On Completion Theorem Proving

Maria Paola Bonacina
Department of Computer Science
SUNY at Stony Brook
Stony Brook, NY 11794-4400, USA
{bonacina}@sbcs.sunysb.edu

December 17, 1990

Abstract

This report is a study of Knuth-Bendix type completion procedures for theorem proving, generation of decision procedures, inductive theorem proving and logic programming. We provide a new abstract framework for completion procedures based on a notion of *proof reduction*. This extends the proof orderings approach of [8, 11], which was originally conceived for completion procedures as generators of decision procedures.

The interpretation of completion as semidecision procedure for theorem proving has been regarded since its introduction [52] as a side effect of the interpretation as generator of confluent systems. A procedure which is guaranteed to eventually generate a confluent system is not efficient as a theorem prover, because confluence of the limit requires that all the critical pairs are considered during the derivation.

We address this problem by decoupling the interpretation as semidecision procedure from the interpretation as generator of confluent systems. We define a new concept of *fairness*, which does not require that all the critical pairs are eventually considered and therefore allows efficient search plans for theorem proving. We show that if the inference rules are refutationally complete and the search plan is fair according to our definition, a completion procedure is a semidecision procedure. This proves the classical result in [52], about completion as a semidecision procedure, from weaker, strictly theorem proving oriented hypotheses.

If the search plan is *uniformly fair*, i.e. such that all the critical pairs are eventually considered, the limit of a derivation by completion is a confluent system, or, more generally, a *saturated* presentation of the given theory. Under additional conditions which depends on the logic, a saturated set is a *decision procedure* and completion is a generator of decision procedures. If uniform fairness is assumed, the semidecision process and the saturation process can be related as it was done in [52] for the equational case. We obtain a completely general version of this theorem, which we call *the general Knuth-Bendix-Huet theorem*.

We apply our abstract framework to give a new presentation of the *Unfailing Knuth-Bendix procedure* [48, 14], the *AC-UKB procedure* [70, 55, 9, 1] with *Cancellation laws* [49], the *S-strategy* [48] and especially the *Inequality Ordered Saturation strategy* [3], which had not been presented as a set of inference rules before.

For the application of completion to disprove inductive conjectures, we show that the so called *inductionless induction* method is a semidecision process. For logic programming, we present an operational and denotational semantics of *rewrite programs*. The operational semantics is given by a completion procedure termed *Linear Completion*.

We conclude indicating some directions for future work, which include a notion of *completion modulo a decision algorithm*.

Contents

Chapter 1

Introduction

The Knuth-Bendix completion procedure [62] computes a possibly infinite confluent rewrite system equivalent to a given set of equations [51]. If a set of equations E and an equation $s \simeq t$ are given, it semidecides whether $s \simeq t$ is a theorem of E, as first remarked in [65, 52]. These results hold if the procedure does not fail on an unoriented equation. Unoriented equations can be handled by adopting the Unfailing Knuth-Bendix method [48, 14], which generates a ground confluent set of equations.

Many completion procedures, related to Knuth-Bendix to a different extent, have been designed. They include procedures for equational theories with special sets of axioms [70, 55, 9], Horn logic with equality [63, 36], first order logic [44, 45, 59, 10], first order logic with equality [46, 47, 49, 50, 74, 78, 15, 16], inductive theorem proving in equational and Horn theories [53, 38, 63] and logic programming [29, 30, 31, 20]. Surveys have been given in [33, 34].

In this report a new abstract framework for the study of Knuth-Bendix type completion procedures is presented and applied to theorem proving, generation of decision procedures, inductive theorem proving and logic programming.

A *completion procedure* is composed of *inference rules* and a *search plan*. The inference rules determine what can be derived from given data. The search plan chooses at each step of the derivation which inference rule to apply to which data and therefore it determines the unique derivation that the procedure computes from a given input.

A completion process can be regarded at a very general level as *problem transformation*. Although the problem $E \models \forall \bar{x} s \simeq t$ is semidecidable in general. it is decidable by reduction if E is ground confluent. The transformation of a set of equations E into a ground confluent set of equations E', which presents the same theory, can then be regarded as transformation of a semidecidable problem into a decidable problem. The semidecidable problem is proving validity by using E and the decidable problem is proving validity by using E'. In theorem proving, a specific semidecidable problem $E \models \forall \bar{x} s \simeq t$, written as $(E; \hat{s} \simeq \hat{t})$, is transformed until it has the form $(E_k; \hat{s}_k \simeq \hat{s}_k)$, where $E_k \models \forall \bar{x} s_k \simeq s_k$ is trivially decidable. In inductive theorem proving, the semidecidable problem is that $\forall \bar{x} s \simeq t$ is *not* an inductive theorem of E: $E \cup \{s \simeq t\}$ is transformed until the problem can be decided by a given oracle.

We study such transformations at the *proof* level. Since the work done in [8, 11], standard

Knuth-Bendix completion of a set of equations into a confluent system is regarded as reduction of the equational proofs $s \leftrightarrow^*_E t$ in the original presentation into rewrite proofs $s \rightarrow^*_{E_\infty} \circ \leftarrow^*_{E_\infty} t$ in the limit E_∞ of the derivation. We continue and extend this approach to the other applications of completion, including theorem proving and inductive theorem proving. This requires more general notions of *proof reduction* and *proof ordering* than those in [8, 11], since the original notions do not account for *backward reasoning*, which is necessary in theorem proving. Each inference step either reduces some proofs or reduces the data set itself by eliminating *redundant* sentences. Proof orderings are used to define all the fundamental concepts throughout this work.

We consider first theorem proving. The interpretation of completion as semidecision procedure for theorem proving is the most important one for us. It appeared first in the landmark paper where Huet proved the correctness of Kunth-Bendix completion [52]. Huet proved that if the search plan is fair, the limit of an unfailing Knuth-Bendix derivation is a confluent rewrite system and, as a consequence, if a theorem $s \simeq t$ is given to the procedure, it semidecides the validity of $\forall \bar{x} s \simeq t$. We refer to this theorem as the *Knuth-Bendix-Huet theorem.*

In spite of this early result, the interpretation of completion as semidecision procedure has been neglected, being obscured by the interpretation as generator of confluent systems. The interpretation of Knuth-Bendix completion as generator of confluent systems is by far the most well known one, whereas theorem proving is basically regarded as a side effect of the generation of a confluent system. This view of completion is not acceptable from the theorem proving perspective, because a procedure which is guaranteed to eventually generate a confluent system cannot be efficient as theorem prover. We reverse the traditional way of presenting completion procedures: we present them as semidecision procedures with the generation of confluent systems as a special side effect.

We decouple the interpretation of completion as semidecision procedure from the interpretation of completion as generator of confluent systems. We prove that if the inference rules are *refutationally complete* and the search plan is *fair*, a completion procedure is a *semidecision procedure*. *Refutational completeness* means that for all unsatisfiable inputs, there exist successful derivations by the inference rules of the strategy. *Fairness* means that whenever successful derivations exist, the search plan guarantees that the computed derivation is successful, that is all the inference steps which are necessary to prove the goal are eventually done. In particular, all the critical pairs which are necessary to prove the goal are eventually considered. We give a new definition of fairness to capture this concept.

This notion of fairness is the key difference between completion for theorem proving and completion for the generation of confluent systems. In Huet's paper [52] and in all the following work on completion [8, 11, 74, 16], fairness of a derivation consists in eventually considering all critical pairs. We call this property *uniform fairness* in order to distinguish it from fairness for theorem proving. Uniform fairness is necessary for the limit of a derivation to be confluent, but it is not necessary for theorem proving, because not all the critical pairs are necessary to prove a given theorem. Considering all critical pairs is an unnecessary source of inefficiency in a theorem proving derivation. All the definitions of fairness of completion procedures appeared so far in the literature [52, 11, 74, 16] require uniform fairness, because they do not separate theorem proving from the generation of a confluent system.

Next, we consider the most classical interpretation of completion, that is as generator of confluent systems, or, more generally, as generator of decision procedures. If the search plan is uniformly fair, the limit of a derivation by completion is a *saturated* presentation. The concept of saturated set is a generalization of confluent system: a presentation is saturated if no non-trivial consequences can be added [63, 16]. If a presentation is saturated, the derivations from that presentation are *linear input* derivations [24]. If linear input derivations from a saturated set are guaranteed to be well founded, a saturated set is a *decision procedure* and the completion procedure is a *generator of decision procedures*. Conditions for the well foundedness of a linear input derivation are known for equational logic and Horn logic with equality [63, 16]. Our notions of uniform fairness, saturated set, well founded linear input derivation, decision procedure and the theorems relating them give an abstract presentation of the application of completion to the generation of decision procedures, which covers and unify all the known results about this interpretation of completion in equational logic and Horn logic with equality.

Still, the separate study of completion as semidecision procedure and completion as generator fails to match entirely the meaning of the Knuth-Bendix-Huet theorem we started from. In that theorem the two interpretations of completion are related: if the limit of a derivation is saturated, any true equation can be proved during the derivation, regardless of whether the limit is finite or not. Therefore, we also give a new, general Knuth-Bendix-Huet theorem to relate the semidecision process and the saturation process in our framework.

Our proof reduction framework has allowed us to achieve two main results. First, we have proved Huet's classical result about completion as a semidecision procedure from weaker, strictly theorem proving oriented hypotheses, which do not imply any confluence property of the limit of the derivation. This proves that the interpretation of completion as semidecision procedure can be decoupled from its interpretation as generator of saturated sets and therefore it is not just a side effect of the latter. Second, we have given the first generalization of the Knuth-Bendix-Huet theorem beyond equational logic. Our theorem does not depend on a specific logic. The only notion which depends on the logic are the conditions for a saturated set to be a decision procedure.

The report is organized as follows. The second chapter contains the basic elements of our framework: *proof orderings*, *proof reduction*, *redundancy* and *completion procedure*. In the third chapter we concentrate on theorem proving. We define *fairness* and we prove that a completion procedure with refutationally complete inference rules and fair search plan is a semidecision procedure. In the fourth chapter, we focuse instead on *uniform fairness*, *saturated* presentations and the *generation of decision procedures*. In the fifth chapter we give our general version of the Knuth-Bendix-Huet theorem. In the sixth chapter we present some completion procedures for equational logic: we show that the basic *Unfailing Knuth-Bendix procedure* [48, 14] and some of its extensions, such as the *AC-UKB procedure* [70, 55, 9, 1] with *Cancellation laws* [49], the *S-strategy* [48] and the *Inequality Ordered Saturation strategy* [3] fit nicely in our framework. To our knowledge, this is the first presentation of these extensions of the UKB procedure as sets of inference rules. The seventh chapter is devoted to the application of completion to disprove inductive conjectures, the so called *inductionless induction* method [53]. Completion for inductionless induction is a *semidecision procedure for disproving inductive conjectures*. In the eighth chapter, we cover the application of completion to logic programming, by giving an operational

and denotational semantics for rewrite programs interpreted by *Linear Completion*. In the last chapter we summarize our results and we indicate some possible directions for future work. These include the extension of our view of completion as problem transformation to *completion modulo a decision algorithm*. If a decision algorithm is known for a certain theory, completion of a theorem proving problem $(S;\varphi)$ does not need to proceed up to a stage $(S_k;\varphi_k)$ where $S_k \models \varphi_k$ is trivial, such as when φ_k is an equation $s \simeq s$. The process halts at stage k if $S_k \models \varphi_k$ is decidable by the given algorithm. This approach opens the way to the study of completion in connection with decision procedures for special theories.

Chapter 2

Completion procedures

Our entire approach to completion procedures is coherently based on a notion of *proof reduction*. In theorem proving, one wants to reduce one single proof, the proof of the target theorem: the derivation halts successfully if the target has been reduced to some trivially true theorem, such as $s \simeq s$, whose proof is *empty*. In traditional completion, one wants to reduce all proofs: for instance, in Knuth-Bendix completion all equational proofs have to be reduced to rewrite proofs.

In order to formalize all concepts in terms of proof reduction, we need a notion of well founded *proof ordering*. Our starting point is the proof orderings approach originally given in [8, 11]. However, proof orderings as in [8] do not apply to a theorem proving derivation, because they allow to compare only two proofs of the same theorem. In theorem proving, the target is modified by inference steps applying to the theorem itself. Therefore, we give a new notion of proof ordering, where proofs of different theorems can be compared.

Then we characterize *inference rules* and *search plan* of a *completion procedure*. At any stage of a derivation the current data set is the portion of the search space which has been actually generated. The size of the search space grows exponentially with the size of the input. An ideal theorem proving strategy would find a proof of the target by consuming the least amount of space and time, that is by generating the smallest portion of the search space. The closer to this ideal a strategy is, on a reasonably large class of problems, the more efficient it is. Both the inference rules and the search plan affect the efficiency.

A search plan consists of one or more criteria to decide which data and inference rule to select for the next step. These criteria are heuristical in nature, since in general it is impossible to determine a priori whether a given step leads us closer to the proof or not. We shall see that some simple criteria can be described using orderings.

Inference rules are either *expansion* rules or *contraction* rules. Expansion inference rules generate new sentences. A theorem proving strategy including expansion rules only is very inefficient, because at each step the number of generated clauses grows. The available memory space can be quickly exhausted without reaching a proof of the target even on fairly easy problems. A contraction rule deletes a sentence or replaces it by smaller ones: by deleting a sentence it prevents the strategy from generating the whole search space of all the sentences which could be derived from that sentence. Contraction rules reduce the growth of the search space. The balance of

expansion and contraction rules influences then significantly the efficiency of the strategy.

Inference rules of a completion procedure have the property that each inference step either reduces some proofs or deletes a *redundant* sentence. Both proof reduction and redundancy are defined in terms of the assumed proof ordering.

2.1 Basic definitions

In this section we recall some basic concepts and notations for theorem proving. Our presentation is consistent with [34, 35].

Given a finite set F of constant symbols and function symbols with their arities and a denumerable set X of variable symbols, $T(F, X)$ is the set of *terms* on F and X. A term is *ground* if it does not contain variables. The set of ground terms is denoted by $T(F)$.

A term s is a *subterm* of a term t if s occurs in t. We write t as $c[s]$ to indicate that s is a subterm of t in the *context* c.

We assume the standard representation of terms as finite ordered trees: a variable x or a constant c is represented as a tree with only a single node labeled by x or c respectively. A term $f(t_1 \dots t_n)$ is represented by a tree whose root has label f and has n ordered outgoing edges labeled $1 \dots n$ pointing to the roots of the trees for the terms $t_1 \dots t_n$.

We write $s = t|u$ to specify that s is the subterm of t at *position* u, where u is the string of natural numbers labeling the edges in the path from the root of the tree for t to the root of the subtree for s. More precisely, the set $\mathcal{O}(t)$ of positions in a term $t = f(t_1 \dots t_n)$ is the set of strings of natural numbers such that $\lambda \in \mathcal{O}(t)$, where λ is the empty string, and if $u \in \mathcal{O}(t_i)$ for some i, $1 \leq i \leq n$, then $i \cdot u \in \mathcal{O}(t)$. The empty string λ denotes the root position, i.e. $t|\lambda = t$, and the string $i \cdot u$ denotes the position u in the i-th subterm of t, i.e. $f(t_1 \dots t_n)|i \cdot u = t_i|u$. Two distinct positions u and v in $\mathcal{O}(t)$ are *disjoint* if neither u is a prefix of v nor v is a prefix of u. The notation $s[t]_u$ represents the term obtained by replacing $s|u$ by t.

A *substitution* σ is a set $\{x_1 \mapsto s_1 \dots x_n \mapsto s_n\}$ such that

- $\forall i, j$, $i \neq j$ implies $x_i \neq x_j$,
- $\forall i, j$, $x_i \notin V(s_j)$, where $V(s_j)$ is the set of variables occurring in the term s_j.

The set $\{x_1 \dots x_n\}$ is called the *domain* of the substitution σ: $Dom(\sigma) = \{x_1 \dots x_n\}$. The set of all the variables occurring in the terms $s_1 \dots s_n$ is called the *range* of σ: $Ran(\sigma) = \bigcup_{j=1}^{n} V(s_j)$. A substitution σ is *ground* if $Ran(\sigma) = \emptyset$. A substitution σ applies to a term as follows:

- $x\sigma = s$ if $x \mapsto s \in \sigma$,
- $x\sigma = x$ if $x \notin Dom(\sigma)$,
- $c\sigma = c$ if c is a constant,
- $f(t_1 \dots t_n)\sigma = f(t_1\sigma \dots t_n\sigma)$, otherwise.

Given two substitutions $\sigma = \{x_1 \mapsto s_1 \dots x_n \mapsto s_n\}$ and $\rho = \{y_1 \mapsto r_1 \dots y_m \mapsto r_m\}$ such that $Dom(\sigma) \cap Ran(\rho) = \emptyset$, their *composition* is the substitution $\sigma\rho = \{x_1 \mapsto s_1\rho \dots x_n \mapsto s_n\rho\} \cup \{y_j \mapsto r_j | y_j \mapsto r_j \in \rho, y_j \notin Dom(\sigma)\}$. Composition of substitutions is associative, i.e. $(\sigma\rho)\theta = \sigma(\rho\theta)$. The second condition in our definition of substitution implies that $Dom(\sigma) \cap Ran(\sigma) = \emptyset$. This property is equivalent to $\sigma\sigma = \sigma$, i.e. *idempotence* of composition.

An *ordering* $\succ$ is a binary relation which is transitive and irreflexive. Transitivity and irreflexivity imply asymmetry. An ordering is *partial* in general. It is *total* if for every two distinct elements s and t in the ordered set, either $s \succ t$ or $t \succ s$. An ordering is *well founded* if there is no infinite chain $s_1 \succ s_2 \succ \dots s_n \succ \dots$.

The subterm relation defines an ordering, called the *subterm ordering*: $t \trianglerighteq s$ if $t = c[s]$ and $t \triangleright s$ if c is not empty. A term t is an *instance* of a term s if there is a substitution σ such that $t = s\sigma$: we write $t \unrhd s$ and the relation $\unrhd$ is called *subsumption ordering*. The substitution σ is called a *matching substitution*. If t is an instance of s and s is an instance of t, the two terms are equal up to a permutation of variables. A substitution which is just a permutation of variables is said to be a *renaming* and the two terms s and t are said to be *variant* of each other, denoted by $s \stackrel{\bullet}{=} t$. Otherwise, t is a *proper instance* of s: we write $t \rhd s$ and we call the relation $\rhd$ the *proper subsumption ordering*.

The *encompassment ordering* $\trianglerighteq$ is the composition of the subterm ordering and the subsumption ordering: $t \trianglerighteq s$ if $t|u = s\sigma$ for some position u and substitution σ; $t \triangleright s$ if $t \trianglerighteq s$ and $s \not\stackrel{\bullet}{=} t$.

Similarly, a substitution σ is an instance of a substitution θ if there is a third substitution ρ such that $\forall x \in Dom(\sigma), x\sigma = x\theta\rho$. The substitution σ is a proper instance of θ if ρ is not a renaming. Similar to terms, we write $\sigma \unrhd \theta$ and $\sigma \rhd \theta$.

Given two terms s and t, a substitution σ is a *unifier* of s and t if $s\sigma = t\sigma$. A substitution σ is a *most general unifier* (mgu) of s and t if σ is a unifier of s and t and for all unifiers ρ of s and t, $\rho \unrhd \sigma$.

We define the following properties for an ordering $\succ$ on terms:

- *monotonicity:* $s \succ t$ implies $c[s] \succ c[t]$ for all contexts c,
- *stability*: $s \succ t$ implies $s\sigma \succ t\sigma$ for all substitutions σ and
- *subterm property*: $c[s] \succ s$ for all terms s and contexts c.

The first property says that if $s \succ t$, then any superterm $c[s]$ of s is greater than the corresponding superterm $c[t]$ of t. The second property says that substitutions preserve the order relation on terms. The third one says that a term is greater than any of its subterms, i.e. an ordering with such property includes the subterm ordering. A monotonic, stable and well founded ordering is a *reduction ordering*. A monotonic and stable ordering with the subterm property is a *simplification ordering*. Monotonicity, stability and subterm property imply well foundedness [27]. Therefore a simplification ordering is also a reduction ordering.

An *equation* is an unordered pair of terms $l \simeq r$. A *rewrite rule* is an ordered pair of terms $l \to r$. A set of rewrite rules is called a *term rewriting system* or *rewrite system*. An equation $l \simeq r$ is oriented into a rewrite rule $l \to r$, if $l \succ r$ for a reduction ordering $\succ$.

A rewrite system R defines a relation $\rightarrow_R$ on terms as follows: $s \rightarrow_R t$ if there are a rewrite rule $l \rightarrow r \in R$, a substitution σ and a position u such that $s|u = l\sigma$ and t is $s[r\sigma]_u$. The relation $\leftrightarrow_R$ is defined as the union $\rightarrow_R \cup \leftarrow_R$. For a set of equations E, $s \leftrightarrow_E t$ if there are an equation $l \simeq r \in E$, a substitution σ and a position u such that $s|u = l\sigma$ and t is $s[r\sigma]_u$; $s \rightarrow_E t$ if $s \leftrightarrow_E t$ and $s \succ t$ for a reduction ordering $\succ$. We denote by $\leftrightarrow_E^*$ the transitive and reflexive closure of $\leftrightarrow_E$. The relation $\leftrightarrow_E^*$ is a congruence, the congruence defined by E on the set of terms. The equality $\leftrightarrow_E = \rightarrow_E \cup \leftarrow_E$ does not hold in general: it holds if and only if the ordering $\succ$ is total in every congruence class defined by E.

