Basic Completion with E-cycle Simplification

Christopher Lynch * and Christelle Scharff **

* Department of Mathematics and Computer Science Box 5815, Clarkson University, Potsdam, NY 13699-5815, USA - E-mail: Christopher.Lynch@clynch.mcs.clarkson.edu - http://www.clarkson.edu/~clynch [1]

** LORIA - CNRS - INRIA Lorraine BP 239, 54506 Vandoeuvre-les-Nancy Cedex, France - E-mail: Christelle.Scharff@loria.fr - http://www.loria.fr/~scharff

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. We believe this new method captures exactly what is needed for completeness. *ECC* implements our method.

Keywords : Equational Logic, Rewriting, Basic Completion.

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 as a constraint, instead of being applied to the conclusion of an inference [KKR90]. 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 [BGLS95,NR92], 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,

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

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.

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* [2]. 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 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. 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 [BGLS95,NR92]. 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, E-cycles, E-cycle Simplification and the construction of the dependency graphs. In section 4, we show that Basic Completion with E-cycle Simplification is complete. Then, in section 5, we show that E-cycle Simplification is strictly more powerful than Basic Simplification.

The full version of this paper that includes complete details and full proofs is available in [LS97].

2 Preliminaries

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.

[2] We define the notion of *ancestor* in the paper.

Let $=^?$ be a binary infix predicate. An *equational constraint* φ is a conjunction $s_1 =^? t_1 \wedge \ldots \wedge s_n =^? t_n$ of syntactic equality $s_i =^? t_i$. $\top$ is the true equational constraint and $\bot$ is the false constraint. The symbol $\approx$ is a binary symbol, written in infix notation, representing semantic equality. In this paper $\succeq_t$ will refer 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 $t \succ_t v$ [3]. A pair $s \approx t[\![\, \varphi \,]\!]$ 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 $t[\![\, \varphi \,]\!]$ if σ is a ground substitution solution of φ. We denote by $Gr(e[\![\, \varphi \,]\!])$ the set of ground instances of an equation $e[\![\, \varphi \,]\!]$. This is extended to a set E by $Gr(E) = \bigcup_{e \in E} Gr(e)$. We call $e\sigma_1[\![\, \varphi_2 \,]\!]$ a retract form of a constrained equation $e[\![\, \varphi \,]\!]$ if $\sigma = mgu(\varphi)$, $\sigma_2 = mgu(\varphi_2)$ and $\forall x \in Dom(\sigma), x\sigma = x\sigma_1\sigma_2$. For example, $g(f(y)) \approx b[\![\, y =^? a \,]\!]$ is a retract of $g(x) \approx b[\![\, x =^? f(a) \,]\!]$.

Reducibility Constraints: We define a predicate symbol Red, which is applied to a term.

A *reducibility constraint* is:

- $\top$ denoting the empty conjunction and the true reducibility constraint or

- $\bot$ denoting the false reducibility constraint or

- of the form $\varphi_{r_1} \wedge \cdots \wedge \varphi_{r_n}$, where φ_{r_i} is of the form $(\bigvee_j Red(t_j))$ or $\neg Red(t)$ or $\top$ where $t,\ t_j \in \mathcal{T}$ (where $\mathcal{T}$ is the set of terms built on a particular signature).

The syntax of the Red predicate is extended in [LS97]. First instances of reducibility constraints can be found in [Pet94] and in [LS95].

A *ground reducibility constraint* is a reducibility constraint such that the 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 if one of the following conditions is true:

- $\varphi_r = \top$.

- $\varphi_r = Red(t)$ and t is reducible in R.

- $\varphi_r = \neg\varphi'_r$ and φ'_r is not satisfiable in R.

- $\varphi_r = \varphi'_r \wedge \varphi''_r$ and φ'_r and φ''_r are satisfiable in R.

- $\varphi_r = \varphi'_r \vee \varphi''_r$ and φ'_r is satisfiable in R or φ''_r is satisfiable in R.

A reducibility constraint φ_r is *satisfiable* iff there exists a rewrite system R and a ground substitution σ such that $\varphi_r\sigma$ is satisfiable in R. Satisfiability is a semantic notion. In our inference procedure, we deal with syntactic objects. For that, we need the notion of *consistency*.

