain an I edge followed by a C edge. The proof follows directly from the characterization of reducibility constraints of I and C edges in the dependency graph.

Theorem 3.1. *No E -path contains an I edge followed by a C edge.*

4. Construction of the dependency graph and E -cycle Simplification

At the beginning of the Basic Completion process, the initial set E is represented by the initial dependency graph G_{init} that is defined as follows. Each equation of the set of equations E to complete is a label of a vertex of the initial dependency graph $G_{init} = (V_{init}, ED_{init})$ and $ED_{init} = \emptyset$. $C_ancestor(v) = \{v\}$ for all $v \in V_{init}$. When an inference of $BCES$ is performed, the dependency graph is updated. A new vertex labelled by the conclusion of the inference is added and edges are added.

The E -cycle Simplification rule is the same as the Standard Simplification rule, with the added condition that the addition of S edges from the "into" premise to the "from" premise and from the "into" premise to the conclusion equation does not create an E -cycle. The E -cycle Blocking rule is the same as the Basic Blocking rule, with the last condition replaced by the condition that the addition of an S edge from the vertex labelled by $u \approx v[\varphi_1[s']]$ to the vertex labelled by $s \approx t[\varphi_2]$ does not create an E -cycle. We explain how Basic Critical Pair inferences and these rules update the dependency graph using *Graph Transitions*.

Definition 4.1. A *Graph Transition* is denoted by $(E_i, G_i) \rightarrow (E_{i+1}, G_{i+1})$, where E_i and E_{i+1} are sets of equations such that E_{i+1} is obtained from E_i by performing a Basic Critical Pair Inference or a deletion rule³ and $G_i = (V_i, ED_i)$ and $G_{i+1} = (V_{i+1}, ED_{i+1})$ are dependency graphs such that G_{i+1} is obtained from G_i by:

³A simplification rule consist of a Critical Pair inference that adds an equation plus a deletion rule.

- A Basic Critical Pair Inference.
We have the following Graph Transition $(\{e_0, e_1\} \cup \Gamma, G_i) \rightarrow (\{e_0, e_1, e_2\} \cup \Gamma, G_{i+1})$ where e_0 is the "into" equation, e_1 is the "from" equation and e_2 is the conclusion equation of the Basic Critical Pair inference. Let e_0 be the label of v_0 and e_1 be the label of v_1 .
 - $V_{i+1} = V_i \cup \{v_2\}$ such that $label(v_2) = e_2$
 - $ED_{i+1} = ED_i \cup E_C \cup E_I$ where:
 $E_C = \emptyset$ if e_2 is an unconstrained equation,
otherwise $E_C = \bigcup_{v \in C_ancestor(v_0)}(v_2, v, C) \cup \bigcup_{v \in C_ancestor(v_1)}(v_2, v, C)$ and $E_I = \{(v_0, v_2, I)\}$.
 $C_ancestor(v_2) = C_ancestor(v_0) \cup C_ancestor(v_1)$.
- A deletion rule.
We have the following Graph Transition $(\{e_0, e_1, \dots, e_n\} \cup \Gamma, G_i) \rightarrow (\{e_1, e_2, \dots, e_n\} \cup \Gamma, G_{i+1})$ where e_0 is removed because of e_1, \dots, e_n . Let e_i be the label of v_i for $i \in \{0, \dots, n\}$.
 - $V_{i+1} = V_i$
 - $ED_{i+1} = ED_i \cup E_S$ where $E_S = \bigcup_{i \in \{1, \dots, n\}}(v_0, v_i, S)$.

We now summarize the above definition. A C edge is created from a constrained equation to its initial *ancestors*, initial unconstrained equations of E . An I edge is added from the vertex labelled by the "into" premise of an inference to the vertex labelled by the conclusion of the inference. This indicates that the "into" premise depends on the conclusion. An S edge is from the simplified equation to the simplifier, and also from the simplified equation to the conclusion of the simplification. This indicates that the simplified equation depends on the other two. The dependency graph will not contain an E-cycle, because only a S edge could create an E-cycle (see theorem 4.1) and E-cycle Simplification forbids creation of E-cycles.

Definition 4.2. - Given a sequence of graphs G_0, G_1, \dots where $G_i = (V_i, ED_i)$ for all i , the *limit* G_∞ is (V_∞, ED_∞) , where $V_\infty = \bigcup_i \bigcap_{j \geq i} V_j$, $label(v) = \bigcup_i \bigcap_{j \geq i} label(v_j)$ for all $v \in V_\infty$, and $ED_\infty = \bigcup_i \bigcap_{j \geq i} ED_j$.

- A *Graph Transition Derivation* from E is a possibly infinite derivation $(E_0 = E, G_0 = G_{init}) \rightarrow (E_1, G_1) \rightarrow \dots$, where for all i , $(E_i, G_i) \rightarrow (E_{i+1}, G_{i+1})$ is a Graph Transition. The *Transition Limit* is denoted by $T_\infty = (E_\infty, G_\infty)$.

The two following theorems are consequences of the way the dependency graph is constructed. The first theorem proves, in particular, that an E-cycle does not contain only C edges. The second theorem proves that it is only a deletion rule, so the addition of an S edge that could create an E-cycle. It also proves that an E-cycle contains at least an S edge.

Theorem 4.1. *A cycle does not contain only C edges.*

Proof: This is because C edges always go from a constrained equation to an unconstrained initial equation of E , the set of equations to complete. \square

Theorem 4.2. *Let $(E_0 = E, G_0 = G_{init}) \rightarrow (E_1, G_1) \rightarrow \dots \rightarrow (E_{n-1}, G_{n-1}) \rightarrow (E_n, G_n) \dots$ be a Graph Transition Derivation. If G_{n-1} does not contain an E-cycle and G_n contains an E-cycle, then E_n was obtained from E_{n-1} by a deletion rule.*

Proof: For proving that it is only a deletion rule that creates an E-cycle, we prove that a Basic Critical Pair Inference does not create an E-cycle.

Assume that G_n is obtained from G_{n-1} in the graph transition derivation by a Basic Critical Pair inference and that G_{n-1} does not contain an E-cycle. In this case, a new vertex v , C edges and an I edge are added to the dependency graph G_{n-1} to form G_n . C edges are added from vertex v to all vertices v_1, \dots, v_n such that $label(v_i)$, for $i \in \{1, \dots, n\}$, are the initial equations $label(v)$ was deduced from. An I edge is added from the vertex labelled by the "into" equation of the inference to vertex v . The only E-cycle that could be created is the one involving v , the I edge just added, and a C edge just added, but this is not an E-cycle, by theorem 3.1. \square

To illustrate *BCES*, we now develop the counter-example of Nieuwenhuis and Rubio [NR92], that proves that Basic Completion with Standard Simplification is incomplete. We adopt the same execution plan.