The following definitions apply to both a rewrite system R and a set of equations E. The only difference is the way the relations $\rightarrow_R$ and $\rightarrow_E$ are defined, as shown above. We denote by $\leftrightarrow_E^*$, $\rightarrow_E^*$ and $\leftarrow_E^*$ the transitive and reflexive closure of $\leftrightarrow_E$, $\rightarrow_E$ and $\leftarrow_E$ respectively. A term s is in *normal form* with respect to E, or equivalently E*-irreducible*, if there is no term t such that $s \rightarrow_E t$. A set of equations E is *Church-Rosser* if for all terms s and t, $s \leftrightarrow_E^* t$ implies $s \rightarrow_E^* \circ \leftarrow_E^* t$. It is *confluent* if for all terms s and t, $s \leftarrow_E^* \circ \rightarrow_E^* t$ implies $s \rightarrow_E^* \circ \leftarrow_E^* t$. The Church-Rosser property and the confluence property are equivalent if $\leftrightarrow_E = \rightarrow_E \cup \leftarrow_E$ holds. A set of equations E is *locally confluent* if for all terms s and t, $s \leftarrow_E \circ \rightarrow_E t$ implies $s \rightarrow_E^* \circ \leftarrow_E^* t$. It is *canonical* if it is both confluent and *reduced*, that is for all $l \simeq r \in E$, l and r are in normal form with respect to $E - \{l \simeq r\}$.

Let P be a finite set of predicate symbols with their arities. We denote by $A(P, F, X)$ and $A(P, F)$ the sets of *atoms* and *ground atoms* on $< P, F, X >$.

If P includes the equality predicate $\simeq$, an equation is an atom in $A(P, F, X)$. A *literal* is an atom or a negated atom. A *clause* is a disjunction of literals. A *unit clause* is a clause made of one single literal. A *Horn clause* is a clause which contains at most one positive literal, called the *head*, while the negative literals form the *body* of the clause. All variables occurring in a clause are implicitly universally quantified. We denote by $\hat{s} \simeq \hat{t}$ an equation which contains only universally quantified variables and therefore can be regarded as a ground equation.

The definitions given for terms and substitutions extend to atoms, literals and clauses. In particular, the (proper) subsumption ordering extends naturally to atoms and clauses and the encompassment ordering extends to equations as follows: $(p \simeq q) \trianglerighteq (l \simeq r)$ means that $p = c[l\sigma]$ and $q = c[r\sigma]$ for some context c and substitution σ; $(p \simeq q) \triangleright (l \simeq r)$ hold if either c is not empty or σ is not a renaming.

The two most common simplification orderings are the *recursive path ordering* [27] and the *lexicographic path ordering* [58].

Given n partially ordered sets $(A_1, \succ_1) \ldots (A_n, \succ_n)$ the *lexicographic extension* $\succ_{lex}$ of the orderings $\succ_1 \ldots \succ_n$ is the ordering on $A_1 \times \ldots \times A_n$ defined as follows: $(a_1 \ldots a_n) \succ_{lex} (b_1 \ldots b_n)$ if and only if there exists an i, $1 \leq i \leq n$, such that $a_j = b_j$, $\forall j < i$ and $a_i \succ_i b_i$. If the orderings $\succ_1 \ldots \succ_n$ are well founded, their lexicographic extension is well founded as well.

Given a partially ordered set $(A, \succ)$ the *multiset extension* $\succ_{mul}$ of the ordering $\succ$ is the ordering on the set of multisets on A, $M(A)$, defined as follows:

- $\{a\} \cup M \succ_{mul} \emptyset$, where $\emptyset$ is the empty multiset.

- $\{a\} \cup M \succ_{mul} \{a\} \cup N$ if $M \succ_{mul} N$.
- $\{a\} \cup M \succ_{mul} \{b\} \cup N$ if $a \succ b$ and $\{a\} \cup M \succ_{mul} N$.

The multiset extension of a well founded ordering is well founded [26].

We assume that $>$ is a partial ordering, called *precedence*, on F. A term $s = f(s_1 \ldots s_n)$ is greater than a term $t = g(t_1 \ldots t_m)$ in the *recursive path ordering*, i.e. $s = f(s_1 \ldots s_n) \succ^{rpo} g(t_1 \ldots t_m) = t$, if and only if one of the following conditions holds:

- $s_i \succeq^{rpo} t$ for some i, $1 \leq i \leq n$.
- $f > g$ and $s \succ^{rpo} t_j$, $\forall j$, $1 \leq j \leq m$.
- $f = g$ and $\{s_1 \ldots s_n\} \succ^{rpo}{}_{mul} \{t_1 \ldots t_m\}$ where $\succ^{rpo}{}_{mul}$ is the multiset extension of $\succ^{rpo}$.

A term $s = f(s_1 \ldots s_n)$ is greater than a term $t = g(t_1 \ldots t_m)$ in the *lexicographic path ordering*, i.e. $s = f(s_1 \ldots s_n) \succ^{lpo} g(t_1 \ldots t_m) = t$, if and only if one of the following conditions holds:

- $s_i \succeq^{lpo} t$ for some i, $1 \leq i \leq n$.
- $f > g$ and $s \succ^{lpo} t_j$, $\forall j$, $1 \leq j \leq m$.
- $f = g$, $(s_1 \ldots s_n) \succ^{lpo}{}_{lex} (t_1 \ldots t_m)$ where $\succ^{lpo}{}_{lex}$ is the lexicographic extension of $\succ^{lpo}$ and $\forall j \geq 2$, $s \succ^{lpo} t_j$.

These two orderings are easy to implement, because their definitions are purely *syntactic*. A third popular ordering is the Knuth-Bendix ordering [62]. Another class of orderings is that of *semantic orderings*, which are based on interpreting the signature of the terms on some partially ordered domain. A survey on orderings and their properties is given in [32].

Simplification orderings on terms can be extended in a coherent way to literals and clauses. For instance, given a precedence $>$ on the set $F \cup P$ of function and predicate symbols, a simplification ordering $\succ$ on terms and literals can be defined as follows: $L = A(s_1 \ldots s_n) \succ B(t_1 \ldots t_m) = M$, if and only if one of the following conditions holds:

- $s_i \succeq M$ for some i, $1 \leq i \leq n$.
- $A > B$ and $L \succ t_j$, $\forall j$, $1 \leq j \leq m$.
- $A = B =\simeq$ and $\{s_1, s_2\} \succ_{mul} \{t_1, t_2\}$ where $\succ_{mul}$ is the multiset extension of $\succ$.
- $A = B \neq\simeq$ and $(s_1 \ldots s_n) \succ_{lex} (t_1 \ldots t_m)$ where $\succ_{lex}$ is the lexicographic extension of $\succ$.

A *complete simplification ordering* is a simplification ordering which is total on the set $T(F) \cup A(P, F)$ of ground terms and literals. Once we have a (complete) simplification ordering on literals, we can extend it to a (complete) simplification ordering on clauses by applying the multiset extension to the ordering on literals.

2.2 Proof orderings for theorem proving

A finite set of sentences S is a *presentation* of the *theory* $Th(S) = \{\varphi | S \models \varphi\}$. A *theorem proving problem* is to decide whether $\varphi \in Th(S)$ for a presentation S of a theory and a *target* φ. A *theorem proving derivation* is a sequence of deductions

$$(S_0; \varphi_0) \vdash (S_1; \varphi_1) \vdash \ldots \vdash (S_i; \varphi_i) \vdash \ldots,$$

where at each step the problem of deciding $\varphi_i \in Th(S_i)$ reduces to the problem of deciding $\varphi_{i+1} \in Th(S_{i+1})$. A step $(S_i; \varphi_i) \vdash (S_{i+1}; \varphi_i)$, where the presentation is modified, is a *forward reasoning* step. A step $(S_i; \varphi_i) \vdash (S_i; \varphi_{i+1})$, where the target is modified, is a *backward reasoning* step, which derives a new goal from the current one. Informally, the derivation halts successfully at stage k if $\varphi_k \in Th(S_k)$ is trivially true and therefore it can be asserted that $\varphi_0 \in Th(S_0)$.

In this section we introduce a notion of *proof ordering*, which allows us to describe a theorem proving derivation as a *proof reduction* process. We denote proofs by capital Greek letters: $\Upsilon(S, \varphi)$ denotes a proof of φ from axioms in S. Proofs are often represented as trees whose nodes are labeled by sentences: the tree associated to $\Upsilon(S, \varphi)$ has φ as label of the root, elements in S as labels of the leaves and a node ψ has children $\psi_1 \ldots \psi_n$ if ψ is derived from $\psi_1 \ldots \psi_n$ by a step in $\Upsilon(S, \varphi)$. In equational logic, such a proof can also be represented as a chain [8]

$$s_1 \leftrightarrow_{l_1 \simeq r_1} s_2 \leftrightarrow_{l_2 \simeq r_2} \ldots \leftrightarrow_{l_{n-1} \simeq r_{n-1}} s_n,$$

where $s_1 \leftrightarrow_{l_1 \simeq r_1} s_2$ means that the equality of s_1 and s_2 is established by the equation $l_1 \simeq r_1$ because s_1 and s_2 are $c[l_1\sigma]$ and $c[r_1\sigma]$ for some context c and substitution σ. We write $s \rightarrow_{l \simeq r} t$ if it is known that $s \succ t$ holds for the assumed ordering $\succ$ on terms. A proof in the form $s \leftarrow \circ \rightarrow t$ is called a *peak*. A proof in the form $s \rightarrow^* \circ \leftarrow^* t$ is called a *rewrite proof*.

An ordering on proofs is defined in general starting from some ordering on the data involved in the proofs. Our first basic assumption is to have a complete simplification ordering $\succ$. We prefer to have a simplification ordering, even if a well founded, monotonic and stable ordering total on ground objects is sufficient. A *proof ordering* is a monotonic, stable and well founded ordering on proofs [8]. As an example we give the following proof ordering from [36]:

Example 2.2.1 *A proof ordering to compare two ground equational proofs* $\Upsilon(E, s \simeq t) = s \leftrightarrow_E^* t$ *and* $\Upsilon'(E', s \simeq t) = s \leftrightarrow_{E'}^* t$*, is defined as follows. We associate to a ground equational step* $s \leftrightarrow_{l \simeq r} t$ *the triple* (s, l, t)*, if* $s \succ t$*. We compare these triples by the lexicographic extension* $>^e$ *of the complete simplification ordering* $\succ$*, the strict encompassment ordering* $\triangleright$ *and again the ordering* $\succ$*. Then we compare two proofs* $\Upsilon(E, s \simeq t)$ *and* $\Upsilon'(E', s \simeq t)$ *by the multiset extension* $>^e{}_{mul}$ *of* $>^e$*.*

Proof orderings as defined in [8] allow us to compare only two proofs $\Upsilon(S, \varphi)$ and $\Upsilon'(S', \varphi)$ of the same theorem φ in different presentations S and S' of a theory. This notion of proof ordering is not suitable for theorem proving, because in a theorem proving derivation

$$(S_0; \varphi_0) \vdash (S_1; \varphi_1) \vdash \ldots \vdash (S_i; \varphi_i) \vdash \ldots$$

both the presentation and the target are transformed. In order to compare the proof of φ_i in S_i and the proof of φ_{i+1} in S_{i+1}, we need a proof ordering such that two proofs $\Upsilon(S, \varphi)$ and $\Upsilon'(S', \varphi')$ of different theorems may be comparable. Proof orderings with this property do exist and can actually be obtained quite easily. For instance the proof ordering of the previous example can be transformed into a proof ordering for proofs of different theorems as follows:

Example 2.2.2 *We can compare any two ground equational proofs* $\Upsilon(E, s \simeq t) = s \leftrightarrow_E^* t$ *and* $\Upsilon'(E', s' \simeq t') = s' \leftrightarrow_{E'}^* t'$ *by comparing the pairs* $(\{s, t\}, s \leftrightarrow_E^* t)$ *and* $(\{s', t'\}, s' \leftrightarrow_{E'}^* t')$ *by the lexicographic combination* $>_u$ *of the multiset extension* $\succ_{mul}$ *of the ordering* $\succ$ *on terms and the multiset extension* $>^e{}_{mul}$ *of* $>^e$.

The minimum proof is the *empty proof*. We denote by *true* the theorem whose proof is empty and we assume that *true* is the bottom element in our ordering on terms and literals. Given a pair $(S; \varphi)$, we can select a minimal proof among all proofs of φ from S:

Definition 2.2.1 *Given a proof ordering* $>_p$*, we denote by* $\Pi(S, \varphi)$ a minimal proof *of* φ *from* S *with respect to* $>_p$*, i.e. a proof such that for all proofs* $\Upsilon(S, \varphi)$ *of* φ *from* S*,* $\Upsilon(S, \varphi) \not<_p \Pi(S, \varphi)$.

By assuming a proof ordering $>_p$, we can regard a theorem proving derivation

$$(S_0; \varphi_0) \vdash (S_1; \varphi_1) \vdash \ldots \vdash (S_i; \varphi_i) \vdash \ldots,$$

as a process of reducing $\Pi(S_0, \varphi_0)$ to the empty proof and φ_0 to *true*. At each step $\Pi(S_i, \varphi_i)$ is replaced by $\Pi(S_{i+1}, \varphi_{i+1})$ and the derivation halts successfully at stage k if $\Pi(S_k, \varphi_k)$ is empty and φ_k is *true*.

Our generalization of the classical notion of proof orderings is more significant than it may seem at first glance. Proof orderings were introduced in [8] to prove correctness of the Knuth-Bendix completion procedure as a procedure which generates possibly infinite, confluent term rewriting systems. A derivation by Knuth-Bendix completion in that context is a process

$$S_0 \vdash S_1 \vdash \ldots \vdash S_i \vdash \ldots.$$

involving a presentation but no target. In other words, it is a purely forward derivation. The purpose of such a derivation is to transform a given presentation equational presentation until it is confluent. Therefore it is sufficient to be able to compare $\Pi(S_i, \varphi)$ and $\Pi(S_{i+1}, \varphi)$ for any theorem φ in the theory.

This is not the case in theorem proving, since the purpose of a derivation is to prove a specific theorem. Theorem proving requires *backward reasoning*, since a theorem proving problem includes a target. Furthermore, backward reasoning is necessary to obtain a *target-oriented* and therefore presumably efficient procedure. The classical proof orderings approach does not apply to theorem proving because it does not provide for backward reasoning. On the other hand, our proof orderings approach applies to both theorem proving and traditional completion.

2.3 Inference rules and search plans

Since completion procedures are theorem proving strategies with special properties, we start by introducing some basic concepts about theorem proving strategies.

A *theorem proving strategy* is a pair $\mathcal{P} =< I; \Sigma >$, where I is a set of *inference rules* and Σ is a *search plan*. Inference rules in I decide what consequences can be deduced from the available data and Σ decides which inference rule and which data to choose next. We discuss first the inference rules and next the search plan.

The general form of an inference rule f is:

f: $\frac{S}{S'}$

where S and S' are sets of sentences. The rule says that given S, the set S' can be inferred. We distinguish between *expansion* inference rules and *contraction* inference rules, as they are called in [36]. An expansion inference rule expands a given set S into a new set S' by deriving new sentences from sentences in S:

f: $\frac{S}{S'}$ where $S \subset S'$.

A contraction inference rule contracts a given set S into a new set S' by either deleting some sentences in S or replacing them by others:

f: $\frac{S}{S'}$ where $S \not\subseteq S'$.

We further distinguish between inference rules which transform the presentation and inference rules which transform the target. We assume that targets are clauses and therefore can be regarded as sets of literals:

- *Presentation inference rules*:
  - *Expansion inference rules*: f: $\frac{(S;\varphi)}{(S';\varphi)}$ where $S \subset S'$.
  - *Contraction inference rules*: f: $\frac{(S;\varphi)}{(S';\varphi)}$ where $S \not\subseteq S'$.
- *Target inference rules*:
  - *Expansion inference rules*: f: $\frac{(S;\varphi)}{(S;\varphi')}$ where $\varphi \subset \varphi'$.
  - *Contraction inference rules*: f: $\frac{(S;\varphi)}{(S;\varphi')}$ where $\varphi \not\subseteq \varphi'$.

Example 2.3.1 *In Unfailing Knuth-Bendix completion the presentation S is a set of equations and the target φ is a ground equation $\hat{s} \simeq \hat{t}$.* Deduction *of a critical pair is an expansion inference rule on the presentation, since it adds to the given set a new equation:*

$$\frac{(E \cup \{p \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{p \simeq q, l \simeq r, p[r]_u\sigma \simeq q\sigma\}; \hat{s} \simeq \hat{t})} \quad \begin{array}{l} p|u \notin X \\ p\sigma \not\preceq q\sigma, p[r]_u\sigma \end{array} \qquad (p|u)\sigma = l\sigma$$

where X is the set of variables of the signature, σ is the most general unifier of $p|u$ and l and $\succ$ is the assumed complete simplification ordering on terms. Deduction *works by checking for a* superposition *between two equations $p \simeq q$ and $l \simeq r$. Two given equations superpose if there exists a non variable subterm, say $p|u$, which unifies with mgu σ with a side l of the other equation $l \simeq r$. This means that the term $p\sigma$ is equal to both $q\sigma$ and $p[r]_u\sigma$. The new equation $p[r]_u\sigma \simeq q\sigma$ is called a* critical pair. *A critical pair is generated only if $p\sigma \not\preceq q\sigma, p[r]_u\sigma$, that is the two equations are applied according to the simplification ordering.* Simplification *is a contraction inference rule which applies to both the presentation:*

$$\frac{(E \cup \{p \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{p[r\sigma]_u \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})} \quad \begin{array}{l} p|u = l\sigma \qquad\qquad p \succ p[r\sigma]_u \\ p \triangleright l \vee q \succ p[r\sigma]_u \end{array}$$

and the target:

$$\frac{(E \cup \{l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{l \simeq r\}; \hat{s}[r\sigma]_u \simeq \hat{t})} \quad \begin{array}{l} \hat{s}|u = l\sigma \\ \hat{s} \succ \hat{s}[r\sigma]_u \end{array}$$

The conditions $p \succ p[r\sigma]_u$ and $\hat{s} \succ \hat{s}[r\sigma]_u$ ensure that simplification is well founded and $p \trianglerighteq l \vee q \succ p[r\sigma]_u$ is explained as follows. Since $p|u = l\sigma$, $p \trianglerighteq l$ holds. If $p \trianglerighteq l$, either p is a proper instance of l or l matches a proper subterm of p. The side l of the applied equation is strictly smaller than the simplified term according to the encompassment ordering and this is a sufficient condition, together with $p \succ p[r\sigma]_u$, for the simplification step to reduce the equational proofs where $p \simeq q$ occurs. If $p \stackrel{\bullet}{=} l$, p and l are variants, and therefore uncomparable in the encompassment ordering. In this case, $q \succ p[r\sigma]_u$ is required to hold, that is the newly generated term $p[r\sigma]_u$ is smaller than both sides of the simplified equation $p \simeq q$. This condition represents a restriction only if $q \not\succ p$, that is simplification applies to the greater side of $p \simeq q$ or $p \simeq q$ is not ordered. If $q \succ p$, simplification applies to the smaller side of $p \simeq q$ and the condition $q \succ p[r\sigma]_u$ is trivially satisfied. We shall see in the following how these conditions guarantee that a simplification step reduces the equational proofs where $p \simeq q$ occurs.

The inference rules are required to be sound. A presentation inference rule is *sound* if $Th(S') \subseteq Th(S)$, since an application of a presentation rule involves the presentation only. A target inference rule is *sound* if $Th(S \cup \{\varphi'\}) \subseteq Th(S \cup \{\varphi\})$.

Deduction by presentation rules is deduction of consequences from the axioms or *forward reasoning*. A target rule applies to both the presentation and the target to infer a new target: deduction by target rules is *backward reasoning*. The Knuth-Bendix procedure computing critical pairs and simplifying rewrite rules to obtain a canonical system performs exclusively forward reasoning. If a theorem to be proved is given as target, the procedure also performs some backward reasoning, when it simplifies the target.

Finally, a search plan Σ decides which inference rule should be applied to what data at any given step during a derivation. A very basic search plan simply sorts the sentences in the data set by a first-in first-out criterion: the sentences which have been generated first are selected first. Such a search plan is very inefficient because it is not based on any knowledge of the state of the proof. Better search plans are those based on some measure of the distance between the current state of the proof and the final proof. Since the final proof is unknown, this distance cannot be determined during the derivation and such a measure is a heuristic. Once the measure has been

defined, the best choice is the one which minimizes that distance. An example of a measure is the complexity of the clauses according to a simplification ordering: since the trivial theorem *true* is the smallest, the smallest clauses in S_i are selected at stage i. This example shows that a heuristic measure of the distance between the current state of the computation and a successful state may be described by a *well founded ordering* on data. In addition, the search plan may set a *precedence* on the inference rules and proceed accordingly:

Example 2.3.2 *A* Simplification-first *search plan [48] for Unfailing Knuth-Bendix completion is a search plan where Simplification has priority over Deduction. Therefore Deduction is considered only if Simplification does not apply to any equation. Equations can be sorted by the multiset extension $\succ_{mul}$ of the ordering on terms, or by size, or by age such as in a first-in first-out plan.*

Different schemes for inference rules, called *deduction* and *deletion*, are given in [16]. The deduction scheme in [16] is the same as our expansion scheme. The deletion scheme instead is different from our contraction scheme, since it only allows to infer S' from S by deleting a sentence in S. If these deduction/deletion schemes are adopted, an inference rule which replaces a sentence by other sentences has to be schematized as the composition of a deduction rule and a deletion rule. For instance, Simplification of $p \simeq q$ into $p[r\sigma]_u \simeq q$ is described as the generation of the equation $p[r\sigma]_u \simeq q$ followed by the deletion of $p \simeq q$.

This approach has the substantial drawback that it requires to consider more general inference rules than those actually included in the set of inference rules of the given strategy. For Simplification of $p \simeq q$ into $p[r\sigma]_u \simeq q$, the equation $p[r\sigma]_u \simeq q$ may not be a critical pair derivable by a step of the Deduction rule. This is the case if $q \succ p$, that is the right hand side of a rewrite rule is simplified. A general paramodulation inference rule, which is not featured by the Unfailing Knuth-Bendix procedure, is then required in order to simulate Simplification. We prefer our expansion/contraction schemes, since they allow us to classify directly every concrete inference rule of a given strategy as either an expansion rule or a contraction rule.