Definition 1. A reducibility constraint φ_r is *inconsistent* if and only if, $\varphi_r = \bot$ or there exist $u_1[t_1\sigma_1], \cdots, u_n[t_n\sigma_n]$ such that σ_i are substitutions and $(\bigvee_{i \in \{1,\cdots,n\}} Red(t_i))$ appears in φ_r and $\neg Red(u_i[t_i\sigma_i])$ $(i \in \{1, \cdots, n\})$ appear in φ_r.

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 consistency and satisfiability.

Theorem 2. *Let φ_r be a reducibility constraint. If φ_r is satisfiable, then φ_r is consistent.*

[3] Recall that $s \not\prec t$ and $u \not\prec v$

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

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

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

– s' is not a variable,

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

Let Γ be a set of equations. This inference means that the set of equations $\{u[s'] \approx v[\![\varphi_1]\!], s \approx t[\![\varphi_2]\!]\} \cup \Gamma$ is transformed to $\{u[s'] \approx v[\![\varphi_1]\!], s \approx t[\![\varphi_2]\!], u[t\sigma] \approx v[\![\, s =^? s' \wedge \varphi_1 \wedge \varphi_2 \,]\!]\} \cup \Gamma$ [4].

We now present *Standard Simplification* and *Basic Simplification* deletion rules.

The *Standard Simplification* deletion rule is the following:

Standard Simplification

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

– s' is not a variable,

– there exists a substitution 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 $u = s'$.

Let Γ be a set of equations. In this rule, the set of equations $\{u[s'] \approx v[\![\varphi_1]\!], s \approx t[\![\varphi_2]\!]\} \cup \Gamma$ is transformed to $\{s \approx t[\![\varphi_2]\!], \mathbf{u[t\sigma_2\mu] \approx v[\![\varphi_1 \wedge \varphi_2\mu]\!]}\} \cup \Gamma$.[5]

Basic Simplification is based on the notion of reduced-relative-to and is described in [BGLS95].

Basic Simplification

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

– 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 $u[s'] \approx v[\![\varphi_1]\!]$.

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 inference to make the application of Basic Simplification possible. *Basic Blocking* is a deletion rule based on the reduced-relative-to condition that deletes an equation with a reducible constraint.

[4] In rules, we assume the two premises have disjoint sets of variables. If the two equations share some variables, we first rename one premise so that they no longer share any variables, before performing the inference rule. We denote by *"into" equation*, the equation $u[s'] \approx v[\![\varphi_1]\!]$, by *"from" equation*, the equation $s \approx t[\![\varphi_2]\!]$ and by *conclusion equation*, the deduced equation.

[5] This formulation resolves the ambiguity of the first notation. The ambiguity can be resolved by remembering that inference rules add equations, while simplification deletion rules add and delete an equation.

We call *BCPBS* the Basic Completion inference system consisting of these two rules plus Basic Blocking and Retraction. In this paper we give a new Basic Completion inference system **BCPES** which uses the Basic Critical Pair rule and a restricted version of the **Standard Simplification** rule, that we call *E-cycle Simplification*. In the full version of the paper [LS97], *BCPES* is extended by *E-cycle Retraction* and *E-cycle Blocking* rules.

3 E-cycle Simplification

In this section, we describe the framework used in the paper. We first describe the *dependency graph* of a set of unconstrained equations E to complete and then we give the definition of an E-cycle. We describe the way the dependency graph is constructed using the inferences rules of *BCPES* using *Graph Transitions*.

3.1 The dependency graph and E-cycles

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 [Her].

3.2 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 *BCPES* is performed, the dependency graph is updated. A new vertex labelled by the conclusion of the rule is added and edges are added.

We now present the E-cycle Simplification rule and explain how Basic Critical Pair inferences and E-cycle Simplification update the dependency graph using *Graph Transitions*.

The **BCPES** inference system is composed of the Basic Critical Pair Inference rule and of the following E-cycle Simplification deletion rule.

E-cycle Simplification

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

– $u[s'] \approx v[\![\varphi_1]\!]$ can be standard simplified by $s \approx t[\![\varphi_2]\!]$ and

– 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 in the dependency graph.