Example 4.1. Let $E = \{a \approx b$ (1), $f(g(x)) \approx g(x)$ (2), $f(g(a)) \approx b$ (3)}. We assume a lexicographic path ordering based on the precedence $f \succ_{prec} g \succ_{prec} a \succ_{prec} b$.

The dependency graph for the development of the full example is in figure 1. The saturated set we obtain is $E_\infty = \{a \approx b$ (1), $f(g(x)) \approx g(x)$ (2), $f(b) \approx b$ (5), $f(g(b)) \approx b$ (7), $g(x) \approx b[x = ? b]$ (8)}.

$$1. \quad \frac{f(g(x)) \approx g(x) \text{ (2)} \quad f(g(a)) \approx b \text{ (3)}}{g(x) \approx b[x = ? a] \text{ (4)}}$$

We add C edges from equation (4) to the initial equations (2) and (3). The reducibility constraint associated to these edges is $Red(a)$. We add an I edge from equation (2) to equation (4). The reducibility constraint associated to this edge is $\neg Red(a)$.

$$2. \quad \frac{f(g(a)) \approx b \text{ (3)} \quad g(x) \approx b[x = ? a] \text{ (4)}}{f(b) \approx b \text{ (5)}}$$

We add no C edge because equation (5) is an unconstrained equation. However, the set of initial equations equation (5) depends on is recorded. Equation (5) depends on the initial equations (2) and (3). We add an I edge from equation (3) to equation (5). The reducibility constraint associated to this edge is \top .

Equation (3) can be standard simplified by equation (4). However, there is no E-cycle Simplification. Indeed, if we add S edges, an E-cycle is created. The S edge from (3) to (4) (whose associated reducibility constraint is \top) and the C edge from (4) to (3) (whose associated reducibility constraint is $Red(a)$) describe an E-cycle. If we delete equation (3) as in Standard Simplification, then we cannot construct a confluent system (equation $g(b) \approx b$ has no rewrite proof), therefore the inference system would not be complete. The presence of the E-cycle prevents us from deleting equation (3). Thus we have used the dependency graph to detect incompleteness. The reducibility-relative-to condition of Basic Simplification also detects this. However, that condition also prevents some simplifications that would not cause loss of completeness, which E-cycle Simplification allows.

5. BCES is complete

In this section, we give the completeness result of *BCES*. In the completeness proof, we need to construct a ground dependency graph which is an instance of the dependency graph we created in the previous section. Our proof is based on the model construction proof of [BG94]. The

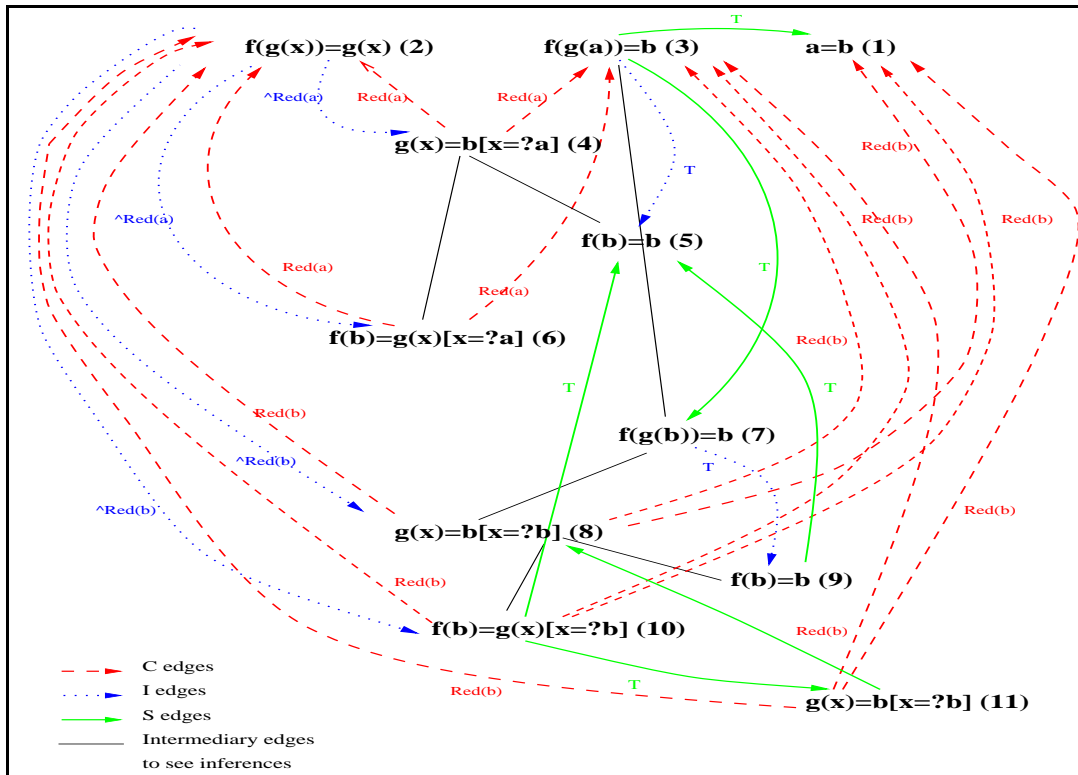


Figure 1 Dependency graph of Basic Completion with E-cycle Simplification of $E = \{a \approx b$ (1), $f(g(x)) \approx g(x)$ (2), $f(g(a)) \approx b$ (3)}

ground dependency graph is used to build a model of irreducible equations and a well-founded ordering \succ_g of the equations.

5.1. The ground dependency graph

In the ground dependency graphs, the labels of vertices are ground equations. Edges are added only if the reducibility constraints associated to them are satisfiable in a particular set of ground equations. In a ground dependency graph, we no longer speak about E-cycle but just about cycle.

We first define GG_{init} , the initial ground dependency graph, for a set of equations E to complete. $GG_{init} = (V_{init}, ED_{init})$, where V_{init} is the set of vertices such that each equation $e \in Gr(E)$ labels one vertex of V_{init} and $ED_{init} = \emptyset$. As in the non-ground case, we set $C_{ancestor}(v) = \{v\}$ for every $v \in V_{init}$. In a ground dependency graph, C edges are added from ground instances of constrained equations to ground instances of unconstrained initial equations. An I edge is added as previously from the "into" premise to the conclusion of an inference. Furthermore, we add an I edge from the "into" premise to the "from" premise of

an inference. We also add I edges from the "into" equation to the "from" and the conclusion equation of inferences at the ground level that simulates "inferences" at a variable position not in the constraint at the non-ground level ⁴.

The consequences of a rule on the ground dependency graph are formalized using *Ground Graph Transitions modulo an equational theory E'* distinguishing the applied rule at the ground level. E' is a set of ground equations with respect to which the reducibility constraints are tested. In the completeness proof, this set is instantiated by $Gr(E_\infty)$. We need to lift from ground level to non-ground level. It is why we speak about Ground Graph Transition Derivation associated to (non-ground) Graph Transition Derivation.