2.4 Completion procedures

A completion procedure $\mathcal{C}$ has then three components, $\mathcal{C} =< I_p, I_t; \Sigma >$, where I_p is the set of presentation inference rules, I_t is the set of target inference rules and Σ is the search plan.

A derivation by a completion procedure is a process of *proof reduction*. A target inference step modifies the target and therefore it affects the proof of the target. We require that the proof of the target is reduced:

Definition 2.4.1 *A target inference step $(S;\varphi) \vdash (S;\varphi')$ is* proof-reducing *if $\Pi(S,\varphi) \geq_p \Pi(S,\varphi')$. It is* strictly proof-reducing *if $\Pi(S,\varphi) >_p \Pi(S,\varphi')$.*

Example 2.4.1 *Simplification of the target as given in Example 2.3.1 is strictly proof-reducing. If we assume the proof ordering $>_u$ introduced in Example 2.2.2, we have $\{\hat{s},\hat{t}\} \succ_{mul} \{\hat{s}',\hat{t}'\}$, since $\hat{s} \succ \hat{s}'$ and $\hat{t} = \hat{t}'$, assuming $\hat{s}$ is simplified to $\hat{s}'$. Therefore $\Pi(E, \hat{s} \simeq \hat{t}) >_u \Pi(E, \hat{s}' \simeq \hat{t}')$.*

For a presentation inference step we allow more flexibility:

Definition 2.4.2 *Given two pairs $(S;\varphi)$ and $(S';\varphi')$, the relation $(S;\varphi) \rhd_{p,\mathcal{T}} (S';\varphi')$ holds if*

1. *either $\Pi(S,\varphi) >_p \Pi(S',\varphi')$*
2. *or*
   (a) *$\Pi(S,\varphi) = \Pi(S',\varphi')$,*
   (b) *$\forall \psi \in \mathcal{T}$, $\Pi(S,\psi) \geq_p \Pi(S',\psi)$ and*
   (c) *$\exists \psi \in \mathcal{T}$ such that $\Pi(S,\psi) >_p \Pi(S',\psi)$.*

Definition 2.4.3 *A presentation inference step $(S;\varphi) \vdash (S';\varphi)$ is* proof-reducing *on $\mathcal{T}$ if $(S;\varphi) \rhd_{p,\mathcal{T}} (S';\varphi)$ holds. It is* strictly proof-reducing *if $\Pi(S,\varphi) >_p \Pi(S',\varphi)$.*

The condition $(S_i;\varphi_i) \rhd_{p,\mathcal{T}} (S_{i+1};\varphi_{i+1})$ says that either the step $(S_i;\varphi_i) \vdash (S_{i+1};\varphi_{i+1})$ reduces the proof of the target, or it reduces the proof of at least one theorem in $\mathcal{T}$, while it does not increase the proof of any theorem in $\mathcal{T}$. A step which reduces the proof of the target is proof-reducing, regardless of its effects on other theorems. On the other hand, an inference step on the presentation may not immediately decrease the proof of the target and still be necessary to decrease it eventually. Such a step is proof-reducing too, if it does not increase any proof and strictly decreases at least one.

Example 2.4.2 *Deduction of a critical pair as given in Example 2.3.1 is proof-reducing. We assume the proof ordering $>_u$ introduced in Example 2.2.2. Given two equations $l \simeq r$ and $p \simeq q$, a* critical overlap *of $l \simeq r$ and $p \simeq q$ is any proof $s \leftarrow_{l\simeq r} v \rightarrow_{p\simeq q} t$, where v is $c[p\tau]$ for some context c and substitution τ and $(p|u)\tau = l\tau$ for some non variable subterm $p|u$ of p. The Deduction rule applied to $l \simeq r$ and $p \simeq q$ generates the critical pair $p[r]_u\sigma \simeq q\sigma$, where σ is the mgu of $p|u$ and l and therefore $\tau = \sigma\rho$ for some substitution ρ. Such a Deduction step affects a minimal proof by replacing any occurrence of the critical overlap $s \leftarrow_{l\simeq r} v \rightarrow_{p\simeq q} t$ by the equational step $s \leftrightarrow_{p[r]_u\sigma\simeq q\sigma} t$, justified by the critical pair. We have $\{(v,l,s),(v,p,t)\} >^e_{mul} \{(s,p[r]_u\sigma,t)\}$ or $\{(v,l,s),(v,p,t)\} >^e_{mul} \{(t,q\sigma,s)\}$, depending on whether $s \succ t$ or $t \succ s$, since $v \succ s$ and $v \succ t$. Therefore $\Pi(E,\psi) >_u \Pi(E \cup \{p[r]_u\sigma \simeq q\sigma\},\psi)$ if a minimal proof of ψ in E contains a critical overlap between $l \simeq r$ and $p \simeq q$, $\Pi(E,\psi) = \Pi(E \cup \{p[r]_u\sigma \simeq q\sigma\},\psi)$ otherwise. If a minimal proof of the target contains a critical overlap of $l \simeq r$ on $p \simeq q$, the Deduction step is strictly proof-reducing.*

Example 2.4.3 *Simplification of an equation in the presentation, given in Example 2.3.1, is proof-reducing. We assume the proof ordering $>_u$ introduced in Example 2.2.2. A Simplification step where an equation $p \simeq q$ is simplified to $p[r\sigma]_u \simeq q$ by an equation $l \simeq r$, affects a minimal proof by replacing a step $s \leftrightarrow_{p\simeq q} t$ by two steps $s \rightarrow_{l\simeq r} v \leftrightarrow_{p[r\sigma]_u\simeq q} t$.*

- *If $t \succ s$, we have $\{(t,q,s)\} >^e_{mul} \{(s,l,v),(t,q,v)\}$ since $t \succ s$ and $s \succ v$.*
- *If $s \succ t$,*

- *if* $p \triangleright l$, *we have*
  - *if* $t \succ v$, $\{(s,p,t)\} >^e_{mul} \{(s,l,v),(t,q,v)\}$ *since* $p \triangleright l$ *and* $s \succ t$,
  - *if* $v \succ t$, $\{(s,p,t)\} >^e_{mul} \{(s,l,v),(v,q,t)\}$ *since* $p \triangleright l$ *and* $s \succ v$;
- *if* $p \stackrel{\bullet}{=} l$ *and* $q \succ p[r\sigma]_u$, $t \succ v$ *follows from* $q \succ p[r\sigma]_u$ *by stability and monotonicity of* $\succ$ *and we have* $\{(s,p,t)\} >^e_{mul} \{(s,l,v),(t,q,v)\}$ *since* $t \succ v$ *and* $s \succ t$.

Therefore $\Pi(E \cup \{p \simeq q\}, \psi) >_u \Pi(E \cup \{p[r\sigma]_u \simeq q\}, \psi)$ *if a minimal proof of* ψ *in* E *contains a step* $s \leftrightarrow_{p \simeq q} t$, $\Pi(E \cup \{p \simeq q\}, \psi) = \Pi(E \cup \{p[r\sigma]_u \simeq q\}, \psi)$ *otherwise. Similar to Deduction, if a step* $s \leftrightarrow_{p \simeq q} t$ *occurs in a minimal proof of the target, the Simplification step is strictly proof-reducing.*

This notion of proof reduction applies to presentation inference steps which are either expansion steps or contraction steps which replace some sentences by others. A contraction step which deletes a sentence φ from a set S without adding any cannot reduce any minimal proof: it is impossible that $\Pi(S, \psi) <_p \Pi(S \cup \{\varphi\}, \psi)$ for some ψ, since $\Pi(S, \psi)$ is a proof in $S \cup \{\varphi\}$. In order to characterize these steps, we introduce a notion of *redundancy*:

Definition 2.4.4 *A sentence* φ *is* redundant *in* S *on domain* $\mathcal{T}$ *if* $\forall \psi \in \mathcal{T}$, $\Pi(S, \psi) = \Pi(S \cup \{\varphi\}, \psi)$.

A sentence is redundant in a presentation if it does not affect any minimal proof.

Example 2.4.4 *An inference rule of Unfailing Knuth-Bendix completion, which deletes an equation without adding any is* Functional subsumption*:*

$$\frac{(E \cup \{p \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{l \simeq r\}; \hat{s} \simeq \hat{t})} \; (p \simeq q) \triangleright (l \simeq r)$$

Functional subsumption *deletes an equation* $p \simeq q$ *because it is subsumed by another equation* $l \simeq r$, *that is* $p = c[l\sigma]$ *and* $q = c[r\sigma]$ *for some context* c *and substitution* σ, *where either* c *is not empty or* σ *is not a renaming of variables. An equation* $p \simeq q$ *subsumed by* $l \simeq r$ *is redundant according to the proof ordering* $>^e_{mul}$ *and therefore to the proof ordering* $>_u$ *as defined in Example 2.2.2. No minimal proof contains a step* $s \leftrightarrow_{p \simeq q} t$ *since the step* $s \leftrightarrow_{l \simeq r} t$ *is smaller: either* $\{(s,p,t)\} >^e_{mul} \{(s,l,t)\}$ *or* $\{(t,q,s)\} >^e_{mul} \{(t,r,s)\}$, *depending on whether* $s \succ t$ *or* $t \succ s$, *since* $p \triangleright l$ *and* $q \triangleright r$.

A notion of redundant clauses was introduced in [74] and in [16], where the term "redundant" was first used. We shall discuss redundancy in more detail in the next chapter, where we shall show that our notion of redundancy is more general than those given in [74] and [16]. The following notion of *reduction* encompasses both proof reduction and deletion of redundant elements:

Definition 2.4.5 *An inference step* $(S; \varphi) \vdash (S'; \varphi')$ *is* reducing *on* $\mathcal{T}$ *if either it is proof-reducing on* $\mathcal{T}$ *or it delete sentences which are redundant in* S *on domain* $\mathcal{T}$.

Definition 2.4.6 *An inference rule* f *is* reducing *if all the inference steps* $(S; \varphi) \vdash_f (S'; \varphi')$ *where* f *is applied are reducing.*

We have finally all the elements to define a completion procedure:

Definition 2.4.7 *A theorem proving strategy* $\mathcal{C} =< I_p, I_t; \Sigma >$ *is a* completion procedure *on domain* $\mathcal{T}$ *if for all pairs* $(S_0; \varphi_0)$*, where* S_0 *is a presentation of a theory and* $\varphi_0 \in \mathcal{T}$*, the derivation*

$(S_0; \varphi_0) \vdash_{\mathcal{C}} (S_1; \varphi_1) \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} (S_i; \varphi_i) \vdash_{\mathcal{C}} \ldots$

has the following properties:

- *monotonicity:* $\forall i \geq 0$, $Th(S_{i+1}) \subseteq Th(S_i)$,
- *relevance:* $\forall i \geq 0$, $\varphi_i \in \mathcal{T}$ *and* $\varphi_{i+1} \in Th(S_{i+1})$ *if and only if* $\varphi_i \in Th(S_i)$ *and*
- *reduction:* $\forall i \geq 0$, *the step* $(S_i; \varphi_i) \vdash_{\mathcal{C}} (S_{i+1}; \varphi_{i+1})$ *is reducing on* $\mathcal{T}$.

The *domain* $\mathcal{T}$ is the set of sentences where the inference rules of the completion procedure are reducing. For instance, for the Knuth-Bendix completion procedure $\mathcal{T}$ is the set of all equations. For the Unfailing Knuth-Bendix procedure, $\mathcal{T}$ is the set of all ground equations.

The *monotonicity* and *relevance* properties establish the soundness of the presentation and the target inference rules respectively. Monotonicity ensures that a presentation inference step does not create new elements which are not true in the theory, while relevance ensures that a target inference step replaces the target by a new target in such a way that proving the latter is equivalent to prove the former. For instance, a simplification step which reduces a target φ to φ' satisfies the relevance requirement because φ' is true if and only if φ is true. An interesting expansion inference rule for the target, called *Ordered saturation*, will be described in detail in Section 6.3.

Reduction is the property which characterizes completion procedures. Clearly, if all the inference rules of a procedure are reducing, the procedure has the reduction property. We have seen in the above examples and we shall see in Chapter 6 that the inference rules of the known equational completion procedures are reducing. Most inference rules are reducing because they are suitably restricted by the complete simplification ordering $\succ$ on terms. A complete simplification ordering on data turns out to be a key element in characterizing a theorem proving strategy as a completion procedure.

Chapter 3

Theorem proving

Completeness of a completion procedure for theorem proving involves both the inference rules and the search plan. First, it requires that whenever $\varphi_0 \in Th(S_0)$ for the given input $(S_0; \varphi_0)$, there exist successful derivations by the inference rules of the procedure. Second, it requires that whenever successful derivations exist, the search plan guarantees that the computed derivation is successful. We call these properties *refutational completeness* of the inference rules and *fairness* of the search plan, respectively. We use refutational completeness for the inference rules in order to reserve *completeness* for the entire strategy given by inference rules and search plan.

Many refutationally complete sets of inference rules are known, but relatively little attention has been given to the fairness problem. A search plan which exhausts the entire search space is obviously fair, albeit grossly inefficient. Therefore, the problem is to reconciliate fairness and efficiency. This problem becomes even more intricate in the presence of contraction inference rules.

Fairness of completion-based methods has been discussed in [11, 74, 16]. A derivation by a completion procedure is fair according to [11] if all the critical pairs from *persisting* equations are eventually generated and all the generated equations are eventually reduced. This definition of fairness captures exactly the requirements that a derivation generating a confluent system must satisfy, but it is not intended for a derivation proving a specified theorem. Indeed, most theorem proving derivations do not satisfy this definition of fairness, since proving a specific theorem does not require in general to generate all the critical pairs.

Thus, a search plan which is fair according to these definitions may force the prover to perform deductions completely irrelevant to prove the intended theorem. In this chapter we define refutational completeness of the inference rules and fairness of the search plan in terms of proof reduction, according to the framework developed in the previous chapter. We prove that if the inference rules are refutationally complete and the search plan is fair with respect to our definitions, the completion procedure is complete and therefore it is a *semidecision procedure* for theorem proving. In our view, this is the most important interpretation of completion.

Our approach to the problem of fairness is new in three respects, all relevant to theorem proving. First, our definition is *target-oriented*, that is it takes the theorem to be proved into account. A target-oriented definition of fairness is important for search plans to be both fair and

efficient. For instance, we may have a problem $(E; \hat{s} \simeq \hat{t})$, where the target $s \simeq t$ is an equation on a signature F_1 and the input presentation E is the union of a set E_1 of equations on the signature F_1 and a set E_2 of equations on another signature F_2, disjoint from F_1. Such a problem can occur in definitions of abstract data types, where the signature F_1 contains the constructors and a set of defined symbols, whereas the signature F_2 is another set of defined symbols. According to our definition, a derivation from $(E; \hat{s} \simeq \hat{t})$, where no inference from E_2 is performed is fair. On the other hand, fairness as defined in previous works on completion [11, 74, 16] would require to compute critical pairs from the equations in E_2 as well.

Second, we emphasize the distinction between fairness as a property of the search plan and completeness as a property of the inference rules. Most theorem proving strategies are simply presented by giving a set of inference rules and the task of designing a suitable search plan is left to the implementation phase. This is not satisfactory, since the actual performance of the prover depends heavily on the search plan. Therefore, we think that it is important to study search plans systematically and define their requirements formally. Our approach to the fairness problem is a first step in this direction. Finally, it is generally acknowledged that the application of contraction inference rules is mandatory for efficiency. However, the known definitions of fairness emphasize the application of expansion rules, whereas ours treats both expansion and contraction rules uniformly.

In the previous chapter we introduced a notion of redundancy. The interest in redundancy of data elements in a theorem proving derivation resides in the importance of contraction inference rules. Contraction inference rules are necessary to make theorem proving feasible. However, few contraction rules are known. The purpose of studying redundancy is to get some insight about how to design new and powerful contraction rules. Therefore we conclude this chapter on theorem proving with a discussion on redundancy.

3.1 Completion procedures as semidecision procedures

Given an input pair $(S_0; \varphi_0)$, a completion procedure works by reducing the proof $\Pi(S_0, \varphi_0)$. If the proof of the target is minimal, the process halts. Since the empty proof is smaller than any other proof, the computation halts at stage k if $\Pi(S_k, \varphi_k)$ is empty and φ_k is *true*.

A procedure is *complete* if, whenever φ_0 is a theorem of S_0, the derivation from $(S_0; \varphi_0)$ reduces φ_0 to *true* and halts. This requires *refutational completeness* of the inference rules and *fairness* of the search plan.

In order to describe these two properties, we introduce a structure called *I-tree*. Given a theorem proving problem $(S_0; \varphi_0)$ and a set of inference rules I, the application of I to $(S_0; \varphi_0)$ defines a tree, the *I-tree rooted at* $(S_0; \varphi_0)$. The nodes of the tree are labeled by pairs $(S; \varphi)$. The root is labeled by the input pair $(S_0; \varphi_0)$. A node $(S; \varphi)$ has a child $(S'; \varphi')$ if $(S'; \varphi')$ can be derived from $(S; \varphi)$ in one step by an inference rule in I. The I-tree rooted at $(S_0; \varphi_0)$ represents all the possible derivations by the inference rules in I starting from $(S_0; \varphi_0)$.

Intuitively, a set I of inference rules is *refutationally complete* if whenever $\varphi_0 \in Th(S_0)$, the I-tree rooted at $(S_0; \varphi_0)$ contains successful nodes, nodes of the form $(S; true)$. The following

definition is an equivalent characterization of this concept in terms of proof reduction:

Definition 3.1.1 *A set $I = I_p \cup I_t$ of inference rules is* refutationally complete *if whenever $\varphi \in Th(S)$ and $\Pi(S, \varphi)$ is not minimal, there exist derivations*

$$(S; \varphi) \vdash_I (S_1; \varphi_1) \vdash_I \ldots \vdash_I (S'; \varphi')$$

such that $\Pi(S, \varphi) >_p \Pi(S', \varphi')$.

This definition says that a set of inference rules is refutationally complete if it can reduce the proof of the target whenever it is not minimal. Since a proof ordering is well founded, it follows that if $\varphi \in Th(S)$, the I-tree rooted at $(S; \varphi)$ contains successful nodes. The advantage of giving the definition of completeness in terms of proof reduction is that the problem of proving completeness of I is reduced to the problem of exhibiting a suitable proof ordering.

Given a completion procedure $\mathcal{C} =< I_p, I_t; \Sigma >$, $I = I_p \cup I_t$, the I-tree rooted at $(S_0; \varphi_0)$ represents the entire search space that the procedure can potentially derive from the input $(S_0; \varphi_0)$. The search plan Σ selects a path in the I-tree: the derivation from input $(S_0; \varphi_0)$ controlled by Σ is the path selected by Σ in the I-tree rooted at $(S_0; \varphi_0)$. Once both a set of inference rules and a search plan are given, the derivation from $(S_0; \varphi_0)$ is unique. A pair $(S_i; \varphi_i)$ reached at stage i of the derivation is a *visited node* in the I-tree. Each visited node $(S_i; \varphi_i)$ has generally many children, but the search plan selects only one of them to be $(S_{i+1}; \varphi_{i+1})$. A search plan Σ is *fair* if whenever the I-tree rooted at $(S_0; \varphi_0)$ contains successful nodes, the derivation controlled by Σ starting at $(S_0; \varphi_0)$ is guaranteed to reach a successful node. Similar to completeness, we rephrase this concept in terms of proof reduction:

Definition 3.1.2 *A derivation*

$$(S_0; \varphi_0) \vdash_{\mathcal{C}} (S_1; \varphi_1) \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} (S_i; \varphi_i) \vdash_{\mathcal{C}} \ldots$$

controlled by a search plan Σ is fair *if and only if for all $i \geq 0$, if there exists a path*

$$(S_i; \varphi_i) \vdash_I \ldots \vdash_I (S'; \varphi')$$

in the I-tree rooted at $(S_0; \varphi_0)$ such that $\Pi(S_i, \varphi_i) >_p \Pi(S', \varphi')$, then there exists an $(S_j; \varphi_j)$ for some $j > i$, such that $\Pi(S', \varphi') \geq_p \Pi(S_j, \varphi_j)$. A search plan Σ is fair *if all the derivations controlled by Σ are fair.*

In other words, if the inference rules allow to reduce the proof of the target at $(S_i; \varphi_i)$, a fair search plan guarantees that the proof of the target will be indeed reduced at a later stage $(S_j; \varphi_j)$.

If the inference rules are complete and the search plan is fair, the completion procedure is *complete*:

Theorem 3.1.1 *Let $\mathcal{C} =< I_p, I_t; \Sigma >$ be a completion procedure. If the set $I = I_p \cup I_t$ of inference rules is refutationally complete and the search plan Σ is fair, then for all derivations*

$$(S_0; \varphi_0) \vdash_{\mathcal{C}} (S_1; \varphi_1) \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} (S_i; \varphi_i) \vdash_{\mathcal{C}} \ldots,$$

where $\varphi_0 \in Th(S_0)$, $\forall i \geq 0$, if $\Pi(S_i, \varphi_i)$ is not minimal, then there exists an (S_j, φ_j), for some $j > i$, such that $\Pi(S_i, \varphi_i) >_p \Pi(S_j, \varphi_j)$.