Definition 3. 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 rule [6] 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:

– 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:
If e_2 is an unconstrained equation then $E_C = \emptyset$, otherwise $E_C = \bigcup_{v \in C\_ancestor(v_0)}(v_2, v, C) \cup \bigcup_{v \in C\_ancestor(v_1)}(v_2, v, C)$.
$C\_ancestor(v_2) = C\_ancestor(v_0) \cup C\_ancestor(v_1)$.
$E_I = \{(v_0, v_2, I)\}$

– A deletion rule.
We have the following Graph Transition $(\{e_0, e_1, \cdots, e_n\} \cup \Gamma, G_i) \rightarrow (\{e_1, e_2, \cdots, e_n\} \cup \Gamma, G_{i+1})$ where e_0 is removed because of $e_1, \cdots, e_n$.
Let e_i be the label of v_i for $i \in \{0, \cdots, n\}$.
- $V_{i+1} = V_i$
- $ED_{i+1} = ED_i \cup E_S$ where:
$E_S = \bigcup_{i \in \{1, \cdots, n\}}(v_0, v_i, S)$.

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. We can notice that E-paths and also E-cycles do not contain an I edge followed by a C edge. This is due to the reducibility constraints associated to I and C edges. 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 7) and E-cycle Simplification forbids creation of E-cycles.

Definition 4. Given a sequence of equations $E_0, E_1, \cdots$, the limit E_∞ is $\bigcup_i \bigcap_{j \geq i} E_j$. Given a sequence of graphs $G_0, G_1, \cdots$ 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 $ED_\infty = \bigcup_i \bigcap_{j \geq i} ED_j$.

[6] A Simplification rule consists of a Critical Pair inference that adds an equation plus a deletion rule.

Definition 5. A *Graph Transition Derivation* from E is a possibly infinite derivation $(E_0 = E, G_0 = G_{init}) \to (E_1, G_1) \to \cdots$, where for all i, $(E_i, G_i) \to (E_{i+1}, G_{i+1})$ 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 6. *An E- cycle does not contain only* C *edges.*

Theorem 7. *Let* $(E_0 = E, G_0 = G_{init}) \to (E_1, G_1) \to \cdots \to (E_{n-1}, G_{n-1}) \to (E_n, G_n) \cdots$ *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.*

To illustrate *BCPES*, 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 8. 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 two inferences processed here is in figure 1. The full development of this example can be found in the full version of the paper [LS97]. The saturated set, we obtain, is $E_\infty = \{a \approx b\ (1), f(g(x)) \approx g(x)\ (2), f(b) \approx b\ (5), f(g(b)) \approx b\ (7), g(x) \approx b[\![\, x =^? a \,]\!]\ (8)\}$.

1. $$\frac{f(g(x)) \approx g(x)\ (2) \qquad f(g(a)) \approx b\ (3)}{g(x) \approx b[\![\, x =^? a \,]\!]\ (4)}$$
   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. $$\frac{f(g(a)) \approx b\ (3) \qquad g(x) \approx b[\![\, x =^? a \,]\!]\ (4)}{f(b) \approx b\ (5)}$$
   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.

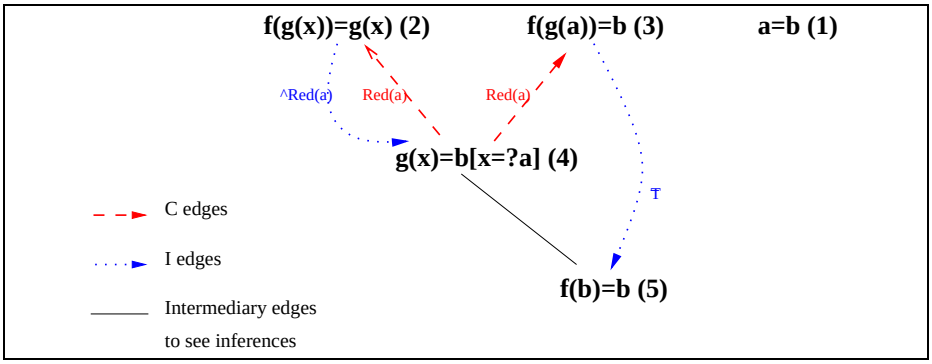

Fig. 1. Dependency graph of Basic Completion with E-cycle Simplification of $E = \{a \approx b, f(g(x)) \approx g(x), f(g(a)) \approx b\}$: the first two inferences.

4 *BCPES* is complete

In this section, we give the completeness result of *BCPES*. 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 ground dependency graph is used to built a model of irreducible equations and a well-founded ordering $\succ_g$ of the equations.