Definition 5.1. Let $(E_0 = E, G_0 = G_{init}) \rightarrow (E_1, G_1) \rightarrow \dots$ be a *Graph Transition Derivation* from E . Then $(GE_0 = Gr(E), GG_0 = GG_{init}) \rightarrow (GE_1 = Gr(E_1), GG_1) \rightarrow \dots$ is its *associated Ground Graph Transition derivation modulo E'* , where for all i , GG_{i+1} is defined as follows:

1. Suppose that E_{i+1} is created from E_i by a Basic Critical Pair Inference.

Then GE_i is $\{e_0[\varphi_0], e_1[\varphi_1]\} \cup \Gamma$, and GE_{i+1} is $\{e_0[\varphi_0], e_1[\varphi_1], e_2[\varphi_2]\} \cup \Gamma$, where Γ is a set of equations, and $e_2[\varphi_2]$ is the conclusion equation of the Basic Critical Pair inference performed between $e_0[\varphi_0]$ as the "into" equation and $e_1[\varphi_1]$ as the "from" equation. Let $e_i[\varphi_i]$ be $s_i \approx t_i[\varphi_i]$ for $i \in \{0, 1, 2\}$.

Let $\Sigma = \{\sigma \mid \sigma \in Sol_G^{e_0}(\varphi_0) \cap Sol_G^{e_1}(\varphi_1)\}$. The set of ground instances of the Basic Critical Pair inference is $\{e_0\sigma, e_1\sigma \mid \sigma \in \Sigma\} \cup \Gamma' \rightarrow \{e_0\sigma, e_1\sigma, e_2\sigma \mid \sigma \in \Sigma\} \cup \Gamma'$, where Γ' contains the ground instances of $e_0[\varphi_0]$ and $e_1[\varphi_1]$, $e_0\sigma'$ and $e_1\sigma''$ such that $\sigma' \notin \Sigma$ and $\sigma'' \notin \Sigma$, plus the members of $Gr(\Gamma)$. For a given σ , $e_0\sigma$ labels vertex u_σ , $e_1\sigma$ labels vertex v_σ , and $e_2\sigma$ labels vertex w_σ .

- $ED_{i+1} = ED_i \cup E_C \cup E_I \cup E_V$ where:

* $E_C = \{(w_\sigma, v, C) \mid \sigma \in \Sigma, v \in C_ancestor(u_\sigma) \cup C_ancestor(v_\sigma), \text{ and } RedCon(\varphi_2\sigma) \text{ is satisfiable in } E'\}$.

* $E_I = \{(u_\sigma, v_\sigma, I), (u_\sigma, w_\sigma, I) \mid \sigma \in \Sigma, e_0\sigma \succ_e e_1\sigma, e_0\sigma \succ_e e_2\sigma \text{ and } \neg RedCon(\varphi_2\sigma) \wedge Red(s_0\sigma, e_0\sigma) \wedge \neg Red(s_1\sigma, e_1\sigma) \text{ is satisfiable in } E'\}$.

* E_V is defined below.

- $C_ancestor(w_\sigma) = w_\sigma \cup C_ancestor(u_\sigma) \cup C_ancestor(v_\sigma)$.

2. Suppose that E_{i+1} is created from E_i by a deletion rule.

Assume E_i is $\{e_0, e_1, \dots, e_n\} \cup \Gamma$ and E_{i+1} is $\{e_1, e_2, \dots, e_n\} \cup \Gamma$, where Γ is a set of equations, such that e_0 is removed in a deletion rule because of e_1, \dots, e_n .

Let Σ be the set of all ground substitutions that are instances of the matching substitution used for the deletion. The set of ground instances of the deletion rule is $\{e_0\sigma, e_1\sigma, \dots, e_n\sigma \mid \sigma \in \Sigma\} \cup \Gamma' \rightarrow \{e_1\sigma, e_2\sigma, \dots, e_n\sigma \mid \sigma \in \Sigma\} \cup \Gamma'$, where Γ' contains the other ground instances of the e_j , $j \in \{0, \dots, n\}$, plus the members of $Gr(\Gamma)$. For all $j \in \{0, \dots, n\}$, for a given σ , let $e_j\sigma$ label $v_{j\sigma}$.

- $V_{i+1} = V_i$

- $ED_{i+1} = (ED_i - E_I) \cup E_S$ where:

* $E_I = \{e_d \mid e_d \text{ is an } I \text{ edge of } ED_i \text{ from an "into" equation to a "from" equation of an inference followed by an } S \text{ edge of } E_S\}$

⁴At the non-ground level, no inference is performed at a variable position. However, we refer to it as an inference. At the ground level, the inference must be performed. This remark applies to the rest of section 5.

$$* E_S = \bigcup_{j \in \{1, \dots, n\}} \{(v_{0\sigma}, v_{j\sigma}, S) \mid e_j\sigma \in \Sigma\}.$$

3. E_V is the union of all E_I , for every “Critical Pair inference” at a variable position not in a constraint, where the E_I is defined as follows:

Consider a “Critical Pair inference” at a variable position not in the constraint between an instance of $e_0[\varphi_0]$ as ”into” equation and $e_1[\varphi_1]$ as ”from” equation of the inference. $e_2[\varphi_2]$ is the conclusion equation. Let $e_i[\varphi_i]$ be $s_i \approx t_i[\varphi_i]$ for $i \in \{0, 1, 2\}$. Let $e_0\sigma, e_1\sigma$ and $e_2\sigma$ be ground instances of equations $e_0[\varphi_0]$, $e_1[\varphi_1]$ and $e_2[\varphi_2]$ such that $\sigma \in \text{Sol}_G^{e_0}(\varphi_0) \cap \text{Sol}_G^{e_1}(\varphi_1)$. For a given σ , $e_0\sigma$ is the equation labelling vertex u_σ , $e_1\sigma$ is the equation labelling vertex v_σ and $e_2\sigma$ is the equation labelling vertex w_σ . Then $E_I = \{(u_\sigma, w_\sigma, I), (u_\sigma, v_\sigma, I) \mid \sigma \in \text{Sol}_G^{e_0}(\varphi_0) \cap \text{Sol}_G^{e_1}(\varphi_1), e_0\sigma \succ_e e_1\sigma, e_0\sigma \succ_e e_2\sigma \text{ and } \neg \text{RedCon}(\varphi_2\sigma) \wedge \text{Red}(s_0\sigma, e_0\sigma) \wedge \neg \text{Red}(s_1\sigma, e_1\sigma) \text{ is satisfiable in } E'\}$.

5.2. Completeness proofs

In this section, we first give completeness results concerning the Ground Graph Transition Derivations and then completeness results concerning *BCES*. But first, we give some lemmas describing properties of ground dependency graphs. These properties follow from the construction of ground dependency graphs.