Proof: if $\Pi(S_i, \varphi_i)$ is not minimal, then by completeness of the inference rules, there exists a path $(S_i; \varphi_i) \vdash_I \ldots \vdash_I (S'; \varphi')$ such that $\Pi(S_i, \varphi_i) >_p \Pi(S', \varphi')$. By fairness of the search plan, there exists an $(S_j; \varphi_j)$, for some $j > i$, such that $\Pi(S_i; \varphi_i) >_p \Pi(S', \varphi') \geq_p \Pi(S_j, \varphi_j)$. □

Completeness means that the completion procedure is a *semidecision procedure* for $Th(S) \cap \mathcal{T}$, where S is any presentation and $\mathcal{T}$ is the domain of the completion procedure:

Corollary 3.1.1 *If a completion procedure $\mathcal{C}$ on domain $\mathcal{T}$ has refutationally complete inference rules and fair search plan, then for all inputs $(S_0; \varphi_0)$ where $\varphi_0 \in Th(S_0)$, the derivation*

$$(S_0; \varphi_0) \vdash_{\mathcal{C}} (S_1; \varphi_1) \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} (S_i; \varphi_i) \vdash_{\mathcal{C}} \ldots$$

reaches a stage k, $k \geq 0$, such that φ_k is the clause true.

Proof: if $\varphi_0 \in Th(S_0)$, then by Theorem 3.1.1 and the well foundedness of $>_p$ the derivation reaches a stage k such that the proof $\Pi(S_k, \varphi_k)$ is minimal. Since the minimal proof is the empty proof, φ_k is the clause *true*. □

3.2 Redundancy

A notion of redundant clauses appeared in [74] and in [16], where the term "redundant" was first used. Redundant clauses according to [74] and [16] are redundant in our sense. On the other hand, there are clauses which are intuitively redundant and redundant according to our definition, but not according to the definitions in [74] and [16].

Definition 3.2.1 (Rusinowitch 1987) [74] *A clause φ is* R-redundant *in a set S if there exists a clause $\psi \in S$ such that ψ properly subsumes φ, i.e. $\varphi \succ\!\!\!\cdot\, \psi$, where $\succ\!\!\!\cdot$ is the proper subsumption ordering on clauses.*

The definition of redundancy in [16] assumes a well founded ordering $>^d$ total on ground clauses[1]:

Definition 3.2.2 (Bachmair and Ganzinger 1990) [16] *A clause φ is* B-redundant *in a set S if for all ground instances $\varphi\sigma$ of φ, there are ground instances $\psi_1 \ldots \psi_n$ of clauses in S such that $\{\psi_1 \ldots \psi_n\} \models \varphi\sigma$ and $\forall j, 1 \leq j \leq n, \varphi\sigma >^d \psi_j$.*

It is immediate to see that if the ordering $>^d$ is the proper subsumption ordering, Definition 3.2.2 specializes to Definition 3.2.1 and therefore a clause which can be subsumed is an example of a B-redundant clause:

Lemma 3.2.1 (Bachmair and Ganzinger 1990) [16] *If a clause is R-redundant, then it is B-redundant.*

In our view, the intuition behind the notion of redundancy is that a clause φ is redundant in S if adding φ to S does not decrease any minimal proof in S (Definition 2.4.4). In fact our definition captures the meaning of Definition 3.2.2:

[1] In [16] a notion of *deletion ordering* is defined for this purpose. Well foundedness and totality on ground clauses are the only properties of a deletion ordering which are relevant to our discussion.

Theorem 3.2.1 *If a clause φ is B-redundant in S, then it is R-redundant in S on the domain of all ground clauses.*

Proof: we assume a proof ordering $>_p$ on ground proofs such that the minimal proof of a ground clause φ in S, $\Pi(S, \varphi)$, is the smallest set $\{\psi_1 \dots \psi_n\}$ of ground instances of clauses in S such that $\{\psi_1 \dots \psi_n\} \models \varphi$, according to the multiset extension ${>^d}_{mul}$ of the ordering $>^d$. Since $>^d$ is well founded and total on ground clauses, $>_p$ is well defined. Let φ be a B-redundant clause in S. We show that $\Pi(S \cup \{\varphi\}, \psi) = \Pi(S, \psi)$ for all ground theorems ψ. Since $S \subset S \cup \{\varphi\}$, $\Pi(S \cup \{\varphi\}, \psi) \leq_p \Pi(S, \psi)$ trivially holds and therefore we simply have to show that $\Pi(S \cup \{\varphi\}, \psi) \not<_p \Pi(S, \psi)$. The proof is done by way of contradiction: if $\Pi(S \cup \{\varphi\}, \psi) <_p \Pi(S, \psi)$, then the smallest set of ground instances of clauses in $S \cup \{\varphi\}$ which logically entails ψ has the form $S' \cup \{\varphi\sigma_1 \dots \varphi\sigma_k\}$ for some set S' of ground instances of clauses in S and some ground substitutions $\sigma_1 \dots \sigma_k$. Since φ is B-redundant in S, for all $\varphi\sigma_i$, $1 \leq i \leq k$, there are ground instances $\{\psi_1^i \dots \psi_n^i\}$ of clauses in S such that $\{\psi_1^i \dots \psi_n^i\} \models \varphi\sigma_i$ and $\varphi\sigma_i >^d \psi_j^i$, $\forall j, 1 \leq j \leq n$. Therefore, $S' \cup \{\psi_1^i \dots \psi_n^i\}_{i=1}^k <^d_{mul} S' \cup \{\varphi\sigma_1 \dots \varphi\sigma_k\}$ and $S' \cup \{\psi_1^i \dots \psi_n^i\}_{i=1}^k \models \psi$, that is $S' \cup \{\varphi\sigma_1 \dots \varphi\sigma_k\}$ cannot be the smallest set entailing ψ. It follows that $\Pi(S \cup \{\varphi\}, \psi) = \Pi(S, \psi)$. □

On the other hand, there are cases where trivially redundant clauses are not redundant according to Definition 3.2.2, whereas they are redundant according to our definition:

Example 3.2.1 *If $S = \{P, \neg R, R\}$, where P and R are ground atoms, P is intuitively redundant and it is redundant according to our Definition 2.4.4: the minimal proof of every ground theorem is given by $\{\neg R, R\}$, since $\{\neg R, R\}$ yields the empty clause and therefore any clause. However, if $R \succ P$ and thus $R >^d P$, P is not B-redundant.*

This example shows that a notion of redundancy based on an ordering on clauses is not ideal, since different precedences on predicate symbols may be needed in order to characterize as redundant different clauses during a computation. A notion of redundancy based on a proof ordering seems to behave more satisfactorily.

We conclude with some discussion about the relationship between redundancy and contraction inference rules. In the previous chapter we have used redundancy to characterize those contraction rules which delete sentences. It is immediate to show that redundancy plays a role also for contraction rules which replace a sentence by others. If a contraction inference step $(S \cup \{\psi\}; \varphi) \vdash (S \cup \{\psi'\}; \varphi)$ is proof-reducing by Condition 2 in Definition 2.4.2, the deleted sentence ψ is clearly redundant in $S \cup \{\psi'\}$. However, this is not necessarily true for a contraction rule which is proof-reducing by Condition 1 in Definition 2.4.2, since Condition 1 simply says that the proof of the target is strictly reduced, without any provision for the proofs of the other theorems. A contraction inference rule may also delete non redundant sentences, provided it is sound and it strictly reduces the proof of the target.

This is a further difference between our approach and the one in [16]. In [16] redundancy is used to define the deletion scheme itself: a deletion inference rule is a rule which deletes redundant sentences. We prefer not to relate so tightly the notions of contraction and redundancy, because the notions of redundancy proposed so far are not target-oriented. According to the three definitions of redundancy we have considered, a sentence is redundant if it is useless with respect to

all the theorems in the domain. Intuitively, a redundant sentence is a sentence which is obvious to discard. Our definition of proof reduction (Definition 2.4.2) allows in principle very strong contraction rules which may delete even non redundant sentences if they do not help in proving a specific target. Further work is necessary to turn an abstract study of contraction and redundancy into concrete inference rules.

Chapter 4

Generation of decision procedures

So far we have regarded a completion procedure as a theorem proving procedure. In this chapter we extend our framework to include completion procedures as *generators of decision procedures*. In this context derivations by completion have the form

$(S_0; \emptyset) \vdash_{\mathcal{C}} (S_1; \emptyset) \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} (S_i; \emptyset) \vdash_{\mathcal{C}} \ldots$.

No target is given: the purpose of the derivation is to transform the presentation only.

If the search plan of a completion procedure satisfies a stronger fairness property, which we call *uniform fairness*, the procedure generates a possibly infinite *saturated* presentation. Uniform fairness is the classical fairness property required in previous work on completion procedures [52, 11, 74, 16]. It basically consists in eventually considering all the inference steps. The concept of saturated set is a generalization of confluent system: a presentation is saturated if no non-trivial consequences can be added [63, 16]. We define both uniform fairness and saturated set in terms of our notion of redundancy and we give a new proof of the theorem which says that a uniformly fair derivation generates a possibly infinite saturated presentation. This theorem covers the generation of confluent rewrite systems by Knuth-Bendix completion [62, 52, 8], the generation of ground confluent sets of equations by Unfailing Knuth-Bendix completion [48, 14] and the generation of saturated sets in [63, 16]. Our theorem is more general than those in [63, 16], because our notion of redundancy is more general, as discussed in Section 3.2.

If a presentation is saturated, the derivations from that presentation are *linear input* derivations [24], that is derivations where each inference step applies to the target. If linear input derivations from a saturated set are guaranteed to be well founded, a saturated set is a *decision procedure* and the completion procedure is a *generator of decision procedures*.

The well foundedness of the derivations is implied by additional requirements, which depend on the logic. In equational logic, a derivation $s \to^* \circ \leftarrow^* t$ made only of well founded simplification steps by a ground confluent set of equations is a well founded linear input derivation and a ground confluent equational presentation is a decision procedure for theorems in the form $\forall \bar{x} s \simeq t$. Sufficient conditions for well foundedness of derivations for ground targets in Horn logic with equality are also known [63, 16].

Our notions of uniform fairness, saturated set, well founded linear input derivation, decision procedure and the theorems relating them give an abstract presentation of the application of

completion to the generation of decision procedures, which covers and unify all the known results about generation of decision procedures in equational logic and Horn logic with equality.

Few theories have a finite saturated presentation and even fewer satisfy the additional requirement for a saturated presentation to be a decision procedure. Therefore, the interpretation of completion as semidecision procedure is more useful in practice.

4.1 Uniform fairness and saturated sets

Definition 3.1.2 of fairness is sufficient for theorem proving as shown by Theorem 3.1.1. If Definition 3.1.2 is applied to Knuth-Bendix completion, it is not sufficient to guarantee that a confluent rewrite system is eventually generated, since it does not guarantee that all critical pairs are eventually considered. This requires a much more demanding fairness property, which we call *uniform fairness*.

The first definition of uniform fairness appeared in [52], where it is required that the search plan sorts the rewrite rules in the data set by a well founded ordering, in order to ensure that no rule is indefinitely postponed. We recall this very first notion of fairness, because it states explicitly that fairness is a property of the search plan. The later definitions of fairness are given at a much higher abstraction level, which may prevent the reader from seeing that fairness is a property of the search mechanism.

Given a derivation by completion starting from a presentation S_0, we denote by $S_\infty = \bigcup_{j\geq 0}\bigcap_{i\geq j} S_i$ the possibly infinite set of all the *persistent* sentences, that is the sentences which are generated at some stage and never deleted afterwards. The set S_∞ is called the *limit* of the derivation. The advantage of dealing with the limit S_∞ is that if the derivation halts at some stage k, $S_\infty = S_k$. Therefore, properties stated in terms of the limit S_∞ apply uniformly to both halting and non halting derivations. This concept has been introduced first in [52] and used among others in [8, 11, 33, 16]. We also denote by $I_e(S)$ the set of clauses which can be generated in one step by the presentation expansion rules of the completion procedure applied to S.

Definition 4.1.1 (Rusinowitch 1987) [74], (Bachmair and Ganzinger 1990) [16] *A derivation*

$(S_0;\emptyset)\vdash_{\mathcal{C}}(S_1;\emptyset)\vdash_{\mathcal{C}}\ldots(S_i;\emptyset)\vdash_{\mathcal{C}}\ldots$

by a completion procedure $\mathcal{C}$ on domain $\mathcal{T}$ is uniformly fair *on domain $\mathcal{T}$ if $\forall\varphi\in I_e(S_\infty)$, there exists an S_j for some $j\geq 0$ such that either $\varphi\in S_j$ or φ is redundant in S_j on domain $\mathcal{T}$.*

This definition of fairness generalizes previous definitions given in [52] and [11]. It says that every clause φ that can be persistently generated by an expansion rule during the completion process is either actually generated at some step, or replaced by other clauses which yield a smaller proof of φ and make it redundant. For instance, a Knuth-Bendix derivation such that all critical pairs from persisting equations are eventually generated or subsumed or reduced to a common term is uniformly fair according to this definition.

Uniform fairness has been studied and progressively refined in [11, 74, 16] with the purpose of solving the problem of the interaction between expansion inference rules and contraction inference

rules. The intuitive meaning of uniform fairness is to be fair to the inference rules, that is to apply all the inference rules to all the data. However, this is impossible, because the inference rules include both contraction rules and expansion rules: if a clause φ is deleted by a contraction step before an expansion rule f is applied to φ, the derivation is not fair to f. The problem has been then to define fairness in such a way that the application of contraction rules is fair. This problem is solved in the definition of uniform fairness by establishing that it is fair not to perform an expansion inference step if its premises are not persistent and it is fair to replace a clause φ by clauses which make it redundant. In this way it is fair to apply contraction inference rules such as simplification and subsumption. In actuality, a uniformly fair procedure will perform exhaustively all expansion steps which are not inhibited by contraction steps.

Fairness and uniform fairness are different in three basic points. Fairness is *target-oriented*, whereas uniform fairness is defined for a derivation without target, because it does not take the target and therefore the target inference rules into consideration. Indeed Definition 4.1.1 is not a definition of fairness for theorem proving. In [74, 16], Definition 4.1.1 is applied to refutational theorem proving, where S_0 contains the negation of the target. In this case the only persisting clause is the empty clause $\Box$ and $I_e(\bigcup_{i\geq 0}\bigcap_{j\geq i} S_j) = \{\Box\}$. Then Definition 4.1.1 says that the limit of the derivation is the empty clause. A notion of fairness given in terms of the limit does not represent useful information for the design of search plans because it does not say anything about how a search plan should choose the successor at any given stage of the derivation.

The definition of fairness does not differentiate between expansion rules and contraction rules and between persisting and non-persisting clauses, because the interaction of expansion and contraction rules is no longer the issue. All inference rules are treated uniformly by considering their effect with respect to the goal of reducing the proof of the target. On the other hand, expansion rules and contraction rules are treated differently in the definition of uniform fairness. Uniform fairness emphasizes fairness with respect to the expansion inference rules, while the role of contraction inference rules is buried in the restriction to persistent clauses. The reason is that it is necessary to consider all critical pairs in order to obtain a confluent set, but it is not necessary to reduce them, although in practice completion without simplification is hopelessly inefficient.

The following example illustrates a set of conditions for an Unfailing Knuth-Bendix derivation which have been proved in [11] to be sufficient for uniform fairness. These conditions represent the most well known definition of (uniform) fairness for a completion procedure:

Example 4.1.1 *A derivation*

$(E_0;\emptyset)\vdash_{UKB}(E_1;\emptyset)\vdash_{UKB}\ldots\vdash_{UKB}(E_i;\emptyset)\vdash_{UKB}\ldots$

is uniformly fair if

- *for all critical pairs* $g \simeq d \in I_e(E_\infty)$, $g \simeq d \in \bigcup_{i\geq 0} E_i$ *and*
- E_∞ *is reduced.*

The Deduction inference rule given in Example 2.3.1 is the only expansion inference rule of Unfailing Knuth-Bendix completion. Therefore the first condition says that all critical pairs derivable by Deduce from persisting equations are eventually generated. The second condition says that all

persisting equations are eventually simplified as much as possible. As was remarked above, the application of contraction rules is allowed but not required: the first condition alone is sufficient for uniform fairness. Since at any stage of the computation it is not known which equations are going to persist and which equations are going to be simplified, the above conditions for uniform fairness prescribe in practice to apply exhaustively all the inference rules of Unfailing Knuth-Bendix completion until none applies.

The concept of uniform fairness leads to the following notion of *saturated* presentation:

Definition 4.1.2 (Kounalis and Rusinowitch 1988) [63], (Bachmair and Ganzinger 1990) [16] *A presentation S is* saturated *on the domain $\mathcal{T}$ of a completion procedure if and only if $\forall \psi \in I_e(S)$, either $\psi \in S$ or ψ is redundant in S on $\mathcal{T}$.*

No non-trivial consequences can be added to a saturated presentation. In the equational case, as remarked in [63], a set of equations is saturated if no non-trivial critical pairs can be deduced, or equivalently, the set is *locally confluent*[1]. As in the definition of uniform fairness, the application of contraction inference rules is allowed but not required: contraction inference rules may still be applicable to a saturated set. A locally confluent equational presentation is not necessarily reduced.

If a derivation is uniformly fair, S_∞ is saturated. Since uniform fairness is defined in terms of redundancy and our notion of redundancy is more general than those in [74] and [16], we give a new proof of this result:

Theorem 4.1.1 (Kounalis and Rusinowitch 1988) [63], (Bachmair and Ganzinger 1990) [16] *If a derivation*

$$(S_0;\emptyset) \vdash_{\mathcal{C}} (S_1;\emptyset) \vdash_{\mathcal{C}} \ldots (S_i;\emptyset) \vdash_{\mathcal{C}} \ldots$$

is uniformly fair on domain $\mathcal{T}$*, then S_∞ is* saturated *on domain $\mathcal{T}$.*

Proof: we show that for all $\varphi \in I_e(S_\infty)$, either $\varphi \in S_\infty$ or φ is redundant in S_∞ on $\mathcal{T}$. By uniform fairness of the derivation, there exists an S_j, for some $j \geq 0$, such that either $\varphi \in S_j$ or φ is redundant in S_j on $\mathcal{T}$:

1. if $\varphi \in S_j$, then either φ is not deleted afterwards, that is $\varphi \in S_\infty$, or φ is deleted at some stage $i > j$. There are in turn two cases: either φ is simply deleted or it is replaced by another sentence φ':
   (a) let S_i be $S \cup \{\varphi\}$ and S_{i+1} be S. By Definition 2.4.7 of completion, such a step deletes a redundant sentence. Then φ is redundant in S_i on $\mathcal{T}$. Since by Definition 2.4.7 of completion, $\forall \psi \in \mathcal{T}$, $\Pi(S_i, \psi) \geq_p \Pi(S_\infty, \psi)$, φ is also redundant in S_∞ on $\mathcal{T}$.
   (b) if $S_i = S \cup \{\varphi\}$ and $S_{i+1} = S \cup \{\varphi'\}$, this step is proof-reducing, that is $\forall \psi \in \mathcal{T}$, $\Pi(S_i, \psi) \geq_p \Pi(S_{i+1}, \psi)$ and $\exists \psi \in \mathcal{T}$ such that $\Pi(S_i, \psi) >_p \Pi(S_{i+1}, \psi)$. If φ itself

[1] A critical pair $g \simeq d$ is trivial in E if $g \rightarrow_E^* \circ \leftarrow_E^* d$. It is well known, by the Knuth-Bendix lemma [62, 51, 8] and its extensions to the UKB context [48, 14], that a set of equations is locally confluent if and only if it has no non-trivial critical pairs.

represents a minimal proof of φ in S_i, then there exists a minimal proof $\Pi(S_{i+1}, \varphi)$ in S_{i+1} such that $\varphi \geq_p \Pi(S_{i+1}, \varphi)$. Since $\varphi \notin S_{i+1}$, $\Pi(S_{i+1}, \varphi)$ is not φ itself and therefore $\varphi >_p \Pi(S_{i+1}, \varphi) \geq_p \Pi(S_\infty, \varphi)$. If φ does not represent a minimal proof of φ in S_i, then there exists a minimal proof $\Pi(S_i, \varphi)$ of φ in S_i such that $\Pi(S_i, \varphi) <_p \varphi$ and therefore $\varphi >_p \Pi(S_i, \varphi) \geq_p \Pi(S_{i+1}, \varphi) \geq_p \Pi(S_\infty, \varphi)$. In both cases there exists a minimal proof $\Pi(S_\infty, \varphi)$ of φ in S_∞ such that $\varphi >_p \Pi(S_\infty, \varphi)$ and by monotonicity and stability of $>_p$, $P[\varphi\sigma] >_p P[\Pi(S_\infty, \varphi)\sigma]$ for all proof contexts P and substitutions σ. In other words, φ is not involved in any minimal proof in S_∞ of a theorem of $\mathcal{T}$, since any occurrence of φ in a proof can be replaced by a proof $\Pi(S_\infty, \varphi)$ smaller than φ itself. It follows that φ is redundant in S_∞ on $\mathcal{T}$.

2. If φ is redundant in S_j on $\mathcal{T}$, since $\forall \psi \in \mathcal{T}$, $\Pi(S_j, \psi) \geq_p \Pi(S_\infty, \psi)$ by Definition 2.4.7 of completion, φ is redundant in S_∞ on $\mathcal{T}$. □

This theorem generalizes the following classical results:

Theorem 4.1.2 (Knuth and Bendix 1970) [62], (Huet 1981) [52], (Bachmair, Dershowitz and Hsiang 1986) [8] *If a derivation*

$$(E_0; \emptyset) \vdash_{KB} (E_1; \emptyset) \vdash_{KB} \ldots (E_i; \emptyset) \vdash_{KB} \ldots$$

by the Knuth-Bendix completion procedure does not fail and is uniformly fair *on the domain $\mathcal{T}$ of all equations, then E_∞ is a* confluent *term rewriting system.*

Knuth-Bendix completion fails if an unoriented equation persists. If a derivation by Knuth-Bendix completion does not fail, all the persistent equations are oriented into rewrite rules according to a reduction ordering and therefore E_∞ is a terminating rewrite system. By Theorem 4.1.1, E_∞ is saturated, i.e. locally confluent. By Newman's lemma [34], a terminating rewrite system is confluent if and only if it is locally confluent. Therefore E_∞ is confluent.