4.1 The ground dependency graphs

In the ground dependency graph, 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.

We first define GG_{init} for a (non-ground) set of equations E to complete. GG_{init} is the initial ground dependency graph (V_{init}, ED_{init}), where V_{init} is the set of vertices such that each $e \in Gr(E)$ 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 [7]. In a ground dependency graph, we no longer speak about E-cycle but only about cycle.

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 $Gr(E_\infty)$. Ground Graph Transitions modulo an equational theory are described in detail in [LS97]. We need to lift

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

from ground level to non-ground level. It is why we speak about Ground Graph Transition Derivation associated to (non-ground) Graph Transition Derivation.

4.2 Completeness proofs

In this section, we first give completeness results concerning the Ground Graph Transition Derivation and then completeness results concerning *BCPES*. 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 graphs contains at least a C and an S edge. The proof is done by contradiction.

Theorem 9. *Let* $(EG_0 = Gr(E), GG_0 = GG_{init}) \rightarrow (EG_1, GG_1) \rightarrow \cdots \rightarrow (EG_n, GG_n) \cdots$ *be a Ground Graph Transition Derivation modulo* E'. *If* GG_n *contains a cycle, then this cycle contains at least a* C *and an* S *edge.*

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

Lemma 10. *Let* $(EG_0 = Gr(E), GG_0) \rightarrow (EG_1, GG_1) \rightarrow \cdots \rightarrow (EG_n, GG_n) \cdots$ *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* $label(v_1)$ *is reducible in* $Gr(E_\infty)$.

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.

Lemma 11. *Let* $(EG_0 = Gr(E), GG_0 = GG_{init}) \rightarrow (EG_1, GG_1) \rightarrow \cdots \rightarrow (EG_n, GG_n) \cdots$ *be a Ground Graph Transition Derivation modulo* $Gr(E_\infty)$. *If* GG_i *is a dependency graph containing a cycle* $\mathcal{C}$, *then* $\mathcal{C}$ *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 11 and theorem 9 are used.

Lemma 12. *Let* $(E_0 = E, G_0 = G_{init}) \rightarrow (E_1, G_1) \rightarrow \cdots \rightarrow (E_n, G_n) \cdots$ *be a Graph Transition Derivation and* $(EG_0 = Gr(E), GG_0 = GG_{init}) \rightarrow (EG_1, GG_1) \rightarrow \cdots \rightarrow (EG_n, GG_n) \cdots$ *its associated Ground Graph Transition Derivation modulo* $Gr(E_\infty)$, *then for all i,*

- *if* GG_i *contains a cycle then* G_i *contains an E-cycle.*
- *if* GG_i *contains an infinite path, then* G_i *contains an E-cycle.*

The first completeness theorem provides a completeness result for Ground Graph Transition derivations. The proof of this theorem is based on the construction of a model of $Gr(E_\infty)$ which is a convergent rewrite system. For doing that the ordering $\succ_g$ is constructed directly from the ground dependency graph GG_∞.

Definition 13. Let $\succ_g$ be the ordering such that $e \succ_g e'$ if and only if there are two vertices 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 12) and so $\succ_g$ is well-founded.

We define redundancy in terms of this ordering.

Definition 14. A ground equation e is *g-redundant* in a set of ground equations E if there are equations $e_1, \cdots, e_n \in E$ such that $e_1, \cdots, e_n \models e$ and $e_i \prec_g e$ for all i.

E-cycle simplification is an example of g-redundancy as expressed in the following lemma.

Lemma 15. *Let E be a set of unconstrained equations. Let $(E_0 = E, G_0 = G_{init}) \rightarrow (E_1, G_1) \rightarrow \cdots \rightarrow (E_n, G_n) \cdots$ be a Graph Transition and $(EG_0 = Gr(E), GG_0 = GG_{init}) \rightarrow (EG_1, GG_1) \rightarrow \cdots \rightarrow (EG_n, GG_n) \cdots$ 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 in some E_i. Then every ground instance e' of e is g-redundant in $Gr(E_\infty)$.*

Theorem 16. *Let E be a set of unconstrained equations. Let $(EG_0 = Gr(E), GG_0 = GG_{init}) \rightarrow (EG_1, GG_1) \rightarrow \cdots \rightarrow (EG_n, GG_n) \cdots$ be a Ground Graph Transition Derivation modulo $Gr(E_\infty)$ where, GG_∞ does not contain a cycle or an infinite path. Then this Ground Graph Transition Derivation is complete in the sense that $Gr(E_\infty)$ is convergent.*

The following main theorem proves the completeness of *BCPES*. The proof is based on the correspondence between the procedural construction of the non-ground dependency graph presented in section 3.2 and the abstract construction of the ground dependency graph presented in section 4. Lemma 12 is mainly used for the proof.