The following theorem provides the result that a cycle in a ground dependency graph contains at least a *C* and an *S* edge. The proofs are done by contradiction.

Theorem 5.1. *Let $(EG_0 = Gr(E), GG_0 = GG_{init}) \rightarrow (EG_1, GG_1) \rightarrow \dots \rightarrow (EG_n, GG_n) \dots$ be a Ground Graph Transition Derivation modulo E' . If GG_n contains a cycle, then this cycle contains at least (a) a *C* and (b) an *S* edge.*

Proof: Let C_g be a cycle present in GG_n .

(a) C_g does not contain only *I* and *S* edges, because *I* and *S* edges always go to smaller equations w.r.t \prec_e in the ground dependency graph. So *I* and *S* edges cannot form a cycle in particular C_g . The contradiction proves that C_g contains at least a *C* edge.

(b) If C_g contains no *S* edges, then, by theorem 4.1 and by (a), it must contain *C* and *I* edges. Therefore, C_g must contain a subpath consisting of an *I* edge followed by a *C* edge. But theorem 3.1 shows this cannot happen⁵. The contradiction proves that C_g does not contain only *C* and *I* edges. So C_g contains at least an *S* edge. \square

The following lemma proves that if an *I* edge goes from a vertex v_1 to a vertex v_2 in GG_∞ , then $\text{label}(v_1)$ is reducible in $Gr(E_\infty)$.

Lemma 5.1. *Let $(EG_0 = Gr(E), GG_0) \rightarrow (EG_1, GG_1) \rightarrow \dots \rightarrow (EG_n, GG_n) \dots$ be a Ground Graph Transition Derivation modulo $Gr(E_\infty)$. If there is an edge from a vertex v_1 to a vertex v_2 in GG_∞ , then $\text{label}(v_1)$ is reducible in $Gr(E_\infty)$.*

Proof: Let $\text{label}(v_1) = s_1\sigma \approx t_1\sigma$ such that $s_1\sigma \approx t_1\sigma$ is a ground instance of an equation $s_1 \approx t_1[\varphi_1]$. If there is an *I* edge from v_1 to v_2 in GG_∞ , then $\text{Red}(s_1\sigma, s_1\sigma \approx t_1\sigma)$ is satisfiable in $Gr(E_\infty)$. So $s_1\sigma$ is reducible in $Gr(E_\infty)$ and so is $s_1\sigma \approx t_1\sigma$. \square

⁵Strictly speaking, theorems 3.1 and 4.1 were given for the non-ground case. But, by the same logic, they apply for Ground Graph Transitions also.

The following lemma proves that a cycle in a ground dependency graph does not contain an I edge from an "into" equation to a "from" equation of an inference. The proof is done by contradiction.

Lemma 5.2. *Let $(EG_0 = Gr(E), GG_0 = GG_{init}) \rightarrow (EG_1, GG_1) \rightarrow \dots \rightarrow (EG_n, GG_n) \dots$ be a Ground Graph Transition Derivation modulo $Gr(E_\infty)$. If GG_i is a dependency graph containing a cycle C_g , then C_g does not contain an I edge from an "into" equation to a "from" equation of an inference.*

The following lemma proves that if there is a cycle or an infinite path at the ground level then there is an E-cycle at the non-ground level. For the proof of this lemma, we basically need to show that the extra edges we added to the graph in the ground case do not create any cycles that do not already exist at the non-ground level. In particular, lemma 5.2 and theorem 5.1 are used.

Lemma 5.3. *Let $(E_0 = E, G_0 = G_{init}) \rightarrow (E_1, G_1) \rightarrow \dots \rightarrow (E_n, G_n) \dots$ be a Graph Transition Derivation and $(EG_0 = Gr(E), GG_0 = GG_{init}) \rightarrow (EG_1, GG_1) \rightarrow \dots \rightarrow (EG_n, GG_n) \dots$ its associated Ground Graph Transition Derivation modulo $Gr(E_\infty)$, then for all i , (a) if GG_i contains a cycle then G_i contains an E-cycle and (b) if GG_i contains an infinite path, then G_i contains an E-cycle.*

Proof: (a) Let C_g be a cycle of GG_i . Let e_{d_i} be the edges of C_g for $i \in \{1, \dots, n\}$. We can notice using lemma 5.2 that e_{d_i} cannot be an I edge from an "into" equation to a "from" equation of an inference. For $i \in \{1, \dots, n-1\}$, e_{d_i} is from equation e_i to equation e_{i+1} and e_{d_n} is from equation e_n to equation e_0 . For each edge e_{d_i} , the reducibility constraint associated to it is satisfiable in $Gr(E_\infty)$.

The cycle C_g corresponds to an E-path C' at the non-ground level. The only edges that will not appear at the non-ground level are edges from an "into" to a "from" equation at the ground level, or edges from an "into" equation to a conclusion equation of an inference at a variable position not in a constraint. The first type of edges will not exist in C_g by lemma 5.2. The second type would be loops at the non-ground level, because the "into" equation and conclusion equation are the same at the non-ground level. Therefore, C' is a cycle.

e_0 is a ground instance of e'_0 . Since C_g exists, the conjunction of reducibility constraints of edges of C' is consistent. By theorem 5.1, C' contains at least a C and an S edge. Therefore C' is an E-cycle.

(b) We first prove that if GG_i contains an infinite path \mathcal{I}_g then this path contains an infinite number of C edges. Then we prove that G_i contains an E-cycle.

Suppose we have an infinite path containing edges $e_{d_1}, e_{d_2}, e_{d_3}, \dots$ in the ground dependency graph GG_i which contains only finitely many distinct C edges. Since there is a finite number of C edges, there must be a last C edge in the sequence. Let e_{d_n} be this last C edge. Then $e_{d_{n+1}}, e_{d_{n+2}}, \dots$ is an infinite path containing only I and S edges, which is impossible because I and S edges always go to smaller equations w.r.t \prec_e , and \prec_e is well-founded. The contradiction proves that there is an infinite number of distinct C edges in an infinite path of the ground dependency graph GG_i .

A C edge present in GG_i is also present in the dependency graph G_i because of the satisfiability of the reducibility constraint associated to it. However, it is impossible for \mathcal{I}_{ng} , the E-path associated

to \mathcal{I}_g , to contain an infinite number of distinct C edges, because C edges are from constrained equation to initial equations and there are only finitely many initial equations. Indeed, this means that \mathcal{I}_{ng} goes through some node labelled by initial equations an infinite number of times and if we go through a node more than once, then we have a cycle in G_i .

Since \mathcal{I}_g exists, the conjunction of reducibility constraints of edges of \mathcal{I}_{ng} is consistent. Since \mathcal{I}_g contains C and S edges, the cycle in G_i is an E-cycle. \square

We now give definitions and a theorem useful in the proof of completeness.