Theorem 4.1.3 (Hsiang and Rusinowitch 1987) [48], (Bachmair, Dershowitz and Plaisted 1989) [14] *If a derivation*

$$(E_0; \emptyset) \vdash_{UKB} (E_1; \emptyset) \vdash_{UKB} \ldots (E_i; \emptyset) \vdash_{UKB} \ldots$$

by the Unfailing Knuth-Bendix completion procedure is uniformly fair *on the domain $\mathcal{T}$ of all ground equations, then E_∞ is a* ground confluent *set of equations.*

For this second result we recall that given a set of equations E, $s \to_E t$ if $s \leftrightarrow_E t$ and $s \succ t$ for a reduction ordering $\succ$. The Unfailing Knuth-Bendix procedure assumes that $\succ$ is a complete simplification ordering. Since a complete simplification ordering is total on ground terms, $\leftrightarrow_{E_\infty} = \to_{E_\infty} \cup \leftarrow_{E_\infty}$ holds for ground terms and E_∞ is Church-Rosser on ground terms if and only if it is ground confluent. Since a complete simplification ordering is well founded, E_∞ is terminating on ground terms. The domain $\mathcal{T}$ of Unfailing Knuth-Bendix is the set of ground equations. By Theorem 4.1.1, E_∞ is saturated on $\mathcal{T}$, i.e. it is locally confluent on ground terms. By Newman's lemma E_∞ is ground confluent and therefore Church-Rosser on ground terms.

The Church-Rosser property on ground terms is important because $E \models \forall \bar{x} s \simeq t$ if and only if $\hat{s} \leftrightarrow_E^* \hat{t}$. If E is Church-Rosser on ground terms, $\hat{s} \leftrightarrow_E^* \hat{t}$ if and only if $\hat{s} \rightarrow_E^* \circ \leftarrow_E^* \hat{t}$ and therefore $E \models \forall \bar{x} s \simeq t$ can be decided by well founded reduction by E. This introduces us to the topic of the following section.

4.2 Decision procedures

Since adding sentences to a saturated presentation is redundant, it is possible to prove theorems from a saturated presentation by performing target inference steps only. A derivation where at each step a new target is obtained from the given target and an axiom of the input presentation is called a *linear input* derivation [24]. If, in addition, all steps are strictly proof-reducing, we have a *linear input reducing* derivation:

Definition 4.2.1 *A derivation*

$(S_0; \varphi_0) \vdash_{\mathcal{C}} (S_1; \varphi_1) \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} (S_i; \varphi_i) \vdash_{\mathcal{C}} \ldots$

by a completion procedure $\mathcal{C}$ is

- *a* linear input *derivation if all its steps are target inference steps,*
- *a* linear input reducing *derivation if all its steps are strictly proof-reducing target inference steps.*

Since a proof ordering is well founded, a linear input reducing derivation is guaranteed to halt. We say that a presentation is *strictly proof-reducing* if all the target steps by that presentation are strictly proof-reducing:

Definition 4.2.2 *Let $\mathcal{C}$ be a completion procedure on domain $\mathcal{T}$. A presentation S of a theory is* strictly proof-reducing *on $\mathcal{T}$ if $\forall \varphi \in \mathcal{T}$ any target inference step $(S; \varphi) \vdash_{\mathcal{C}} (S; \varphi')$ is strictly proof-reducing.*

In Unfailing Knuth-Bendix completion, the target inference rule Simplification itself is strictly proof-reducing, as shown in Example 2.4.1. Therefore, any set of equations is trivially a strictly proof-reducing presentation and any derivation made only of simplification steps is a linear input reducing derivation. This is because Simplification is restricted by a complete simplification ordering in such a way that any sequence of simplification steps is well founded.

For the following theorem we recall that given a set of inference rules I, we denote by I_e the subset of the expansion inference rules on the presentation and by I_t the subset of the target inference rules:

Theorem 4.2.1 *Let I be a refutationally complete set of inference rules, such that the subset $I_e \cup I_t$ is also refutationally complete. If a presentation S of a theory is* saturated *and* strictly proof-reducing *on $\mathcal{T}$, then $\forall \varphi_0 \in \mathcal{T}$, there exists a* linear input reducing *derivation*

$(S; \varphi_0) \vdash_{\mathcal{I}} (S; \varphi_1) \vdash_{\mathcal{I}} \ldots \vdash_{\mathcal{I}} (S; \varphi_i) \vdash_{\mathcal{I}} \ldots.$

Proof: since $I_e \cup I_t$ is refutationally complete, we can restrict our attention to the derivations by $I_e \cup I_t$. First we show that no rule in I_e applies to S. If an expansion inference rule f derives S' from S, then, since f is proof-reducing, there is a $\psi \in \mathcal{T}$ such that $\Pi(S, \psi) >_p \Pi(S', \psi)$. Since S is saturated on $\mathcal{T}$, this is impossible. Then, the derivation is made of target inference steps only and therefore it is a linear input derivation. Furthermore, the target steps by S are strictly proof-reducing by hypothesis and therefore the derivation is linear input reducing. □

In the equational case, a ground confluent set of equations is saturated and strictly proof-reducing and therefore it yields linear input derivations made of well founded simplification steps only.

Definition 4.2.3 *Let $\mathcal{C}$ be a complete completion procedure on domain $\mathcal{T}$. A presentation S of a theory is a* decision procedure *for $Th(S) \cap \mathcal{T}$, if for all $\varphi_0 \in \mathcal{T}$, the derivation*

$$(S; \varphi_0) \vdash_{\mathcal{C}} (S; \varphi_1) \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} (S; \varphi_i) \vdash_{\mathcal{C}} \ldots,$$

is guaranteed to halt at a stage k, for some $k > 0$, such that $\varphi_k = true$ if $\varphi_0 \in Th(S)$ and $\varphi_k \neq true$ if $\varphi_0 \notin Th(S)$.

We regard the presentation S as an algorithm, which, if interpreted by the procedure $\mathcal{C}$, decides validity of sentences in $\mathcal{T}$ in the theory of S. No inference step is performed on S itself and the derivation halts at stage k if no target inference rule applies to φ_k.

Lemma 4.2.1 *Let $\mathcal{C}$ be a completion procedure on domain $\mathcal{T}$ with refutationally complete inference rules I and fair search plan Σ. If a presentation S of a theory is such that for all $\varphi_0 \in \mathcal{T}$, there exists a* linear input reducing *derivation*

$$(S; \varphi_0) \vdash_{\mathcal{I}} (S; \varphi_1) \vdash_{\mathcal{I}} \ldots \vdash_{\mathcal{I}} (S; \varphi_i) \vdash_{\mathcal{I}} \ldots,$$

the presentation S is a decision procedure *for $Th(S) \cap \mathcal{T}$.*

Proof: the existence of linear input reducing derivations from $(S; \varphi_0)$ guarantees that the I-tree rooted at $(S; \varphi_0)$ contains some halting paths, since by the well foundedness of $>_p$ a linear input reducing derivation is guaranteed to halt. Therefore, by fairness of the search plan Σ, for all $\varphi_0 \in \mathcal{T}$, the computed derivation from $(S; \varphi_0)$ is guaranteed to halt at some stage k. Either φ_k is *true* or φ_k is not *true*. By completeness of I, φ_k is *true* if and only if $\varphi_0 \in Th(S)$. Therefore, S is a decision procedure for $Th(S) \cap \mathcal{T}$. □

Corollary 4.2.1 *A presentation S of a theory which is* saturated *and* strictly proof-reducing *on $\mathcal{T}$ is a decision procedure for $\mathcal{T} \cap Th(S)$.*

Proof: it follows from Theorem 4.2.1 and Lemma 4.2.1. □

Finally, we can characterize a completion procedure as a *generator of decision procedures*:

Definition 4.2.4 *A completion procedure $\mathcal{C}$ has the* strong monotonicity *property if for all steps $(S; \varphi) \vdash_{\mathcal{C}} (S'; \varphi')$, $Th(S') = Th(S)$.*

Corollary 4.2.2 *Let $\mathcal{C}$ be a completion procedure on domain $\mathcal{T}$ such that*

- *the set of inference rules I is refutationally complete,*
- *the search plan Σ is uniformly fair and*
- *the procedure satisfies the strong monotonicity property.*

For all presentations S_0, if the limit S_∞ of the derivation

$(S_0;\emptyset) \vdash_{\mathcal{C}} (S_1;\emptyset) \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} (S_i;\emptyset) \vdash_{\mathcal{C}} \ldots$

is strictly proof-reducing on $\mathcal{T}$, then S_∞ is a decision procedure for $\mathcal{T} \cap Th(S_0)$.

Proof: by Theorem 4.1.1, S_∞ is saturated on $\mathcal{T}$. Since it is strictly proof-reducing on $\mathcal{T}$, S_∞ is a decision procedure for $\mathcal{T} \cap Th(S_\infty)$ by Corollary 4.2.1. By the strong monotonicity property, $Th(S_\infty) = Th(S_0)$ and therefore S_∞ is a decision procedure for $\mathcal{T} \cap Th(S_0)$. □

In equational logic, a ground confluent set of equations E is a *decision procedure* for the validity of theorems in the form $\forall \bar{x} s \simeq t$. Sufficient conditions for a presentation in Horn logic with equality to be strictly proof-reducing have been given in [63] and [16]:

Definition 4.2.5 (Kounalis and Rusinowitch 1988) [63] *A Horn clause $A \vee \neg B_1 \vee \ldots \vee \neg B_n$ is* ground preserving *if the following two conditions hold:*

- *all variables occurring in a negated literal also occur in A and*
- *if A is an equation $s \simeq t$, either it can be oriented into a rewrite rule or s and t have the same set of variables.*

Theorem 4.2.2 (Kounalis and Rusinowitch 1988) [63], (Bachmair and Ganzinger 1990) [16] *A saturated presentation of ground preserving clauses in Horn logic with equality is a decision procedure for targets in the form $B_1 \wedge \ldots \wedge B_m$, where each B_i is a ground positive literal.*

The presentation is assumed to be saturated by the expansion inference rules of the completion procedures for Horn logic with equality given in [63] and [16]. This result fits in our framework as follows. The ground preserving condition is designed to ensure that whenever an inference step is applied between a clause in the saturated set and a ground target, the newly generated target is ground as well. Furthermore, the resolution and paramodulation inference rules in [63] and [16] are *ordered*, i.e. they are restricted by a given complete simplification ordering on terms and literals in such a way that at each step a ground literal in the target is replaced by a set of smaller ground literals. Therefore ordered resolution and ordered paramodulation between a ground preserving clause and a ground target are strictly proof-reducing. A presentation made only of ground preserving clauses is then strictly proof-reducing for ground targets and if it is also saturated, it is a decision procedure by Theorem 4.2.1, Lemma 4.2.1 and Corollary 4.2.1. The restriction to ground preserving clauses is quite strong. For instance the Horn clause $T(x,y) \vee \neg R(x,z) \vee \neg T(z,y)$ in the definition of the transitive closure T of a relation R is not ground preserving, because of the variable z. This is not surprising, since few theories have decision procedures.

Chapter 5

The Knuth-Bendix-Huet theorem

We have seen that if the search plan is fair, a completion procedure is a semidecision procedure and if the search plan is uniformly fair, a completion procedure is a generator of decision procedures. In this chapter we show the relation between these two interpretations of completion. Namely, we show that a completion procedure which is able to eventually generate a decision procedure is itself a semidecision procedure. This theorem generalizes the landmark result in [52], where the interpretation of completion as semidecision procedure was first given.

5.1 A general Knuth-Bendix-Huet theorem

We start by generalizing the notion of *failure* of a derivation by Knuth-Bendix completion:

Definition 5.1.1 *A uniformly fair derivation*

$(S_0;\emptyset)\vdash_{\mathcal{C}}(S_1;\emptyset)\vdash_{\mathcal{C}}\ldots\vdash_{\mathcal{C}}(S_i;\emptyset)\vdash_{\mathcal{C}}\ldots$

by a completion procedure $\mathcal{C}$ on domain $\mathcal{T}$ fails *if for some $\varphi\in\mathcal{T}\cap Th(S_\infty)$ there is no linear input reducing derivation from $(S_\infty;\varphi)$.*

In the equational case a linear input reducing derivation is a derivation made only of well founded simplification steps. Equivalently, a linear input reducing derivation is a derivation which computes a rewrite proof $s\to^*\circ\leftarrow^* t$.

A Knuth-Bendix derivation fails if and only if it generates a persistent equation which cannot be ordered by the given reduction ordering. The existence of such an equation $l\simeq r\in E_\infty$ is a sufficient and necessary condition for failure. It is sufficient because $l\simeq r$ is itself a theorem for which there is no rewrite proof in E_∞: $l\leftrightarrow_{l\simeq r} r$ is not a rewrite proof and l and r are E_∞-irreducible, since $l\simeq r$ is persistent. It is necessary because if there is an equation $s\simeq t\in Th(E_\infty)$ which does not have a rewrite proof in E_∞, then $\Pi(E_\infty, s\simeq t)$ contains either a peak $\leftarrow\circ\rightarrow$ or an unoriented step $\leftrightarrow$. Since the derivation is uniformly fair, E_∞ is locally confluent and a minimal proof in E_∞ does not contain peaks. Then $\Pi(E_\infty, s\simeq t)$ contains an unoriented step, which means E_∞ contains and unoriented equation.

Failures are handled by the search plan. A search plan for Knuth-Bendix completion may

decree a failure either as soon as an unordered equation is generated or when no inference rule applies and the current data set contains unordered equations.

Unfailing Knuth-Bendix completion does not fail because it assumes a total ordering on ground terms and its domain is restricted to ground equations. Since the domain is restricted to ground equations, only ground equational proofs in E_∞ are relevant. All ground proof steps can be ordered, because the ordering is total on ground terms. Since the derivation is uniformly fair, E_∞ is locally confluent on ground terms and minimal ground equational proofs in E_∞ do not contain peaks, that is for all theorems $\forall \bar{x} s \simeq t \in Th(E_\infty)$ there is a rewrite proof in E_∞.

In Horn logic with equality, we know by Theorem 4.2.2 that if S_∞ is saturated and all its clauses are ground preserving, then for all $\varphi \in \mathcal{T} \cap Th(S_\infty)$ there is a linear input reducing derivation from $(S_\infty; \varphi)$, where $\mathcal{T}$ is the domain of all conjunctions of ground positive literals. By the above definition of failure, if a uniformly fair derivation fails, then for some $\varphi \in \mathcal{T} \cap Th(S_\infty)$ there is no linear input reducing derivation from $(S_\infty; \varphi)$. Then either S_∞ is not saturated or S_∞ contains a clause which is not ground preserving. Since the derivation is uniformly fair, S_∞ is indeed saturated. It follows that S_∞ contains a clause which is not ground preserving. In other words, a violation of the ground preserving requirement is a necessary condition for failure. However, it is not sufficient, as the following example shows:

Example 5.1.1 *The input set S_0 is given by the clauses defining the transtive closure T of a relation R*

$T(x, y) \vee \neg R(x, y)$,

$T(x, y) \vee \neg R(x, z) \vee \neg T(z, y)$

and some pairs in R

$R(a, b)$,

$R(b, c)$,

$R(c, d)$.

We assume the precedence $R > T > d > c > b > a$ and an ordering on terms and literals as defined in Section 2.1. The saturated presentation computed by ordered resolution contains among others all the input clauses and all the facts about T which can be deduced from the input:

$T(a, b)$,

$T(b, c)$,

$T(c, d)$,

$T(a, c)$,

$T(b, d)$,

$T(a, d)$.

The second clause in the input is not ground preserving and therefore the saturated set contains a clause which is not ground preserving. However, the saturated presentation contains all the true facts about the relations R and T, so that any target given by a conjunctions of ground positive literals can be proved by a linear input reducing derivation made only of ground ordered resolution

steps between the target and the facts in the saturated set.

Since the ground preserving property is a sufficient, but not necessary condition for a saturated presentation of Horn clauses to yield linear input reducing derivations, it follows that a violation of the ground preserving requirement is a necessary, but not sufficient condition for failure. To our knowledge, the problem of finding a necessary and sufficient characterization of failure for Horn logic with equality is still open.

The notion of failure is assumed in the following theorem:

Theorem 5.1.1 *Let $\mathcal{C}$ be a completion procedure on domain $\mathcal{T}$ such that*

- *the set of inference rules I is refutationally complete,*
- *the search plan Σ is fair and uniformly fair and*
- *the procedure satisfies the strong monotonicity property.*

For all derivations

$(S_0;\varphi_0)\vdash_{\mathcal{C}}(S_1;\varphi_1)\vdash_{\mathcal{C}}\ldots\vdash_{\mathcal{C}}(S_i;\varphi_i)\vdash_{\mathcal{C}}\ldots,$

if $\varphi_0 \in Th(S_0)$ and the derivation does not fail, then there exists a $k \geq 0$ such that φ_k is the clause true.

Proof: since Σ is uniformly fair, S_∞ is saturated by Theorem 4.1.1. By strong monotonicity, $Th(S_\infty) = Th(S_0)$ holds. Since the derivation does not fail, for all $\varphi_0 \in \mathcal{T} \cap Th(S_0)$, there exists a successful linear input reducing derivation

$(S_\infty;\varphi_0)\vdash_I(S_\infty;\varphi_1)\vdash_I\ldots\vdash_I(S_\infty;true).$

This derivation is finite and therefore it involves only a finite subset S of S_∞. Since S_∞ is the set of all persistent sentences generated during the derivation, S is a finite set of persistent sentences. It follows that there exists an S_j, for some $j \geq 0$, such that $S \subseteq S_j$ and the above linear input reducing derivation in S_∞ can be performed in S_j:

$(S_j;\varphi_0)\vdash_I(S_j;\varphi_1)\vdash_I\ldots\vdash_I(S_j;true).$

The I-tree rooted at $(S_0;\varphi_0)$ contains the path

$(S_0;\varphi_0)\vdash_I\ldots\vdash_I(S_j;\varphi_0)\vdash_I(S_j;\varphi_1)\vdash_I\ldots\vdash_I(S_j;true).$

More generally, the I-tree rooted at $(S_0;\varphi_0)$ contains successful nodes. Then, by fairness of the search plan, the computed derivation reaches one such node. □

The above theorem does not say that the path

$(S_0;\varphi_0)\vdash_I\ldots\vdash_I(S_j;\varphi_0)\vdash_I(S_j;\varphi_1)\vdash_I\ldots\vdash_I(S_j;true)$

is the one actually computed by the procedure. This is not true in general. One reason is that the actual derivation may include target inference steps by non persisting elements of the presentation.

Both uniform fairness and fairness are required: uniform fairness ensures that the limit is saturated, while fairness guarantees that the necessary target inference steps are eventually performed. Uniform fairness does not imply anything about application of the target inference rules.

This theorem generalizes *the Knuth-Bendix-Huet theorem*:

Theorem 5.1.2 (Huet 1981) [52] *For all uniformly fair and fair derivations*

$(E_0; \hat{s}_0 \simeq \hat{t}_0) \vdash_{KB} (E_1; \hat{s}_1 \simeq \hat{t}_1) \vdash_{KB} \ldots \vdash_{KB} (E_i; \hat{s}_i \simeq \hat{t}_i) \vdash_{KB} \ldots$

by Knuth-Bendix completion, if $\forall \bar{x} s_0 \simeq t_0 \in Th(E_0)$ *and the derivation does not fail, then there exists a* $k \geq 0$ *such that* $\hat{s}_k = \hat{t}_k$.

According to the original formulation of this theorem, under the above hypotheses there exists a $k \geq 0$ such that $\hat{s}_0 \rightarrow^*_{E_0} \circ \rightarrow^*_{E_1} \ldots \rightarrow^*_{E_k} \circ \leftarrow^*_{E_k} \ldots \leftarrow^*_{E_1} \circ \leftarrow^*_{E_0} \hat{t}_0$. In our framework, the simplification steps of the above rewrite proof are the target inference steps performed during the derivation. The same result holds for Unfailing Knuth-Bendix, without the restriction to non failing derivations:

Theorem 5.1.3 (Hsiang Rusinowitch 1987) [48], (Bachmair, Dershowitz and Plaisted 1989 [14] *For all uniformly fair and fair derivations*

$(E_0; \hat{s}_0 \simeq \hat{t}_0) \vdash_{UKB} (E_1; \hat{s}_1 \simeq \hat{t}_1) \vdash_{UKB} \ldots \vdash_{UKB} (E_i; \hat{s}_i \simeq \hat{t}_i) \vdash_{UKB} \ldots$

by Unfailing Knuth-Bendix completion, if $\forall \bar{x} s_0 \simeq t_0 \in Th(E_0)$*, there exists a* $k \geq 0$ *such that* $\hat{s}_k = \hat{t}_k$.

Theorem 5.1.1 can be considered the first generalization of these two classical results. The study of saturated presentations for Horn logic with equality in [63, 16] fail to generalize the Knuth-Bendix-Huet theorem, because it does not relate the saturation process to the semidecision process. The fundamental concept in the Knuth-Bendix-Huet theorem is that if the limit of a derivation is saturated, any true target can be proved during the derivation, regardless of whether the limit is finite or not. The capability of the completion procedure to *semidecide* the validity of targets is a side effect of the saturation process, or, better, the saturation process itself *is* a semidecision process.

Both [63] and [16] focuse on the properties of a finite saturated set. The saturation process is conceived not as a semidecision process, but as a *compilation* process to generate a finite saturated set. Theorem proving is then regarded as a two-phase process: first compile the given presentation into a finite saturated one and next prove theorems in the saturated presentation. Accordingly, a motivation for the study of contraction, redundancy and ordering based restrictions to expansion inference rules in [63] and especially in [16] is termination of saturation. The purpose of these studies is to restrict the inference system, while still retaining completeness, in such a way that a finite saturated presentation can be obtained for a reasonably high number of interesting theories.