Theorem 17. *Basic Completion with E-cycle Simplification is complete.*

5 Comparison with Basic Simplification

In this section, we compare E-cycle Simplification with Basic Simplification. 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 18. *Let $(E_0 = E, G_0 = G_{init}) \rightarrow (E_1, G_1) \rightarrow \cdots \rightarrow (E_n, G_n) \cdots$ be a Graph Transition Derivation such that $E_0 \rightarrow E_1 \rightarrow \cdots \rightarrow E_n$ is a derivation of* BCPBS. *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.*

Lemma 19. *Let $(E_0 = E, G_0 = G_{init}) \rightarrow (E_1, G_1) \rightarrow \cdots \rightarrow (E_n, G_n) \cdots$ be a Graph Transition Derivation such that $E_0 \rightarrow E_1 \rightarrow \cdots \rightarrow E_n$ is a derivation of* BCPBS. *Then if there is an i such that G_i contains an E-cycle, then this E-cycle does not contain any* I *edge.*

Theorem 20. *Let $(E_0 = E, G_0 = G_{init}) \rightarrow (E_1, G_1) \rightarrow \cdots \rightarrow (E_n, G_n) \cdots$ be a Graph Transition Derivation such that $E_0 \rightarrow E_1 \rightarrow \cdots \rightarrow E_n$ is a derivation of* BCPBS. *Then there is no i such that G_i contains an E-cycle.*

As a direct corollary, we get that if $E_0, E_1, \cdots, E_n$ is a derivation of *BCPBS*, then it is also a derivation of *BCPES*. Inversely, we provide an example that shows that a derivation of *BCPES* is not a derivation of *BCPBS*.

Example 21. Let $E = \{g(x) \approx f(x)\ (1), g(a) \approx b\ (2), h(f(a)) \approx b\ (3)\}$. 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 *BCPES*.

1. $$\frac{g(x) \approx f(x)\ (1) \qquad g(a) \approx b\ (2)}{f(x) \approx b[\![\, x =^{?} a \,]\!]\ (4)}$$
   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. $$\frac{h(f(a)) \approx b\ (3) \qquad f(x) \approx b[\![\, x =^{?} a \,]\!]\ (4)}{h(b) \approx b\ (5)}$$
   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 $h(f(a)) \approx b$.

6 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. We plan to use this method for AC Basic Completion and in particular, for simplification in AC Basic Completion.

We have implemented our method of Basic Completion with E-cycle Simplification. The system is called *ECC* (E-cycle Completion). It is written in *ELAN* [KKV95], which is a language based on rewriting and adapted for prototyping. Some implementation details are available at http://www.loria. fr/∼scharff. The implementation can perform both E-cycle Simplification and Basic Simplification. The system is fully operational, but we have not yet had time to perform interesting experiments. For the conference, we plan to run some examples with the two different methods of simplification and give expermental results comparing the methods.

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.

[BGLS95] L. Bachmair, H. Ganzinger, 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.

[Her] M. Hermann. Constrained reachability is np-complete. http://www.loria.fr/ hermann/publications.html#notes.

[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.

[KKR90] Claude Kirchner, Hélène Kirchner, and M. Rusinowitch. Deduction with symbolic constraints. *Revue d'Intelligence Artificielle*, 4(3):9–52, 1990. Special issue on Automatic Deduction.

[KKV95] Claude Kirchner, Hélène Kirchner, and Marian Vittek. ELAN *V 1.17 User Manual*. Inria Lorraine & Crin, Nancy (France), first edition, November 1995.

[LS95] Christopher Lynch and Wayne Snyder. Redundancy criteria for constrained completion. In *Theoretical Computer Science*, volume 142, pages 141–177, 1995.

[LS97] Christopher Lynch and Christelle Scharff. Basic completion with e-cycle simplification, 1997. http://www.loria.fr/∼scharff.

[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.