Definition 5.2. $Irred(Gr(E_\infty))$ is a ground set of oriented equations defined by $Irred(Gr(E_\infty)) = \{s\sigma \rightarrow t\sigma \in Gr(s \approx t[\varphi]) \mid s \approx t[\varphi] \in E_\infty \text{ and } s\sigma \text{ is irreducible by a smaller equation than } s\sigma \approx t\sigma \text{ in } Irred(Gr(E_\infty)) - \{s\sigma \rightarrow t\sigma\}\}$.

$Irred(Gr(E_\infty))$ is convergent. It is ground, left-reduced by construction so it is confluent. Besides it is terminating because rewrite rules can be oriented using \succ_t , so it is convergent.

Theorem 5.2. *Let R and R' be two rewrite systems. If $R \subseteq R'$ such that R is confluent and $R \equiv R'$, then R' is confluent.*

The ordering \succ_g used in the completeness proof is constructed directly from the ground dependency graph GG_∞ as explained in the following definition.

Definition 5.3. Let \succ_g be the ordering such that $e \succ_g e'$ if and only if there is a v and v' in GG_∞ such that $label(v) = e$, $label(v') = e'$, and there is a path in GG_∞ from v to v' .

The ordering may not be total, but it is defined on the equations we use in the completeness proof. GG_∞ contains no infinite path or cycle if we do Basic Completion with E-cycle Simplification (see lemma 5.3) and so \succ_g is well-founded. We define redundancy in terms of this ordering.

Definition 5.4. A ground equation e is g -redundant in a set of ground equations E if there are equations $e_1, \dots, e_n \in E$ such that $e_1, \dots, e_n \models e$ and $e_i \prec_g e$ for all i .

E-cycle Simplification and E-cycle blocking deletion rules are examples of g -redundancy as expressed in the following lemma.

Lemma 5.4. *Let E be a set of unconstrained equations. Let $(E_0 = E, G_0 = G_{init}) \rightarrow (E_1, G_1) \rightarrow \dots \rightarrow (E_n, G_n) \dots$ be a Graph Transition derivation and $(EG_0 = Gr(E), GG_0 = GG_{init}) \rightarrow (EG_1, GG_1) \rightarrow \dots \rightarrow (EG_n, GG_n) \dots$ its associated Ground Graph Transition Derivation modulo $Gr(E_\infty)$ where, GG_∞ does not contain a cycle and an infinite path. Let e be an equation that is E-cycle simplified or E-cycle blocked in some E_i . Then every ground instance e' of e is g -redundant in $Gr(E_\infty)$.*

The completeness of Ground Graph Transitions is expressed in the following theorem.

Theorem 5.3. *Let E be a set of unconstrained equations. Let $(EG_0 = Gr(E), GG_0 = GG_{init}) \rightarrow (EG_1, GG_1) \rightarrow \dots \rightarrow (EG_n, GG_n) \dots$ be a Ground Graph Transition Derivation modulo $Gr(E_\infty)$ where, GG_∞ does not contain a cycle and an infinite path. Then this Ground Graph Transition Derivation is complete in the sense that $Gr(E_\infty)$ is convergent.*

Proof: We will prove that $Gr(E_\infty)$ is convergent by proving that it is terminating (**Step 1**) and that it is confluent (**Step 2**).

Step 1: $Gr(E_\infty)$ is terminating because it can be oriented by \succ_t , which is total on ground terms.

Step 2 : We want to apply theorem 5.2. We will prove that $Gr(E_\infty)$ is confluent by proving that $Irred(Gr(E_\infty)) \subseteq Gr(E_\infty)$ (**Step 2.1**), $Irred(Gr(E_\infty))$ is confluent (**Step 2.2**) and $Irred(Gr(E_\infty)) \equiv Gr(E_\infty)$ (**Step 2.3**).

Step 2.1: $Irred(Gr(E_\infty)) \subseteq Gr(E_\infty)$ (obvious).

Step 2.2: $Irred(Gr(E_\infty))$ is confluent (See after definition 5.2).

Step 2.3: We will prove that $Irred(Gr(E_\infty)) \equiv Gr(E_\infty)$ by proving that $Gr(E_\infty) \models Irred(Gr(E_\infty))$ (**Step 2.3.1**) and that $Irred(Gr(E_\infty)) \models Gr(E_\infty)$ (**Step 2.3.2**).

Step 2.3.1: $Gr(E_\infty) \models Irred(Gr(E_\infty))$ is trivial, because $Irred(Gr(E_\infty)) \subseteq Gr(E_\infty)$.

Step 2.3.2: We will prove that $Irred(Gr(E_\infty)) \models Gr(E_\infty)$ by proving that for all ground rewrite rules $s \rightarrow t$, if $Irred(Gr(E_\infty)) \models s \approx t$, then $Gr(E_\infty) \models s \approx t$. This can be done by proving that for all ground rewrite rules $s \rightarrow t$, if $s \rightarrow t \in Gr(E_\infty)$, then $Irred(Gr(E_\infty)) \models s \approx t$.

Let $s \rightarrow t$ be a rewrite rule, $s'\sigma \rightarrow t'\sigma$ ground instance of an equation $s' \approx t' \llbracket \varphi \rrbracket$ of E_∞ . We will prove that $Irred(Gr(E_\infty)) \models s \approx t$ by induction.

The *Induction Hypothesis* is: For all ground equation e' such that $e' \prec_g s \approx t$, if $e' \in Gr(E_\infty)$, then $Irred(Gr(E_\infty)) \models e'$.

Case 1: Suppose that $s \rightarrow t \in Irred(Gr(E_\infty))$. Trivially, $Irred(Gr(E_\infty)) \models s \approx t$.

Case 2: Suppose that $s \rightarrow t \notin Irred(Gr(E_\infty))$.

Case 2.1: Suppose that σ is irreducible in $Gr(E_\infty)$ and $\varphi\sigma$ is irreducible in $Gr(E_\infty)$.

By the fact that $s \rightarrow t \notin Irred(Gr(E_\infty))$, there exists a smallest rewrite rule $u \rightarrow v$ in $Irred(Gr(E_\infty))$ that reduces s . Then u is irreducible in $Gr(E_\infty)$, because otherwise there is a smaller rewrite rule reducing s . So s is of the form $s = s[u]_p$. As σ is irreducible in $Gr(E_\infty)$, there is a Critical Pair inference which can be lifted to a non-ground inference, since the inference is at a non variable position. This inference is:

$$\frac{s[u] \approx t \quad u \approx v}{s[v] \approx t}$$

Since E_∞ is saturated, there is an I edge from $s[u] \approx t$ to $u \approx v$. This edge cannot be removed, because we cannot add an S edge leading out of $u \approx v$, since $u \approx v$ is irreducible in $Gr(E_\infty)$ (see lemma 5.1).