Our approach is different. We prefer to interpret completion as a semidecision procedure and we study contraction, redundancy and ordering based restrictions to expansion inference rules in order to make the semidecision process more efficient. This different perspective explains why we have proposed a new definition of fairness. In the two-phase approach it is impossible not to assume uniform fairness, since uniform fairness is necessary fo saturation. Instead, we have proved in Theorem 3.1.1 that a completion procedure is a semidecision procedure assuming only fairness of the search plan and refutational completeness of the inference rules. These are weaker

hypotheses than those in the Knuth-Bendix-Huet theorem. This result is important to us because it shows that the interpretation of completion as semidecision procedure can be decoupled from its interpretation as generator of saturated sets and therefore it is not just a side effect of the latter. In other words, it is not necessary to eventually generate a saturated set in order to have a semidecision procedure.

However, we also need our general Knuth-Bendix-Huet theorem to show that if both fairness and uniform fairness are assumed, semidecision and saturation coexist in the same process. Theorem 5.1.1 is general in that it does not depend on a specific logic. The only notion which depends on the logic is the notion of linear input reducing derivation and therefore the notion of failure. As we have discussed earlier, satisfactory characterizations of these notions are not known for logics larger than equational logic. We propose our general Knuth-Bendix-Huet theorem as a template theorem, which can be specialized to different logics by giving suitable characterizations of the notions of linear input reducing derivation and failure.

Chapter 6

Completion procedures in equational logic

In this chapter we give a new presentation of some Knuth-Bendix type completion procedures for equational logic in the framework developed so far. All the strategies we present are extensions or variations of the Unfailing Knuth-Bendix procedure. Unlike the standard Knuth-Bendix and Unfailing Knuth-Bendix procedures, these variations were never presented as sets of inference rules in a unified style. These procedures are much more interesting than the standard ones from the theorem proving point of view, because they are more efficient and indeed they have been successfully implemented [1, 2, 3, 5]. In particular this family of procedures includes the *Inequality Ordered-Saturation strategy*, a refined version of the Unfailing Knuth-Bendix procedure, which has given very good experimental results [3]. Since the presentation in [3] focuses especially on the experimental results, the treatment given here is the first theoretical description of the Inequality Ordered-Saturation strategy.

6.1 Unfailing Knuth-Bendix completion

The *Unfailing Knuth-Bendix procedure* [48, 14] is a semidecision procedure for equational theories. A presentation is a set of equations E_0 and a theorem is an equational theorem $\forall \bar{x} s_0 \simeq t_0$. A derivation by UKB has the form

$$(E_0; \hat{s}_0 \simeq \hat{t}_0) \vdash_{UKB} (E_1; \hat{s}_1 \simeq \hat{t}_1) \vdash_{UKB} \ldots (E_i; \hat{s}_i \simeq \hat{t}_i) \vdash_{UKB} \ldots$$

where we denote by $\hat{s}_0 \simeq \hat{t}_0$ an equation which contains only universally quantified variables and therefore can be regarded as a ground equation. A derivation halts at stage k if $\hat{s}_k$ and $\hat{t}_k$ are identical. We assume that the complete simplification ordering $\succ$ is such that $\forall s, s \succ \mathit{true}$. At each step of the completion process the pair $(E_{i+1}; \hat{s}_{i+1} \simeq \hat{t}_{i+1})$ is derived from the pair $(E_i; \hat{s}_i \simeq \hat{t}_i)$ by applying one of the following inference rules:

- *Presentation inference rules*:
  - *Simplification*:

$$\frac{(E \cup \{p \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{p[r\sigma]_u \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})} \quad \begin{array}{l} p|u = l\sigma \qquad\qquad p \succ p[r\sigma]_u \\ p \triangleright l \vee q \succ p[r\sigma]_u \end{array}$$

- *Deduction*:
$$\frac{(E \cup \{p \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{p \simeq q, l \simeq r, p[r]_u\sigma \simeq q\sigma\}; \hat{s} \simeq \hat{t})} \quad \begin{array}{l} p|u \notin X \qquad (p|u)\sigma = l\sigma \\ p\sigma \not\preceq q\sigma, p[r]_u\sigma \end{array}$$
- *Deletion*:
$$\frac{(E \cup \{l \simeq l\}; \hat{s} \simeq \hat{t})}{(E; \hat{s} \simeq \hat{t})}$$
- *Functional subsumption*:
$$\frac{(E \cup \{p \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{l \simeq r\}; \hat{s} \simeq \hat{t})} \; (p \simeq q) \triangleright (l \simeq r)$$

- *Target inference rules*:
  - *Simplification*:
$$\frac{(E \cup \{l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{l \simeq r\}; \hat{s}[r\sigma]_u \simeq \hat{t})} \quad \begin{array}{l} \hat{s}|u = l\sigma \\ \hat{s} \succ \hat{s}[r\sigma]_u \end{array}$$
  - *Deletion*:
$$\frac{(E; \hat{s} \simeq \hat{s}\})}{(E; true)}$$

We have already presented these inference rules through Examples 2.3.1, 2.4.1, 2.4.2, 2.4.3 and 2.4.4. We simply remark here that the original definitions of Simplification and Deduction given in [48] have slightly different conditions. Simplification requires that $l\sigma \succ r\sigma$ and Deduction requires that $p\sigma \not\preceq q\sigma$ and $l\sigma \not\preceq r\sigma$. We adopt here the conditions given in [36], because they put weaker requirements on Simplification and stronger requirements on Deduction than the original ones. However, these conditions may be more expensive to compute, since they require to perform both substitution application and term replacement.

Simplification is the most important among the above inference rules, because it reduces dramatically the number and the size of the generated equations. A search plan for UKB should give to Simplification the highest priority among all the inference rules, so that the target and the presentation are always kept fully simplified. A search plan with this property is called *Simplification-first* [48]. If Simplification is not applied, the Deduction inference rule rapidly saturates the memory space with equations, making impossible to reach a proof in reasonable time.

In order to characterize the UKB procedure as a completion procedure, we define a proof ordering $>_{UKB}$ to compare the proofs $\Pi(E_i, \hat{s}_i \simeq \hat{t}_i)$. We use the ordering $>_u$ introduced in Example 2.2.2. Then $\Pi(E_i, \hat{s}_i \simeq \hat{t}_i) >_{UKB} \Pi(E_j, \hat{s}_j \simeq \hat{t}_j)$ holds if and only if $(\{\hat{s}_i, \hat{t}_i\}, \hat{s}_i \leftrightarrow^*_{E_i} \hat{t}_i) >_u (\{\hat{s}_j, \hat{t}_j\}, \hat{s}_j \leftrightarrow^*_{E_j} \hat{t}_j)$ holds.

Lemma 6.1.1 *The presentation inference rules of the UKB procedure are reducing.*

Proof: Deduction and Simplification are proof-reducing, Deletion and Functional subsumption delete redundant equations:

- The proof for Deduction was given in Example 2.4.2.
- The proof for Simplification was given in Example 2.4.3.
- A trivial equation $l \simeq l$ is redundant: no minimal proof contains a step $s \leftrightarrow_{l \simeq l} s$ since the subproof given by the single term s is smaller: $\{(s, l, s)\} >^{e}_{mul} \{\epsilon\}$, where the empty triple ϵ is the proof complexity of s.
- The proof for Functional subsumption was given in Example 2.4.4. □

Lemma 6.1.2 *The target inference rules of the UKB procedure are strictly proof-reducing.*

Proof: the proof for Simplification was given already in Example 2.4.1. For a Deletion step we have $\{\hat{s}_i, \hat{t}_i\} \succ_{mul} \{true\}$, since *true* is smaller than any term. Therefore $\Pi(E_i, \hat{s}_i \simeq \hat{t}_i) >_{UKB} \Pi(E_i, \hat{s}_{i+1} \simeq \hat{t}_{i+1})$. □

We can then show that UKB is a completion procedure:

Theorem 6.1.1 *The Unfailing Knuth-Bendix procedure is a completion procedure on the domain $\mathcal{T}$ of all ground equations.*

Proof: for all equational presentations E_0 and for all ground targets $\hat{s}_0 \simeq \hat{t}_0$ the derivation

$$(E_0; \hat{s}_0 \simeq \hat{t}_0) \vdash_{UKB} (E_1; \hat{s}_1 \simeq \hat{t}_1) \vdash_{UKB} \ldots (E_i; \hat{s}_i \simeq \hat{t}_i) \vdash_{UKB} \ldots$$

has the monotonicity, relevance and reduction properties. Monotonicity and relevance follow by soundness of the inference rules, which is proved among others in [52, 8, 11]. Reduction follows from Lemma 6.1.1 and Lemma 6.1.2. □

If a *fair* search plan is provided, the UKB procedure is a semidecision procedure for equational theories:

Theorem 6.1.2 (Hsiang and Rusinowitch 1987) [48], (Bachmair, Dershowitz and Plaisted 1989) [14] *An equation $\forall \bar{x} s \simeq t$ is a theorem of an equational theory E if and only if the Unfailing Knuth-Bendix procedure derives* true *from $(E; \hat{s} \simeq \hat{t})$.*

Implementations of the Unfailing Knuth-Bendix procedure are described in [69, 1, 2, 18, 3, 5].

6.2 Extensions: AC-UKB and cancellation laws

Many equational problems involve associative and commutative (AC) operators. An AC function f satisfies the equations

$f(f(x, y), z) \simeq f(x, f(y, z))$ *(associativity)* and

$f(x, y) \simeq f(y, x)$ *(commutativity).*

Handling associativity and commutativity as any other equation turns out to be very inefficient. Commutativity cannot be ordered and therefore it may generate a very high number of equations through the Deduction inference rule, whereas Simplification does not apply as often as it is desirable to reduce the size and the number of the equations. Associativity can be oriented into $f(f(x,y),z) \to f(x,f(y,z))$, but still it may trigger the generation of a very high number, even an infinite number, of equations, as the following example shows:

Example 6.2.1 *Given the associativity rule*

$$f(f(x,y),z) \to f(x,f(y,z))$$

and the rewrite rule

$$f(a,f(x,b)) \to f(x,b),$$

associativity superposes on the subterm $f(x,b)$ in the left hand side of the second rule, yielding the critical pair

$$f(f(x,y),b) \simeq f(a,f(x,f(y,b))),$$

which can be simplified and oriented into

$$f(a,f(x,f(y,b))) \to f(x,f(y,b)).$$

Again associativity superposes on the subterm $f(y,b)$ of the new rule, generating

$$f(a,f(x,f(y,f(z,b)))) \to f(x,f(y,f(z,b))).$$

Associativity superposes now on $f(z,b)$: in this way an infinite sequence of critical pairs can be generated.

The efficiency of the UKB strategy can be greatly improved if associativity and commutativity are not given in the input, but built in the inference rules. The UKB procedure with associativity and commutativity built in the inference rules is called *AC-UKB* [1]. The basic idea is to replace syntactic identity by equality *modulo AC*. Let AC be a set of associativity and commutativity axioms. Two terms s and t are equal modulo AC, if $s \simeq t$ is a theorem of AC, which we write $s \leftrightarrow^*_{AC} t$. The inference rules of the UKB procedure are modified in such a way that any two terms which are equal modulo AC are regarded as identical.

The first modification is to require that the complete simplification ordering on terms $\succ$ is in some sense "compatible" with replacing identity by equality modulo AC. More precisely, this "compatibility" requirement is a *commutation* property. Given two relations R and S, we say that R *commutes* over S if $S \circ R \subseteq R \circ S$, where $\circ$ is composition of relations. The complete simplification ordering $\succ$ is required to commute over $\leftrightarrow^*_{AC}$: this means that for any two terms s and t, if there is a third term r such that $s \leftrightarrow^*_{AC} r$ and $r \succ t$, there is also a term r' such that $s \succ r'$ and $r' \leftrightarrow^*_{AC} t$. Secondly, matching and unification are replaced by AC-matching and AC-unification. A term s matches a term t *modulo AC* if there is a substitution σ such that $s\sigma \leftrightarrow^*_{AC} t$. Similarly, two terms s and t unify *modulo AC* if there is a substitution σ such that $s\sigma \leftrightarrow^*_{AC} t\sigma$. Finally, the strict encompassment ordering $\triangleright$ is replaced by the ordering $\triangleright_{AC}$, that is $s \triangleright_{AC} t$ if and only if $s \triangleright r$ and $r \leftrightarrow^*_{AC} t$ for some term r.

The set of inference rules of the UKB procedure is therefore modified as follows:

- *Presentation inference rules*:
  - *Simplification*:
    $$\frac{(E \cup \{p \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{p[r\sigma]_u \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})} \quad \begin{array}{l} p|u \leftrightarrow^*_{AC} l\sigma \\ p \triangleright_{AC} l \vee q \succ p[r\sigma]_u \end{array} \qquad p \succ p[r\sigma]_u$$
  - *Deduction*:
    $$\frac{(E \cup \{p \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{p \simeq q, l \simeq r, p[r]_u\sigma \simeq q\sigma\}; \hat{s} \simeq \hat{t})} \quad \begin{array}{l} p|u \notin X \\ p\sigma \not\preceq q\sigma, p[r]_u\sigma \end{array} \qquad (p|u)\sigma \leftrightarrow^*_{AC} l\sigma$$
  - *Extension*:
    $$\frac{(E \cup \{f(p,q) \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{f(p,q) \simeq r, f(p,q,z) \simeq f(r,z)\}; \hat{s} \simeq \hat{t})} \quad \begin{array}{l} f \text{ is } AC \\ f(p,q) \not\preceq r \end{array}$$
  - *Deletion*:
    $$\frac{(E \cup \{l \simeq l\}; \hat{s} \simeq \hat{t})}{(E; \hat{s} \simeq \hat{t})}$$
  - *Functional subsumption*:
    $$\frac{(E \cup \{p \simeq q, l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{l \simeq r\}; \hat{s} \simeq \hat{t})} \; (p \simeq q) \triangleright_{AC} (l \simeq r)$$
- *Target inference rules*:
  - *Simplification*:
    $$\frac{(E \cup \{l \simeq r\}; \hat{s} \simeq \hat{t})}{(E \cup \{l \simeq r\}; \hat{s}[r\sigma]_u \simeq \hat{t})} \quad \begin{array}{l} \hat{s}|u \leftrightarrow^*_{AC} l\sigma \\ \hat{s} \succ \hat{s}[r\sigma]_u \end{array}$$
  - *Deletion*:
    $$\frac{(E; \hat{s} \simeq \hat{s})}{(E; true)}$$

This set of inference rules is obtained from the set of inference rules of the UKB procedure by replacing identity by equality modulo AC as explained above and by adding a new inference rule, called *Extension*. The *Extension* inference rule is a specialized version of the *Deduction* inference rule, designed to compute superpositions of equations in E onto associativity axioms. Namely, if $f(p,q) \simeq r$ is an equation in E, f is AC and $f(p,q) \not\preceq r$, the equation $f(p,q) \simeq r$ trivially superposes onto the associativity axiom $f(f(x,y),z) \simeq f(x,f(y,z))$, yielding the critical pair $f(p,f(q,z)) \simeq f(r,z)$, which we write in *flattened* form as $f(p,q,z) \simeq f(r,z)$. These critical pairs are called *extended rules*. Computing the extended rules is sufficient to ensure completeness of the AC-UKB procedure: no other critical pairs between E and AC need to be computed [70].

The extension of UKB to AC-UKB is feasible because algorithms for AC-matching and AC-unification are available. An algorithm for AC-unification, its application in a completion procedure and the extended rules first appeared in [76, 77, 70]. The correctness of the AC-unification algorithm was proved in [37]. General theoretical frameworks for working with equations modulo a set of axioms A are given in [55] and in [9]. These results are surveyed in [34] and more specifically for unification problems in [57].

The UKB or AC-UKB procedure can be further improved by building in the inference rules for the *cancellation laws*. A function F is *right cancellable* if it satisfies the *right cancellation law*

$$\forall x, y, z \;\; f(x,y) = f(z,y) \;\supset\; x = z$$

The *left cancellation law* is defined symmetrically. Cancellation laws may reduce considerably the size of the equations. They are implemented as inference rules as follows [49]:

Cancellation 1:

$$\frac{(E \cup \{f(p,u) \simeq f(q,v)\}; \hat{s} \simeq \hat{t})}{(E \cup \{f(p,u) \simeq f(q,v), p\sigma \simeq q\sigma\}; \hat{s} \simeq \hat{t})} \; u\sigma = v\sigma$$

Cancellation 2:

$$\frac{(E \cup \{f(d_1,d_2) \simeq y\}; \hat{s} \simeq \hat{t})}{(E \cup \{f(d_1,d_2) \simeq y, d_1\sigma \simeq x\}; \hat{s} \simeq \hat{t})} \; \begin{array}{ll} y \in V(d_1) & \sigma = \{y \mapsto f(x,d_2)\} \\ y \notin V(d_2) & x \textit{ is a new variable} \end{array}$$

Cancellation 3:

$$\frac{(E \cup \{f(p_1,q_1) \simeq r_1, f(p_2,q_2) \simeq r_2\}; \hat{s} \simeq \hat{t})}{(E \cup \{f(p_1,q_1) \simeq r_1, f(p_2,q_2) \simeq r_2, p_1\sigma \simeq p_2\sigma\}; \hat{s} \simeq \hat{t})} \; \begin{array}{l} q_1\sigma = q_2\sigma \\ r_1\sigma = r_2\sigma \end{array}$$

Cancellation 4:

$$\frac{(E \cup \{f(p,u) \simeq f(q,u)\}; \hat{s} \simeq \hat{t})}{(E \cup \{p \simeq q\}; \hat{s} \simeq \hat{t})}$$

where the function f is right cancellable. In *Cancellation 2*, if the substitution $\sigma = \{y \mapsto f(x,d_2)\}$ is applied to the given equation, it becomes $f(d_1\sigma, d_2) \simeq f(x,d_2)$, since y does not occur in d_2. The cancellation law reduces this equation to $d_1\sigma \simeq x$.

Cancellation 4 is not necessary for the purpose of completeness, since the same effect can be obtained by a step of *Cancellation 1* with empty mgu followed by a step of Functional subsumption. It is added to improve the efficiency.

In order to prove that the UKB procedure with the cancellation inference rules is a completion procedure, we need to prove that the Cancellation inference rules are proof-reducing. We adopt the proof ordering $>_{UKBC}$ defined as follows: a ground equational step $s \simeq t$ justified by an equation $l \simeq r$ has complexity measure $(s, l\sigma, l, t)$, if s is $c[l\sigma]$, t is $c[r\sigma]$ and $s \succ t$. Complexity measures are compared by the lexicographic combination $>^{ec}$ of the orderings $\succ$, $\trianglerighteq$, $\trianglerighteq$ and $\succ$. Proofs are compared by the lexicographic combination $>_{uc}$ of the multiset extansions $\succ_{mul}$ and $>^{ec}{}_{mul}$: $\Pi(E_i, \hat{s}_i \simeq \hat{t}_i) >_{UKBC} \Pi(E_j, \hat{s}_j \simeq \hat{t}_j)$ if and only if $(\{\hat{s}_i, \hat{t}_i\}, \hat{s}_i \leftrightarrow^*_{E_i} \hat{t}_i) >_{uc} (\{\hat{s}_j, \hat{t}_j\}, \hat{s}_j \leftrightarrow^*_{E_j} \hat{t}_j)$. The proof of Lemma 6.1.1 is unaffected if $>_{UKBC}$ replaces $>_{UKB}$.

Lemma 6.2.1 *The* Cancellation *inference rules are proof-reducing.*

Proof: we show that if $(E_i; \hat{s}_i \simeq \hat{t}_i) \vdash_{UKB} (E_{i+1}; \hat{s}_i \simeq \hat{t}_i)$ is a Cancellation step, then if $\Pi(E_i, \hat{s} \simeq \hat{t}) \neq \Pi(E_{i+1}, \hat{s} \simeq \hat{t})$, that is the inference step affects the proof of $\hat{s} \simeq \hat{t}$, $\Pi(E_i, \hat{s} \simeq \hat{t}) >^{ec}{}_{mul} \Pi(E_{i+1}, \hat{s} \simeq \hat{t})$ holds.

- An application of the rule Cancellation 1 to an equation $f(p,u) \simeq f(q,v)$ affects any minimal proof in E_i which contains a step $s \leftrightarrow t$ such that $s = c[f(p,u)\tau]$, $t = c[f(q,v)\tau]$ and $\tau \trianglerighteq \sigma$, where $\trianglerighteq$ is the subsumption ordering and σ is the mgu such that $u\sigma = v\sigma$ of the application of Cancellation 1. The step $s \leftrightarrow_{f(p,u) \simeq f(q,v)} t$ has complexity $(s, f(p,u)\tau, f(p,u), t)$, if $s \succ t$.

In the minimal proofs in E_{i+1} the step $s \leftrightarrow_{f(p,u) \simeq f(q,v)} t$ is replaced by a step $s \leftrightarrow_{p\sigma \simeq q\sigma} t$ justified by the new equation $p\sigma \simeq q\sigma$ generated by the application of Cancellation 1. The step $s \leftrightarrow_{p\sigma \simeq q\sigma} t$ has complexity $(s, p\tau, p\sigma, t)$. Since $f(p,u)\tau \triangleright p\tau$, $(s, f(p,u)\tau, f(p,u), t) >^{ec} (s, p\tau, p\sigma, t)$ follows. A symmetric argument applies if $t \succ s$.

- An application of the rule Cancellation 2 to an equation $f(d_1, d_2) \simeq y$ affects any minimal proof in E_i which contains a step $s \leftrightarrow t$ such that $s = c[f(d_1, d_2)\tau]$, $t = c[y\tau]$ and $\tau \trianglerighteq \sigma$, where σ is $\{y \mapsto f(x, d_2)\}$. Since $y \in V(d_1)$, we have $f(d_1, d_2)\tau \succ y\tau$ by the subterm property and therefore $s \succ t$ by monotonicity, so that the step $s \leftrightarrow t$ has complexity $(s, f(d_1, d_2)\tau, f(d_1, d_2), t)$. In the minimal proofs in E_{i+1} the step $s \leftrightarrow t$ is replaced by a step $s \leftrightarrow_{d_1\sigma \simeq x} t$ justified by the new equation $d_1\sigma \simeq x$ generated by the application of Cancellation 2. The step $s \leftrightarrow_{d_1\sigma \simeq x} t$ has complexity $(s, d_1\tau, d_1\sigma, t)$. Since $f(d_1, d_2)\tau \triangleright d_1\tau$, $(s, f(d_1, d_2)\tau, f(d_1, d_2), t) >^{ec} (s, d_1\tau, d_1\sigma, t)$ follows.

- An application of the rule Cancellation 3 to two equations $f(p_1, q_1) \simeq r_1$ and $f(p_2, q_2) \simeq r_2$ affects any minimal proof in E_i which contains a subproof $s \leftrightarrow u \leftrightarrow t$ such that $s = c[f(p_1, q_1)\tau]$, $u = c[r_1\tau]$, $t = c[f(p_2, q_2)\tau]$ and $\tau \trianglerighteq \sigma$, where σ is the mgu such that $q_1\sigma = q_2\sigma$ and $r_1\sigma = r_2\sigma$ of the application of Cancellation 3. It follows that $q_1\tau = q_2\tau$ and $r_1\tau = r_2\tau$ too. The subproof $s \leftrightarrow u \leftrightarrow t$ is replaced in any minimal proof in E_{i+1} by a single step $s \leftrightarrow_{p_1\sigma \simeq p_2\sigma} t$ justified by the new equation $p_1\sigma \simeq p_2\sigma$ generated by the application of Cancellation 3.

  1. If $s \succ t \succ u$, the subproof $s \leftrightarrow u \leftrightarrow t$ has complexity $\{(s, f(p_1, q_1)\tau, f(p_1, q_1), u), (t, f(p_2, q_2)\tau, f(p_2, q_2), u)\}$ and the step $s \leftrightarrow_{p_1\sigma \simeq p_2\sigma} t$ has complexity $(s, p_1\tau, p_1\sigma, t)$. Since $f(p_1, q_1)\tau \triangleright p_1\tau$, the result follows. A symmetric argument applies if $t \succ s \succ u$.
  2. If $s \succ u \succ t$, the subproof $s \leftrightarrow u \leftrightarrow t$ has complexity $\{(s, f(p_1, q_1)\tau, f(p_1, q_1), u), (u, r_1\tau, r_1, t)\}$ and the step $s \leftrightarrow_{p_1\sigma \simeq p_2\sigma} t$ has complexity $(s, p_1\tau, p_1\sigma, t)$. Since $f(p_1, q_1)\tau \triangleright p_1\tau$, the result follows. A symmetric argument applies if $t \succ u \succ s$.
  3. If $u \succ s \succ t$, the subproof $s \leftrightarrow u \leftrightarrow t$ has complexity $\{(u, r_1\tau, r_1, s), (u, r_1\tau, r_1, t)\}$ and the step $s \leftrightarrow_{p_1\sigma \simeq p_2\sigma} t$ has complexity $(s, p_1\tau, p_1\sigma, t)$. Since $u \succ s$, the result trivially follows. A symmetric argument applies if $u \succ t \succ s$. □

The UKB or AC-UKB procedure enriched with the inference rules for cancellation is complete [49]. Most of the experimental results reported in [1, 2, 3, 5] are obtained by AC-UKB with the inference rules for cancellation.

6.3 The Inequality Ordered-Saturation strategy

The UKB procedure is complete, but it is not very efficient in general. The main source of inefficiency is the Deduction inference rule, that is the forward reasoning component of UKB. This problem has been considered first for the application of Knuth-Bendix completion to the generation of confluent systems. Computing all the critical pairs is a sufficient but not necessary condition for the limit of a derivation to be confluent. A *critical pair criterion* is a criterion to

establish that certain critical pairs are not necessary to obtain a confluent system. Such criteria are studied in [12].

We rather analyze the problem from the theorem proving point of view. In the theorem proving context criteria such as those in [12] are still useful, but certainly not satisfactory, since they are not target-oriented.

All the backward reasoning steps in an UKB derivation are Simplification steps, which are strictly proof-reducing. On the other hand, a Deduction step is guaranteed to reduce the proof of some theorem, but not necessarily the proof of the target. The UKB procedure is inefficient because it generates many critical pairs which do not help in proving the target.

Therefore, our goal is to reduce the number of critical pairs generated or equivalently to perform less forward reasoning and more backward reasoning.

For the forward reasoning part, a possible approach to the problem consists in designing search plans which generate first the critical pairs that are estimated to be likely to reduce the proof of the target. Since the effect of a critical pair on the target cannot be completely determined a priori, such a search plan is based on *heuristical criteria* that measure how useful a critical pair is expected to be with respect to the task of simplifying the goal. Some examples of these heuristics are given in [3, 4].

For the backward reasoning part, we observe that if the target $\hat{s}_i \simeq \hat{t}_i$ is fully simplified with respect to E_i, $\hat{s}_i \simeq \hat{t}_i$ is minimal in the ordering $\succ_{mul}$ among all the ground equations E-equivalent to the input target $s_0 \simeq t_0$, where $E = \bigcup_{0 \leq j \leq i} E_j$. If a Simplification-first plan is adopted, UKB maintains a minimal target. Therefore, it could seem that no improvement can be obtained on the target side. However, we shall see that this is not the case.

The notion of minimal target is relative to the assumed partially ordered set (poset) of targets. If we assume the poset of ground equations ordered by $\succ_{mul}$, $\hat{s}_i \simeq \hat{t}_i$ is minimal among the ground equations E-equivalent to the input target $s_0 \simeq t_0$. The situation changes if we assume as poset of targets the poset of disjunctions of ground equations ordered by an ordering $\succ'_{mul}$ defined as follows: $N_1 \succ'_{mul} N_2$ if $min(N_1) \succ_{mul} min(N_2)$, where N_1 and N_2 are disjunctions of ground equations and $min(N)$ is the smallest equation in N according to $\succ_{mul}$. Since the equations are ground and the simplification ordering is total on ground, there is a smallest element in a disjunction and this ordering is well defined. Furthermore, the poset of equations is embedded[1] in the poset of disjunctions.

We show why the backward reasoning part of UKB is not guaranteed to compute a minimal target if the poset of disjunctions is assumed. Let $(E_i; \hat{s}_i \simeq \hat{t}_i)$ be the current stage in an UKB derivation and $l \simeq r$ be an un-orientable equation in E_i, such that $\hat{s}_i|u = l\sigma$ for some position u and substitution σ, but $\hat{s}_i \prec \hat{s}_i[r\sigma]_u$. In other words, l matches a subterm of $\hat{s}_i$ but Simplification does not apply because $\hat{s}_i$ would not be replaced by a smaller term. However, we assume that the

[1]Given two posets $\mathcal{P}_1 = (D_1, >_1)$ and $\mathcal{P}_2 = (D_2, >_2)$, an *embedding* $h: \mathcal{P}_1 \to \mathcal{P}_2$ is an injective function $h: D_1 \to D_2$ which preserves the ordering: for all $x, y \in D_1$, $x >_1 y$ implies $h(x) >_2 h(y)$. The function which maps a ground equation into the disjunction given by the ground equation itself is clearly an embedding of the poset of ground equations into the poset of disjunctions of ground equations, since the smallest element in a disjunction given by a single equation is the equation itself.

target $\hat{s}_i[r\sigma]_u \simeq \hat{t}_i$ is generated nonetheless and that by simplification it reduces to an equation which is smaller than $\hat{s}_i \simeq \hat{t}_i$, that is $\hat{s}_i[r\sigma]_u \to^*_{E_i} \hat{s}'$, $\hat{t}_i \to^*_{E_i} \hat{t}'$ and $\{\hat{s}', \hat{t}'\} \prec_{mul} \{\hat{s}_i, \hat{t}_i\}$. If these conditions hold, we have that the disjunction $\hat{s}_i \simeq \hat{t}_i \vee \hat{s}' \simeq \hat{t}'$ is smaller than the disjunction given by $\hat{s}_i \simeq \hat{t}_i$ alone in the poset of disjunctions defined above. Therefore, if we assume the poset of disjunctions as poset of targets, it is not true that UKB maintains a minimal target.

The intuition behind the choice of considering disjunctions of equations rather than equations is that if we consider more than one target equation, we have a greater chance to find a short proof. In order to work on disjunctions of equations, we need to add to the UKB procedure an expansion inference rule, so that the target is eventually expanded into a disjunction of ground equations. Such an expansion inference rule must satisfy the relevance requirement, so that proving the validity of any of the equations in the disjunction is equivalent to prove the input target $s_0 \simeq t_0$. Also, the application of such rule must be restricted, in order to avoid the generation of a high number of target equations, which may slow down the search for a solution.

This new inference rule is superposition of an un-orientable equation onto a target equation $\hat{s} \simeq \hat{t}$ to generate a new target equation. A newly generated target equation is first simplified as much as possible and then it is kept only if it is smaller than $\hat{s} \simeq \hat{t}$:

Ordered saturation:

$$\frac{(E \cup \{l \simeq r\}; N \cup \{\hat{s} \simeq \hat{t}\})}{(E \cup \{l \simeq r\}; N \cup \{\hat{s} \simeq \hat{t}, \hat{s}' \simeq \hat{t}'\})} \quad \hat{s}|u = l\sigma \quad \hat{s}[r\sigma]_u \to^*_E \hat{s}' \quad \hat{t} \to^*_E \hat{t}' \quad \{\hat{s}', \hat{t}'\} \prec_{mul} \{\hat{s}, \hat{t}\}$$

Ordered saturation applies if $\hat{s} \prec \hat{s}[r\sigma]_u$, since if $\hat{s} \succ \hat{s}[r\sigma]_u$ holds, simplification would apply. The target equation $\hat{s}' \simeq \hat{t}'$ might have a shorter proof than the other target equations. We do not know which one has the shortest proof. We keep all of them to broaden our chance of reaching the proof as soon as possible.

In addition, we need to modify the *Deletion* inference rule, since the computation halts successfully as soon as an equation in the disjunction is reduced to a trivial equation:

Deletion:

$$\frac{(E; N \cup \{\hat{s} \simeq \hat{s}\})}{(E; true)}$$

The procedure obtained by adding Ordered saturation to UKB and by modifying Deletion as above, is called the *Inequality Ordered-Saturation strategy* (IOS) [3]. A derivation by the IOS strategy has the form

$$(E_0; N_0) \vdash_{IOS} (E_1; N_1) \vdash_{IOS} \ldots \vdash_{IOS} (E_i; N_i) \vdash_{IOS} \ldots$$

where the set N_0 contains the initial goal $\hat{s}_0 \simeq \hat{t}_0$ and at stage i, N_i is the current set of target equations. The derivation halts at stage k if N_k contains a target $\hat{s}_i \simeq \hat{t}_i$ such that $\hat{s}_i$ and $\hat{t}_i$ are identical and the clause in N_k reduces to $true$.

In order to show that the IOS strategy is a completion procedure, we assume a proof ordering $>_{IOS}$ defined as follows: $\Pi(E; N) >_{IOS} \Pi(E'; N')$ if and only if $\Pi(E; min(N)) >_{UKB} \Pi(E'; min(N'))$. In other words the proof of a disjunction is represented by the proof of the

smallest target in the disjunction.

Lemma 6.3.1 *The* Ordered saturation *inference rule is proof-reducing.*

Proof: we show that if $(E_i; N_i) \vdash_{IOS} (E_i; N_{i+1})$ is an Ordered saturation step, then $\Pi(E_i, N_i) \geq_{IOS} \Pi(E_i, N_{i+1})$. Since $N_i \subset N_{i+1}$, $min(N_i) \succeq_{mul} min(N_{i+1})$ and the result follows. □

Theorem 6.3.1 *The Inequality Ordered-Saturation strategy is a completion procedure.*

Proof: it follows from Theorem 6.1.1 and Lemma 6.3.1. □

The IOS strategy has been implemented and observed to perform better than the UKB procedure [3]. In practice, few target equations are kept, so that the overhead of handling them is negligible with respect to the advantage of keeping more than one target.

6.4 The S-strategy

The *S-strategy* [48] is an extension of the UKB procedure to the logic of equality and inequality. A presentation is a set of equations E_0 and a theorem φ is a sentence $\bar{Q}\bar{x}\ s_0 \simeq t_0 \vee \ldots \vee s_n \simeq t_n$, where $\bar{Q}\bar{x}$ is any sequence of quantifier-variable pairs. A theorem φ in this form is transformed into a target $N_0 = s_0 \simeq t_0 \vee \ldots \vee s_n \simeq t_n$, where all variables are implicitly existentially quantified, by replacing all the universally quantified variables by constants and by dropping the quantifiers. If φ is $\forall \bar{x} s_0 \simeq t_0$, N_0 is $\hat{s}_0 \simeq \hat{t}_0$ and the S-strategy reduces to the UKB procedure. A computation has the form

$$(E_0; N_0) \vdash_S (E_1; N_1) \vdash_S \ldots \vdash_S (E_i; N_i) \vdash_S \ldots$$

where $\forall i \geq 0$, E_i is a set of equations and N_i is a disjunction of target equations with existentially quantified variables. A derivation halts at stage k if N_k contains a target $s_i \simeq t_i$ whose sides are unifiable. The set of inference rules of UKB is modified as follows:

- *Presentation inference rules*:
  - *Simplification*:
    $$\frac{(E \cup \{p \simeq q, l \simeq r\}; N)}{(E \cup \{p[r\sigma]_u \simeq q, l \simeq r\}; N)} \begin{array}{l} p|u = l\sigma \\ p \triangleright l \vee q \succ p[r\sigma]_u \end{array} \qquad p \succ p[r\sigma]_u$$
  - *Deduction*:
    $$\frac{(E \cup \{p \simeq q, l \simeq r\}; N)}{(E \cup \{p \simeq q, l \simeq r, p[r]_u\sigma \simeq q\sigma\}; N)} \begin{array}{l} p|u \notin X \\ p\sigma \not\preceq q\sigma, p[r]_u\sigma \end{array} \qquad (p|u)\sigma = l\sigma$$
  - *Deletion*:
    $$\frac{(E \cup \{l \simeq l\}; N)}{(E; N)}$$
  - *Functional subsumption*:
    $$\frac{(E \cup \{p \simeq q, l \simeq r\}; N)}{(E \cup \{l \simeq r\}; N)}\ (p \simeq q) \triangleright (l \simeq r)$$

- *Target inference rules*:
  - *Simplification*:
    $$\frac{(E \cup \{l \simeq r\}; N \cup \{s \simeq t\})}{(E \cup \{l \simeq r\}; N \cup \{s[r\sigma]_u \simeq t\})} \quad \begin{array}{l} s|u = l\sigma \\ s \succ s[r\sigma]_u \end{array}$$
  - *Deduction*:
    $$\frac{(E \cup \{l \simeq r\}; N \cup \{s \simeq t\})}{(E \cup \{l \simeq r\}; N \cup \{s \simeq t, s[r]_u\sigma \simeq t\sigma\})} \quad \begin{array}{l} s|u \notin X \quad (s|u)\sigma = l\sigma \\ s\sigma \not\preceq s[r]_u\sigma \end{array}$$
  - *Deletion*:
    $$\frac{(E; N \cup \{s \simeq t\})}{(E; true)} \; s\sigma = t\sigma$$

The *Deduction* inference rule applies to both the presentation and the target. The *Deletion* rule for the target is modified because the target contains variables: a contradiction is detected when the two sides of a target equation unify.

In order to characterize the S-strategy as a completion procedure, we define the following ordering: $\Pi(E_i, N_i) >_S \Pi(E_{i+1}, N_{i+1})$ if and only if $\Pi(\hat{E}_i, \hat{s}_i \simeq \hat{t}_i) >_{UKB} \Pi(\hat{E}_{i+1}, \hat{s}_{i+1} \simeq \hat{t}_{i+1})$, where $\forall i \geq 0$, $\hat{E}_i \cup \{\hat{s}_i \neq \hat{t}_i\}$ is the smallest finite unsatisfiable set of ground instances of clauses in $E_i \cup \neg N_i$. We show that this ordering is well defined. First we show how the pair $(\hat{E}_i; \hat{s}_i \simeq \hat{t}_i)$ is defined. N_i is a theorem of E_i if and only if $E_i \cup \neg N_i$ is unsatisfiable, where N_i is a disjunction of equations $s_0 \simeq t_0 \vee \ldots \vee s_n \simeq t_n$ with existentially quantified variables and therefore $\neg N_i$ is a conjunction of inequations $s_0 \neq t_0 \wedge \ldots \wedge s_n \neq t_n$ with universally quantified variables. By the Herbrand Theorem [24], the set $E_i \cup \neg N_i$ is unsatisfiable if and only if there is a finite subset of ground instances of the clauses in $E_i \cup \neg N_i$ which is unsatisfiable. Since $\neg N_i$ is a set of inequations with universally quantified variables, an unsatisfiable ground instance of $E_i \cup \neg N_i$ needs to contain just one ground inequation: $\hat{E}_i \cup \{\hat{s}_i \neq \hat{t}_i\}$ is the smallest such set with respect to the ordering $\succ_{mul}$. Since $\succ$ is total on ground terms, there exists a smallest set.

The above definition of the ordering $>_S$ says that the complexity of the proof $\Pi(E_i, N_i)$ is measured by the complexity of the ground proof $\Pi(\hat{E}_i, \hat{s}_i \simeq \hat{t}_i)$ and that the impact of the inference steps on $\Pi(E_i, N_i)$ is measured by the impact of the inference steps on $\Pi(\hat{E}_i, \hat{s}_i \simeq \hat{t}_i)$. This approach is correct if to every inference step on $(E_i; N_i)$ corresponds an inference steps on $(\hat{E}_i; \hat{s}_i \simeq \hat{t}_i)$ and vice versa. In order to prove this, we need the following lemma, which rephrases for the S-strategy the *Paramodulation Lifting Lemma*. We recall that a ground substitution is E-irreducible if it does not contain any pair $\{x \mapsto t\}$ such that t can be simplified by an equation in E.

Lemma 6.4.1 (Peterson 1983) [71], (Hsiang and Rusinowitch 1987) [50] *If σ is a ground, E-irreducible substitution, then for all inference rules f of S-strategy, if $(E\sigma; s\sigma \simeq t\sigma) \vdash_f (E'; s' \simeq t')$, then $(E; s \simeq t) \vdash_f (E''; s'' \simeq t'')$, where E' and $s' \simeq t'$ are instances of E'' and $s'' \simeq t''$ respectively.*

Lemma 6.4.2 *$(E_i; N_i) \vdash_S (E_{i+1}; N_{i+1})$ if and only if $(\hat{E}_i; \hat{s}_i \simeq \hat{t}_i) \vdash_S (\hat{E}_{i+1}; \hat{s}_{i+1} \simeq \hat{t}_{i+1})$.*

Proof:

$\Rightarrow$) An inference step on $(E_i; N_i)$ is trivially an inference step on $(\hat{E}_i; \hat{s}_i \simeq \hat{t}_i)$, since an inference step on non ground clauses is trivially an inference step on all their instances.

$\Leftarrow$) Since $(\hat{E}_i; \hat{s}_i \simeq \hat{t}_i)$ is minimal, $\hat{E}_i \subseteq E_i\sigma$ and $\hat{s}_i \simeq \hat{t}_i \in N_i\sigma$ for an E_i-irreducible substitution σ. Therefore, by Lemma 6.4.1, an inference step on $(\hat{E}_i; \hat{s}_i \simeq \hat{t}_i)$ is an inference step on $(E_i; N_i)$.□

We can finally state the following theorem:

Theorem 6.4.1 *The S-strategy is a completion procedure on the domain $\mathcal{T}$ of all ground equations.*

Proof: monotonicity and relevance follow from the soundness of the inference rules [48]. By the definition of the ordering $>_S$, the inference rules of S-strategy are proof-reducing if they are proof-reducing on ground proofs with respect to the ordering $>_{UKB}$. This follows from Lemma 6.1.1 and Lemma 6.1.2, since target Deduction is just target Simplification if the target is ground. □

If a *fair* search plan is provided, the S-strategy is a semidecision procedure for theories in the logic of equality and inequality:

Theorem 6.4.2 (Hsiang and Rusinowitch 1987) [48] *A sentence $\bar{Q}\bar{x}\ s_0 \simeq t_0 \vee \ldots \vee s_n \simeq t_n$ is a theorem of an equational theory E if and only if the S-strategy derives* true *from $(E; s_0 \simeq t_0 \vee \ldots \vee s_n \simeq t_n)$.*

Chapter 7

Disproving inductive conjectures

The Knuth-Bendix completion procedure has been also applied to *disprove inductive conjectures* in equational theories. This method has been called *inductionless induction*, *proof by consistency* or *proof by the lack of inconsistency* by several authors [68, 40, 53, 66, 54, 38, 60, 61, 13, 56]. Extensions of this method to Horn logic with equality are explored in [63].

7.1 Semidecision procedures to disprove inductive conjectures

A clause φ is an *inductive theorem* of S, written $S \models_{Ind} \varphi$, if and only if for all ground substitutions σ, $\varphi\sigma \in Th(S)$. We denote by $Ind(S)$ the set of all the inductive theorems of S, $Ind(S) = \{\varphi |\ S \models_{Ind} \varphi\}$, by $GTh(S)$ the set of all the ground theorems of S, $GTh(S) = \{\varphi |\ \varphi \in Th(S), \varphi\ ground\}$ and by $G(\varphi)$ the set of all the ground instances of φ, $G(\varphi) = \{\varphi\sigma |\ \varphi\sigma\ ground\}$.