There are I edges from the vertex labelled by $s[u] \approx t$ to the vertex labelled by $u \approx v$ and also to the vertex labelled by $s[v] \approx t$. $Red(s[u], s[u] \approx t) \wedge \neg Red(u, u \approx v) \wedge \neg RedCon(\varphi\sigma)$ is satisfiable in $Gr(E_\infty)$. So due to the graph ordering \succ_g , we have $s[v] \approx t \prec_g s[u] \approx t$ and $u \approx v \prec_g s[u] \approx t$.

Case 2.1.a: Suppose that $s[v] \rightarrow t \in Gr(E_\infty)$.

From the facts that $s[v] \rightarrow t \in Gr(E_\infty)$ and $s[v] \approx t \prec_g s[u] \approx t$ and by the induction hypothesis, we deduce that $Irred(Gr(E_\infty)) \models s[v] \approx t$. Furthermore, $Irred(Gr(E_\infty)) \models u \approx v$ (because $u \rightarrow v \in Irred(Gr(E_\infty))$), so we have $Irred(Gr(E_\infty)) \models s[u] \approx t$.

Case 2.1.b: Suppose that $s[v] \rightarrow t \notin Gr(E_\infty)$.

$s[v] \rightarrow r$ is not in $Gr(E_\infty)$ because it is g -redundant in $Gr(E_i)$ for an i and so in $Gr(E_\infty)$ (see lemma 5.4). So $s[v] \approx t$ is implied by smaller equations of $Gr(E_\infty)$. There are $e_1 \in Gr(E_\infty), \dots, e_n \in Gr(E_\infty)$ such that $\forall i, e_i \prec_g s[v] \approx t$ and $e_1, \dots, e_n \models s[v] \approx t$.

As $e_i \in Gr(E_\infty)$ and $e_i \prec_g s[u] \approx t$, we can apply the induction hypothesis to e_i . So $e_i \in Gr(E_\infty)$ implies that $Irred(Gr(E_\infty)) \models e_i$. But $e_1, \dots, e_n \models s[v] \approx t$ so by transitivity, $Irred(Gr(E_\infty)) \models s[v] \approx t$.

From the facts that $Irred(Gr(E_\infty)) \models s[v] \approx t$ and $Irred(Gr(E_\infty)) \models u \approx v$ (because $u \rightarrow v \in Irred(Gr(E_\infty))$), we deduce that $Irred(Gr(E_\infty)) \models s[u] \approx t$.

Case 2.2 : Suppose that σ is reducible in $Gr(E_\infty)$ and $\varphi\sigma$ is irreducible in $Gr(E_\infty)$.

As σ is reducible in $Gr(E_\infty)$ and $\varphi\sigma$ is irreducible in $Gr(E_\infty)$, there is a ‘‘Critical Pair inference’’ at the ground level that is a Critical Pair inference at a variable position not in the constraint in the non-ground level. This inference is:

$$\frac{s[u] \approx t \quad u \approx v}{s[v] \approx t}$$

There is an I edge from $s[u] \approx t$ to $u \approx v$. There are I edges from the vertex labelled by $s[u] \approx t$ to the vertex labelled by $u \approx v$ and also to the vertex labelled by $s[v] \approx t$. $Red(s[u], s[u] \approx t) \wedge \neg Red(u, u \approx v) \wedge \neg RedCon(\varphi\sigma)$ is satisfiable in $Gr(E_\infty)$. So due to the graph ordering \succ_g , we have $s[v] \approx t \prec_g s[u] \approx t$ and $u \approx v \prec_g s[u] \approx t$. Since σ is reducible in $Gr(E_\infty)$ and $\varphi\sigma$ is irreducible in $Gr(E_\infty)$, $s[u] \approx t$ and $s[v] \approx t$ are instances of the same equation $s' \approx t'[\varphi]$ and are in $Gr(E_\infty)$.

From the facts that $s[v] \approx t \in Gr(E_\infty)$ and $s[v] \approx t \prec_g s[u] \approx t$ because of the I edges and by the induction hypothesis, we can deduce that $Irred(Gr(E_\infty)) \models s[v] \approx t$. Furthermore, $Irred(Gr(E_\infty)) \models u \approx v$ (because $s[u] \approx t \succ_g u \approx v$ and because of the induction hypothesis), so we have $Irred(Gr(E_\infty)) \models s[u] \approx t$.

Case 2.3 : Suppose that the constraint $\varphi\sigma$ is reducible in $Gr(E_\infty)$.

In this case, there is a Critical Pair inference at the ground level that corresponds to a Critical Pair inference in the constraint at the non-ground level. Equation $s' \approx t'[\varphi]$ is E-cycle blocked.

$s \approx t$ is not in $Gr(E_\infty)$ because it is g-redundant in $Gr(E_i)$ for an i and so in $Gr(E_\infty)$. So $s \approx t$ is implied by smaller equations of $Gr(E_\infty)$. There are $e_1 \in Gr(E_\infty), \dots, e_n \in Gr(E_\infty)$, such that $\forall i, e_i \prec_g s \approx t$. Furthermore, $e_1, \dots, e_n \models s \approx t$.

As $e_i \in Gr(E_\infty)$ and $e_i \prec_g s \approx t$, we can apply the induction hypothesis to e_i . So $e_i \in Gr(E_\infty)$ implies that $Irred(Gr(E_\infty)) \models e_i$. But $e_1, \dots, e_n \models s \approx t$ so by transitivity, $Irred(Gr(E_\infty)) \models s \approx t$. \square

The following main theorem proves the completeness of Basic Completion with E-cycle Simplification. The proof is based on the correspondence between the procedural construction of the non-ground dependency graph presented in section 4 and the abstract construction of the ground dependency graph presented in section 5.

Theorem 5.4. *Basic Completion with E-cycle Simplification is complete.*

Proof: Let $(E_0 = E, G_0 = G_{init}) \rightarrow (E_1, G_1) \rightarrow \dots (E_n, G_n) \rightarrow \dots$ be a Graph Transition Derivation obtained using the inference rules of *BCES* that describes Basic Completion with E-cycle Simplification and $(EG_0 = Gr(E), GG_0 = GG_{init}) \rightarrow (EG_1, GG_1) \rightarrow \dots (EG_n, GG_n) \rightarrow \dots$ be its associated Ground Graph Transition Derivation modulo $Gr(E_\infty)$.

For all i , G_i is a dependency graph without E-cycle, because the set of inference rules *BCES* avoids the creation of E-cycle. So by lemma 5.3, for all i , the ground dependency graph GG_i does not contain any cycle and GG_i does not contain an infinite path. We can then apply theorem 5.3 to conclude that Basic Completion with E-cycle Simplification is complete. \square

6. Comparison with Basic Simplification

In this section, we compare E-cycle Simplification with Basic Simplification. A main point of E-cycle Simplification is that the test of application of an E-cycle Simplification rule is not local as is the case in Basic Simplification but depends on the equations deduced to this point. We prove that if we simplify because of a Basic Simplification then there is no E-cycle in the dependency graph we construct and so there is an E-cycle Simplification. So Basic Simplification is a subset of E-cycle Simplification. The proof of this theorem is based on a series of lemmas that show what patterns of edges can be added to the graph. So we show that the reducibility-relative-to condition never allows an edge to be added that would create an E-cycle. Furthermore, we give an example where we can simplify with E-cycle Simplification and where Basic Simplification does not permit us to simplify.

Lemma 6.1. *Let $(E_0 = E, G_0 = G_{init}) \rightarrow (E_1, G_1) \rightarrow \dots \rightarrow (E_n, G_n) \dots$ be a Graph Transition Derivation such that $E_0 \rightarrow E_1 \rightarrow \dots \rightarrow E_n$ is a derivation of BCBS. Then for all i , G_i contains no E-path consisting of an I edge followed by S edges and then by a C edge.*

Proof: Let G_i be a dependency graph obtained by a Graph Transition Derivation using rules of BCBS, and containing an E-path \mathcal{P} consisting of an I edge e_{inf} followed by S edges and then by a C edge e_{const} .

e_{inf} goes from vertex v_1 to vertex v_2 . The E-path of S edges goes from vertex v_2 to vertex v_n ($n > 2$) and e_{const} goes from vertex v_n to vertex v_{n+1} . We can notice that $n = 2$ is impossible by theorem 3.1. For $i \geq 1$, let e_i be the label of v_i .

Because of Basic Simplification, the equation e_n is substitution reduced relative to equation e_2 . So if $Red(Constraint(e_n))$ is consistent, then $Red(Constraint(e_2))$ is too. The reducibility constraint associated to edge e_{inf} is $\neg Red(Constraint(e_2))$, the reducibility constraint associated to the S edges is \top and the reducibility constraint associated to edge e_{const} is $Red(Constraint(e_n))$. So by the inconsistency of the constraints, e_{inf} and e_{const} cannot be contained in a E-path. So there is a contradiction. So it is impossible to have an E-path consisting of an I edge e_{inf} followed by S edges and then by a C edge e_{const} in G_i . \square

Lemma 6.2. *Let $(E_0 = E, G_0 = G_{init}) \rightarrow (E_1, G_1) \rightarrow \dots \rightarrow (E_n, G_n) \dots$ be a Graph Transition Derivation such that $E_0 \rightarrow E_1 \rightarrow \dots \rightarrow E_n$ is a derivation of BCBS. Then if there is an i such that G_i contains an E-cycle, then this E-cycle does not contain any I edge.*

Proof: Let G_i be a dependency graph obtained by a Graph Transition Derivation using rules of BCBS, and containing an E-cycle \mathcal{C} with an I edge e_{inf} .

\mathcal{C} contains an E-path composed of the I edge e_{inf} followed by S edges and then by a C edge e_{const} , because \mathcal{C} contains at least a C edge by definition, at least an S edge due to theorem 4.2 and because of the fact that an I edge cannot be followed by a C edge due to theorem 3.1. However this is impossible because of lemma 6.1. So there is a contradiction and we conclude that G_i does not contain an I edge. \square

Theorem 6.1. *Let $(E_0 = E, G_0 = G_{init}) \rightarrow (E_1, G_1) \rightarrow \dots \rightarrow (E_n, G_n) \dots$ be a Graph Transition Derivation such that $E_0 \rightarrow E_1 \rightarrow \dots \rightarrow E_n$ is a derivation of BCBS. Then there is no i such that G_i contains an E-cycle.*

Proof: Let G_i be a dependency graph obtained by a Graph Transition Derivation using rules of *BCBS*, and containing an E-cycle \mathcal{C} .

By definition, \mathcal{C} contains at least a C edge and an S edge (see theorem 4.2 respectively). By lemma 6.2, \mathcal{C} does not contain an I edge. So \mathcal{C} contains only C and S edges. Therefore \mathcal{C} contains a subpath \mathcal{P} consisting of a C edge followed by zero or more S edges, followed by a C edge.

Let the vertices of the path \mathcal{P} be $v_1 \cdots, v_n$, and the label of each v_i be e_i . Let the edge between v_1 and v_2 be a C edge. Therefore, e_1 must be a constrained equation, and e_2 must be an unconstrained equation. Since there is an S edge between v_{i-1} and v_i for $i \in \{3, \dots, n-1\}$, we know that each e_i is substitution reduced relative to e_{i-1} for $i \in \{3, \dots, n-1\}$, because we are doing Basic Simplification. Since e_2 is unconstrained, this implies that every e_i is unconstrained for $i \in \{3, \dots, n-1\}$. In particular e_{n-1} is unconstrained. But the fact that there is a C edge from v_{n-1} to v_n implies that e_{n-1} is constrained. This is a contradiction, so \mathcal{C} cannot exist. \square

As a direct corollary, we get that if E_0, E_1, \dots, E_n is a derivation of *BCBS*, then it is also a derivation of *BCES*. Inversely, we provide an example that shows that a derivation of *BCES* is not a derivation of *BCBS*.

Example 6.1. Let $E = \{g(x) \approx f(x) \text{ (1)}, g(a) \approx b \text{ (2)}, h(f(a)) \approx b \text{ (3)}\}$. We assume a lexicographic path ordering based on the precedence $h \succ_{prec} g \succ_{prec} f \succ_{prec} a \succ_{prec} b$. Let us assume the following execution plan using *BCES*.

$$1. \quad \frac{g(x) \approx f(x) \text{ (1)} \quad g(a) \approx b \text{ (2)}}{f(x) \approx b[x = ? a] \text{ (4)}}$$

We add C edges from equation (4) to initial equations (1) and (2). The reducibility constraint associated to these edges is $Red(a)$. We add an I edge from equation (1) to equation (4). The reducibility constraint associated to this edge is $\neg Red(a)$.

$$2. \quad \frac{h(f(a)) \approx b \text{ (3)} \quad f(x) \approx b[x = ? a] \text{ (4)}}{h(b) \approx b \text{ (5)}}$$

The equation $h(f(a)) \approx b$ (3) is E-cycle simplified by equation $f(x) \approx b[x = ? a]$ (4). Indeed, no E-cycle is created when we add S edges from equation (3) to equations (4) and (5). With Basic Simplification, the equation $h(f(a)) \approx b$ cannot be deleted because $f(x) \approx b[x = ? a]$ is not reduced relative to $h(f(a)) \approx b$.

The saturated set is $E_\infty = \{g(x) \approx f(x) \text{ (1)}, g(a) \approx b \text{ (2)}, f(x) \approx b[x = ? a] \text{ (4)}, h(b) \approx b \text{ (5)}\}$

7. Incomplete simplification methods

In this section, we define some other simplification methods and show their incompleteness for Basic Completion. Our intention is to discover where the “line” is drawn between complete and incomplete simplification methods and to give some evidence that our method is “close to this line”.

A simplification step in the Completion process, where an equation $l \approx r$ simplifies an earlier equation $l' \approx r'$ is a *Backward Simplification*. Otherwise, it is called a *Forward Simplification*.

The example of Nieuwenhuis-Rubio [NR92] developed in section 3 shows that Basic Completion with Backward Simplification is not complete. We now provide a counter-example to show that Forward Simplification is not complete.

Example 7.1. Let $E = \{h(f(f(a))) \approx b \text{ (1)}, f(x) \approx x \text{ (2)}\}$. We assume a lexicographic path ordering based on the precedence $h \succ_{prec} f \succ_{prec} a \succ_{prec} b$. From (1) and (3) we deduce $h(x) \approx b \llbracket x =? f(a) \rrbracket$ (3). From (1) and (2) we also deduce $h(f(x)) \approx b \llbracket x =? a \rrbracket$ (4). Using Forward simplification, equation (4) can be simplified by equation (3) to a trivial equation and removed. This destroys completeness, because we cannot prove $h(a) \approx b$, which is true. Using E-cycle simplification, we could not do this simplification. The saturated set obtained using Basic Completion with E-cycle Simplification is $E_\infty = \{h(f(a)) \approx b \text{ (1)}, f(x) \approx x \text{ (2)}, h(x) \approx b \llbracket x =? f(a) \rrbracket \text{ (3)}, h(x) \approx b \llbracket x =? a \rrbracket \text{ (5)}\}$.

This example also disproves some other simplification methods. For instance, in this example, every simplifier that is used has initial ancestors that were not simplified. Let us consider this property. It might be intuitive to think that if we keep that property then completeness is preserved. Indeed, those initial ancestors can be reduced and we can generate a reduced version of the simplifier. However, the previous example disproves this statement.

8. Conclusion

We have presented a new method of Simplification in the Basic Completion of a set of equations E , called E-cycle Simplification. Our approach is easy to understand because it is based on a graph. Indeed, E-cycle Simplification is based on the creation of a dependency graph during the completion process showing the dependencies between equations. It permits us to control completeness of Completion such that, whenever E-cycle Simplification allows a simplification, completeness is preserved. We compare our method with Basic Simplification and prove that Basic Simplification is a strict subset of E-cycle Simplification.

Our method is shown complete using an abstract proof technique based on model construction. We think that this abstract framework is promising in the sense that this method of proof can lead us to an analysis of different simplification strategies from the point of view of completeness in constrained completion procedures. We conjecture that all complete Simplification methods for Basic Completion can be fit into our framework.

The method is implemented in *ECC* (E-cycle Completion) in ELAN [KKV95]. Some implementation and experimental details with the two complete methods of simplification are available at <http://www.loria.fr/~scharff>. In the comparison of the two methods, we noticed that some examples do not terminate with Basic Simplification without retraction but terminate with E-cycle Simplification without retraction.

References

- [BG94] L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217-247, 1994.
- [BGLS92] L. Bachmair and H. Ganzinger and C. Lynch and W. Snyder. Basic Paramodulation and Superposition. *Proceedings 11th International Conference on Automated Deduction, Saratoga Springs (N.Y., USA)*, pages 462-476, 1992.

- [BGLS95] L. Bachmair and H. Ganzinger and C. Lynch and W. Snyder. Basic Paramodulation. *Information and Computation*, 121(2):172-192, 1995.
- [DJ90] N. Dershowitz and J.-P. Jouannaud. *Handbook of Theoretical Computer Science*, volume B, chapter 6: Rewrite Systems, pages 244-320. Elsevier Science Publishers B. V. (North-Holland), 1990. Also as: Research report 478, LRI.
- [HM98] M. Hermann. Constrained Reachability is NP-complete. <http://www.loria.fr/~hermann/publications.html#notes>.
- [KB70] D.E. Knuth and P.B. Bendix. Simple word problems in universal algebras. *Computational Problems in Abstract Algebra*, pages 263-297. Pergamon Press, Oxford, 1970.
- [KKR90a] C. Kirchner and H. Kirchner and M. Rusinowitch. Deduction with symbolic constraints. *Revue d'Intelligence Artificielle*, 4(3):9-52, 1990. Special issue on Automatic Deduction.
- [KKR90b] C. Kirchner and H. Kirchner and M. Rusinowitch. Constrained First Order Logic. *Technical report, Centre de Recherche en Informatique de Nancy*, March 1990.
- [KKV95] C. Kirchner and H. Kirchner and M. Vittek. ELAN V 1.17 User Manual, first edition. *Inria Lorraine & Crin, Nancy (France)*, november 1995.
- [Kol96] G. Kolata. With Major Math Proof, Brute Computers Show Flash of Reasoning Power. *New York Times*, December 1996, page C1.
- [LSc98a] C. Lynch and C. Scharff. Basic Completion with E-cycle Simplification. *Lecture Notes in Artificial Intelligence*, volume 1476, pages 209-221. Springer-Verlag, September 1998, Plattsburgh (USA).
- [LSc98b] C. Lynch and C. Scharff. Basic Completion with E-cycle Simplification. 1998, <http://www.loria.fr/~scharff>.
- [LS95] C. Lynch and W. Snyder. Redundancy criteria for constrained completion. *Theoretical Computer Science*, volume 142, pages 141-177, 1995.
- [McC94] W. W. McCune. Otter 3.0: Reference manual and guide. Argonne National Laboratory, 1994.
- [McC96] W. W. McCune. 33 Basic Test Problems: A Practical Evaluation of Some Paramodulation Strategies. In Robert Veroff, editor, *Automated Reasoning and its Applications: Essays in Honor of Larry Wos*, chapter 5, pages 71-114. MIT Press, 1997.
- [McC97a] W. W. McCune. Solution of the Robbins Problem. *Journal of Automated Reasoning*, 19(3):263-276, 1997.
- [McC97b] W. W. McCune. Well-behaved search and the Robbins Problem. *Invited lecture at the Rewriting Techniques and Applications Conference*, Sitges (Spain), 1997, <http://www.mcs.anl.gov/home/mccune/ar/robbins/>.
- [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.
- [Pet94] G.E. Peterson. Constrained Term-Rewriting Induction with Applications. *Methods of Logic in Computer Science*, 1(4):379-412, 1994.
- [Vig97] L. Vigneron. daTac (system demonstration). *International Conference TABLEAUX'97 - Analytic Tableaux and Related Methods*, Abbaye des Prémontrés, Pont-à-Mousson (France), may 1997.