A clause φ is not an inductive theorem of S if exists a ground substitution σ such that $\varphi\sigma \notin GTh(S)$. If $GTh(S)$ is decidable, the complement of $Ind(S)$ is semidecidable. On the other hand $Ind(S)$ may not be semidecidable even if $GTh(S)$ is decidable. Since if $Ind(S)$ were semidecidable, both $Ind(S)$ and its complement would be semidecidable, which means the problem is decidable.

If $\varphi \notin Ind(S)$, then $GTh(S \cup \{\varphi\}) \neq GTh(S)$, since $\varphi\sigma \in GTh(S \cup \{\varphi\})$ for all ground instances $\varphi\sigma$. Thus, we can prove that φ is not an inductive theorem of S by proving the following target:

$\Phi_0 = \exists\sigma\ \exists\psi \in S \cup \{\varphi\}$ *such that* σ *is ground and* $\psi\sigma \in GTh(S \cup \{\varphi\}) - GTh(S)$.

If there exists an oracle $\mathcal{O}$ to decide such a target, a completion procedure $\mathcal{C} =< I_p, I_t; \Sigma; \mathcal{O} >$ equipped with the oracle $\mathcal{O}$ will be a *semidecision procedure for disproving inductive conjectures*. A derivation has the form

$(S \cup \{\varphi\}; \Phi_0) \vdash_{\mathcal{C}} (S_1; \Phi_1) \vdash_{\mathcal{C}} \ldots (S_i; \Phi_i) \vdash_{\mathcal{C}} \ldots$.

At each step the target is

$\Phi_i = \exists\sigma\ \exists\psi \in S_i$ *such that* σ *is ground and* $\psi\sigma \in GTh(S_i) - GTh(S)$,

where σ *ground* means a substitutions mapping variables to ground terms on the signature of S. No inference step applies to the target: the procedure takes as input the presentation $S \cup \{\varphi\}$ given

by the original presentation and the inductive conjecture and it proceeds by applying inference rules to the presentation until it obtains a presentation S_k such that the oracle applied to S_k answers positively and replaces Φ_k by *true*.

In the equational case, Φ_i is

$\Phi_i = \exists\sigma\ \exists s_i \simeq t_i \in E_i$ *such that* σ *is ground and* $(s_i \simeq t_i)\sigma \in GTh(E_i) - GTh(E)$.

An oracle to decide Φ_i is available only under the assumption that the input set of equations E is ground confluent. Under this hypothesis, Φ_i is true if and only if there are two ground E-irreducible terms s and t such that $s_i\sigma \to_E^* s$, $t_i\sigma \to_E^* t$ and $s \simeq t \in GTh(E_i)$. Therefore, we can restrict our attention to ground E-irreducible terms.

A first oracle was given in [53] for equational presentations satisfying the *principle of definition*:

Definition 7.1.1 (Huet and Hullot 1982) [53] *A presentation E of an equational theory satisfies the* principle of definition *if*

- *the signature F of E is given by the disjoint union $F = C \uplus D$, of two sets of function symbols, the set C of* constructors *and the set D of* defined symbols,
- *the set $T(C)$ of all ground constructor terms is* free, *that is there are no two terms t_1 and t_2 in $T(C)$ such that $t_1 \leftrightarrow_E^* t_2$ and*
- *all function symbols in D are* completely defined *on C, that is for all ground term $t \in T(F)$, there exists a unique ground constructor term $t' \in T(C)$ such that $t \leftrightarrow_E^* t'$.*

If a presentation E satisfies the principle of definition, the ground E-irreducible terms are the ground terms made only of constructor symbols. Therefore, Φ_i is true if and only if there are two ground constructor terms t_1 and t_2 such that $t_1 \leftrightarrow_{E_i}^* t_2$. The following inference rules implement this test [53]:

- *Disproof 1*:
  $\dfrac{(E \cup \{f(t_1 \dots t_n) \simeq g(s_1 \dots s_n)\}; \Phi)}{(E \cup \{f(t_1 \dots t_n) \simeq g(s_1 \dots s_n)\}; true)}\ f, g \in C,\ f \neq g$
- *Disproof 2*:
  $\dfrac{(E \cup \{f(t_1 \dots t_n) \simeq x\}; \Phi)}{(E \cup \{f(t_1 \dots t_n) \simeq x\}; true)}\ f \in C$
- *Decompose*:
  $\dfrac{(E \cup \{f(t_1 \dots t_n) \simeq f(s_1 \dots s_n)\}; \Phi)}{(E \cup \{t_1 \simeq s_1 \dots t_n \simeq s_n\}; \Phi')}\ f \in C$

where Φ' is Φ_i for $E_i = E \cup \{t_1 \simeq s_1 \dots t_n \simeq s_n\}$. The two *Disproof* inference rules detect that equalities between constructor terms have been derived. By the principle of definition, the theory of E_0 does not include such equalities. They are a consequence of adding the inductive conjecture $s \simeq t$, which is therefore disproved. The *Decompose* rule is added for the purpose of efficiency. It replaces an equation $f(t_1 \dots t_n) \simeq f(s_1 \dots s_n)$, where f is a constructor, by the equations $t_1 \simeq s_1 \dots t_n \simeq s_n$: since f is a constructor, by the principle of definition two terms $f(t_1 \dots t_n)$ and $f(s_1 \dots s_n)$ may be equal only if their arguments are equal.

Theorem 7.1.1 (Huet and Hullot 1982) [53], (Bachmair 1988) [13] *If E is a ground confluent equational presentation, satisfying the principle of definition, the Unfailing Knuth-Bendix completion procedure enriched with the inference rules* Decompose, Disproof 1 *and* Disproof 2 *is a semidecision procedure for the complement of $Ind(E)$.*

This result was obtained assuming a uniformly fair search plan on the domain of all ground equations.

A more general oracle has been proposed in [54] for the Knuth-Bendix completion procedure and extended to the UKB procedure in [13]. This test is based on *ground reducibility*: a term t is *ground E-reducible* if for all ground substitutions σ, $t\sigma$ is E-reducible. Ground E-reducibility is decidable only if E is a ground confluent rewrite system [72]. Therefore, the test in [54, 13] applies only if the input presentation E is ground confluent and all its equations can be oriented into rewrite rules.

We assume that E has these properties and we call it R. An equation $l \simeq r$ is *ground R-reducible* if for all ground substitutions σ, such that $l\sigma$ and $r\sigma$ are distinct, either $l\sigma$ or $r\sigma$ is R-reducible. If an equation $l \simeq r$ which is not ground R-reducible is derived from $R \cup \{s \simeq t\}$ at stage i, there is a ground instance $l\sigma \simeq r\sigma$ of the equation such that $l\sigma$ and $r\sigma$ are distinct and R-irreducible, but $l\sigma \simeq r\sigma \in GTh(E_i)$. This means that Φ_i is true and the inductive conjecture is diproved. The following inference rule implements this test:

Disproof 3:

$$\frac{(E \cup \{l \simeq r\}; \Phi)}{(E \cup \{l \simeq r\}; true)}\ l \simeq r \text{ is not ground } R - \text{reducible}$$

Theorem 7.1.2 (Jouannaud and Kounalis 1986) [54], (Bachmair 1988) [13] *If R is a ground confluent rewrite system, the Unfailing Knuth-Bendix completion procedure enriched with the inference rule* Disproof 3 *is a semidecision procedure for the complement of $Ind(R)$.*

Like in the previous result, a uniformly fair search plan on the domain of all ground equations was assumed.

The ground reducibility test is not a practical solution to the problem of inductive theorem proving, because its complexity is very high. Furthermore, the two above results about the UKB procedure for disproving inductive conjectures have been obtained in a context where completion was considered a generator of confluent systems and the capability of disproving inductive conjectures was regarded as a side effect. This explains why both results were obtained under the hypothesis of a uniformly fair search plan.

On the other hand, we have shown that disproving an inductive conjecture is a semidecision process of a specific target. Therefore, only the proof of the target needs to be reduced. We define the proof of the target Φ_i as follows:

$\Pi(S_i, \Phi_i) = \Pi(S_i, min\{\psi\sigma |\ \psi \in S_i, \psi\sigma \in GTh(S_i) - GTh(S)\})$,

that is the proof of the target is the proof of the smallest ground instance of some clause in S_i which is a theorem in S_i but not in S.

In the equational case, a completion procedure which eventually generates a ground confluent set of equations, is able to reduce the proofs of all ground theorems and therefore the proof of the target. However, this is not necessary. Since the proof of the target is the proof of the smallest ground theorem which is not a theorem of the original presentation, we can restrict our attention to a smaller set of ground theorems:

Definition 7.1.2 (Fribourg 1986) [38] *Given a ground confluent presentation E, a set of substitutions H is* E-inductively complete *if for all ground substitutions ρ, there exist a substitution $\sigma \in H$ and a ground substitution τ such that $\rho \rightarrow_E^* \sigma\tau$.*

For instance, if E includes the axioms $0+x \simeq x$ and $succ(x)+y \simeq succ(x+y)$, a set of substitutions H is E-inductively complete if it contains two substitutions σ_1 and σ_2 such that $\{x \mapsto 0\} \in \sigma_1$ and $\{x \mapsto succ(y)\} \in \sigma_2$. Indeed, all ground terms reduce either to 0 or to some term $succ^n(0)$, so that a set of substitutions which covers the instances $x \mapsto 0$ and $x \mapsto succ(y)$ cover all instances.

Clearly, we are interested in minimal E-inductively complete sets of substitutions. All such sets are equivalent for our purposes, since they all have the property of covering all the ground substitutions. We denote by H_E one such set and by $\mathcal{IT}_E$ the domain of all the ground equations which are instances of substitutions in H_E, that is $\mathcal{IT}_E = \{(l \simeq r)\sigma\tau \mid \sigma \in H_E, (l \simeq r)\sigma\tau \textit{ is ground}\}$. The proof of the target is the proof of the smallest ground theorem which is not a theorem of the original presentation. This smallest ground theorem is in $\mathcal{IT}_E$ and therefore reducing the proofs of the theorems in $\mathcal{IT}_E$ is sufficient to guarantee that the proof of the target is reduced, as was first proved in [38] for the application of Knuth-Bendix completion to disprove inductive conjectures in equational theories:

Theorem 7.1.3 (Fribourg 1986) [38] *A completion procedure $\mathcal{C} =< I_p, I_t; \Sigma; \mathcal{O} >$ on the domain $\mathcal{IT}_E$, with complete inference rules and fair search plan is a semidecision procedure for the complement of $Ind(E)$ for all equational presentations E, for which the oracle $\mathcal{O}$ is computable.*

As a consequence, the Deduction inference rule of UKB can be restricted in such a way that at stage i of the derivation, superpositions on $p \simeq q$ at position u are performed only if the set of mgus $\{\sigma | l\sigma = (p|u)\sigma, l \simeq r \in E_i\}$ is E-inductively complete. The position u is called *completely superposable* in [38]:

Deduction on completely superposable positions:

$$\frac{(E \cup \{p \simeq q, l \simeq r\}; \Phi)}{(E \cup \{p \simeq q, l \simeq r, p[r]_u\sigma \simeq q\sigma\}; \Phi')} \quad \begin{array}{l} p|u \notin X,\ (p|u)\sigma = l\sigma \\ u \textit{ is completely superposable} \end{array} \qquad p\sigma \not\preceq q\sigma, p[r]_u\sigma$$

where Φ' is Φ_i for $E_i = E \cup \{p \simeq q, l \simeq r, p[r]_u\sigma \simeq q\sigma\}$. Furthermore, for all equations $p \simeq q$, generated during the derivation, it is sufficient to perform superpositions on just one completely superposable position in $p \simeq q$. In other words, a search plan which selects just one completely superposable position in every equation is fair. We can summarize these results as follows:

Theorem 7.1.4 (Fribourg 1986) [38] *If the Deduction inference rule is restricted to completely superposable positions and the search plan selects one completely superposable position in every generated equation, Theorem 7.1.1 and Theorem 7.1.2 still hold.*

This result requires an algorithm to detect the completely superposable positions. An equivalent characterization is the following: a position u in p is completely superposable if for all ground instances $(p|u)\rho$ there is an equation $l \simeq r$ in E such that $(p|u)\rho = l\sigma$ and $l\sigma \succ r\sigma$. The problem of detecting completely superposable positions reduces to the ground reducibility problem. However, if the presentation satisfies the principle of definition, a position u is completely superposable if $p|u$ is a term which has a defined symbol at the root and only constructor symbols and variables at the positions below the root. Therefore, the above theorem can be applied in practice to presentations satisfying the principle of definition.

Chapter 8

Logic programming

In this chapter we present a completion procedure for logic programming, called *Linear Completion*. Logic programs interpreted by Linear Completion are rewrite systems or *rewrite programs*. The main difference between rewrite programs and Prolog programs is that rewrite programs differentiate between predicates which are *mutually exclusively defined* and those which are not. A predicate is mutually exclusively defined if the head of each of its clauses is logically equivalent to its body. A typical such example is the usual definition of *append*. In rewrite programs, mutually exclusively defined predicates are defined by logical equivalences, whereas predicates which are not mutually exclusively defined are defined by implications as in Prolog. We give a few examples showing how rewrite programs may turn out to be more accurate than Prolog programs in capturing the user's intended semantics, because of this additional feature.

Our procedure Linear Completion differs from the previous one [29, 30], because we allow *simplification* of goals by their ancestors. The basic interpreter for pure Prolog does not have contraction inference rules, since resolution is an expansion inference rule. We are therefore interested in studying the effect of a contraction inference rule in the interpretation of logic programs.

Simplification reduces the search space and the amount of backtracking performed by the interpreter. It prevents certain infinite loops which are otherwise unavoidable in pure Prolog. If simplification is allowed, programs given by equivalences and programs given by implications show a different behaviour. Namely, rewrite programs where predicates are defined by equivalences may give fewer answers than the Prolog programs defining the same predicates by implications, because some answers are pruned by simplification. This happens in spite of the fact that rewrite programs are *denotationally equivalent* to Prolog programs at the ground level.

We give a fixpoint characterization of rewrite programs and we show that the proof theoretic semantics given by Linear Completion, the model theoretic semantics and the fixpoint semantics are all equivalent. Rewrite programs and Prolog programs for the same predicates have the same fixpoint, i.e. the same set of ground true facts. Rewrite programs may give fewer answers because if predicates are defined by equivalences, distinct Prolog answers turn out to be equivalent with respect to the rewrite program and "collapse" into one. However, rewrite programs are also guaranteed not to lose any necessary answers: for every Prolog answer there is an equivalent

answer given by the rewrite program. This explains why a smaller set of answers covers the same set of ground true facts.

We conclude with a comparison between the pruning effects of simplification and those of subsumption based loop checking mechanisms for Prolog presented in [7, 17].

8.1 Rewrite programs

Term rewriting systems have been widely applied in functional programming [23, 39, 41, 43] and to a less extent in logic programming [29, 30, 31, 73]. In case of functional programs the evaluation mechanism is reduction and a computation consists in reducing a ground input term to an irreducible form, which represents the output. To achieve logic programming the evaluation mechanism is extended to reduction and linear superposition. This amounts to a strongly restricted form of Knuth-Bendix completion, termed *Linear Completion* [30]. A computation consists in generating an answer substitution for a non ground query as it is done in Prolog.

Despite various approaches suggested, there is a common misconception that rewrite programs have the same semantics as Prolog except for a different evaluation mechanism. Surprisingly enough, this is not true in general. We present a more precise operational and denotational semantics and show how rewrite programs can avoid certain infinite loops which occur in similar Prolog programs.

The main reason for the different behaviour of rewrite programs is the utilization of an inference rule for simplification. We demonstrate its use with a simple example. The following Prolog program:

$append([\,], L, L).$

$append([X|L1], Y, [X|L2]) :- append(L1, Y, L2).$

with the query

$?- append(X, [b|Y], [a, b, c|Z]).$

generates an infinite set of solutions as shown in Fig. 8.1.

The rewrite program, on the other hand, is defined as

$append([\,], L, L) \rightarrow true$

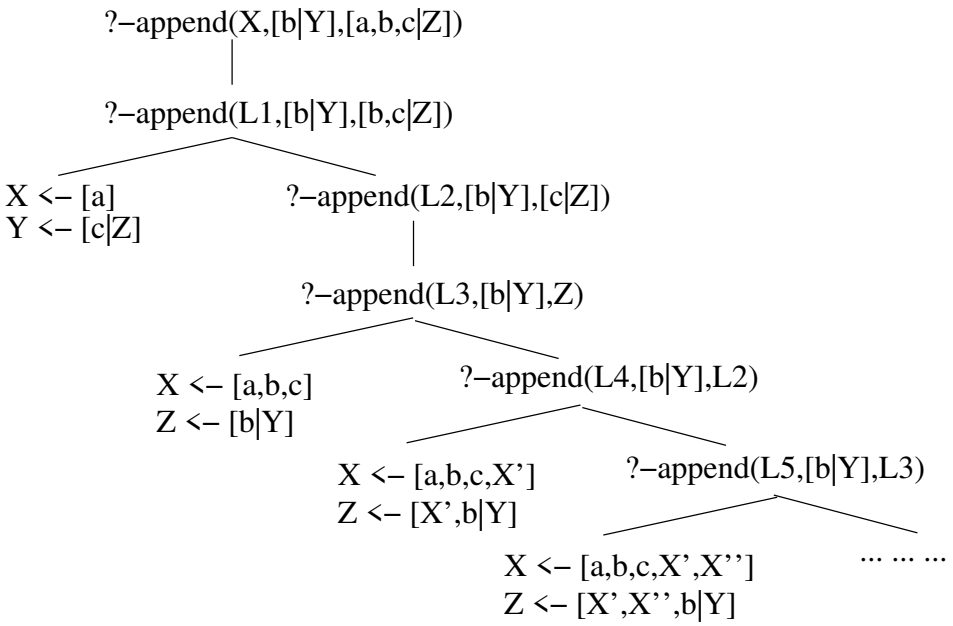
$append([X|L1], Y, [X|L2]) \rightarrow append(L1, Y, L2)$

and the query is

$append(X, [b|Y], [a, b, c|Z]) \rightarrow answer(X, Y, Z).$

If we execute this program by Linear Completion we get only the first two answers as shown in Fig. 8.2.

The last step, labeled by $\downarrow$, is a simplification step where an ancestor goal, labeled (G1) in Fig. 8.2, rewrites the current goal (G2) to a trivial goal in the form $answer(\,) \leftrightarrow answer(\,)$. Since



$\{X \mapsto [a],\ Y \mapsto [c|Z]\}$
$\{X \mapsto [a, b, c],\ Z \mapsto [b|Y]\}$
$\{X \mapsto [a, b, c, X'],\ Z \mapsto [X', b|Y]\}$
$\{X \mapsto [a, b, c, X', X''],\ Z \mapsto [X', X'', b|Y]\}$
. . . . . .
$\{X \mapsto [a, b, c, X', X'', \ldots, X^n],\ Z \mapsto [X', X'', \ldots, X^n, b|Y]\}$
. . . . . .

Figure 8.1: Prolog generates an infinite set of solutions

no inference can be applied to this goal, the execution halts with just two answers.

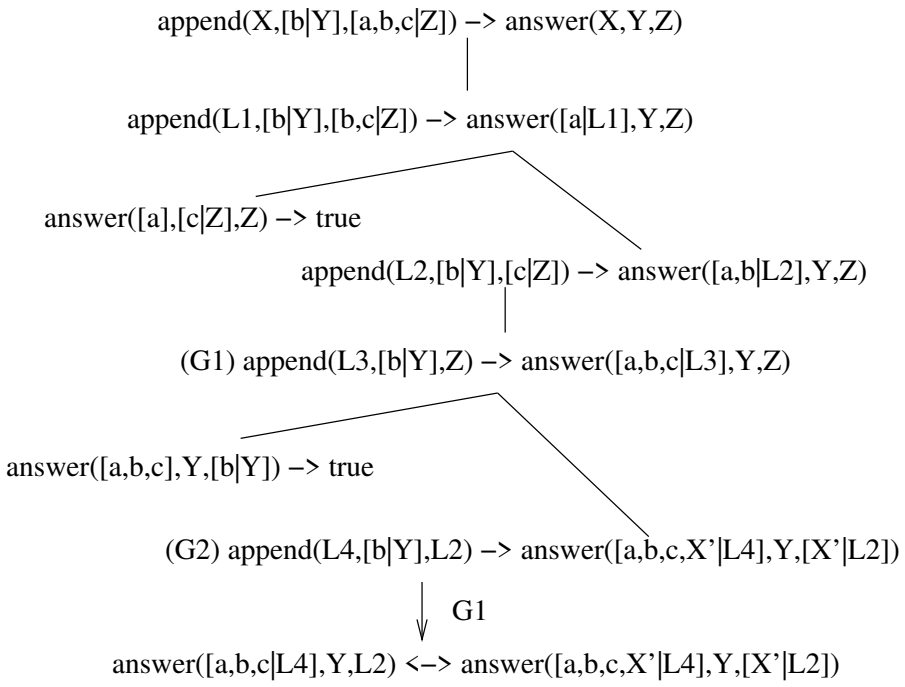
The reason for this different behaviour is that the Prolog program and the rewrite program give two different definitions of *append*. The "program units" in a rewrite program and in a Prolog program are interpreted in two different ways. In Prolog, each unit is a clause, where "$:-$" indicates the logical *if*. In rewrite programs, the logical connective "$\rightarrow$" means *if and only if*. The infinitely many answers in the form

$$\{X \mapsto [a, b, c, X', X'', \ldots, X^n],\ Z \mapsto [X', X'', \ldots, X^n, b|Y]\}$$

given by Prolog are not equivalent if *append* is defined by implications, but they all collapse to the second solution

$$\{X \mapsto [a, b, c],\ Z \mapsto [b|Y]\}$$

if *append* is defined by bi-implications. The first answer of the rewrite program corresponds to the first answer of the Prolog program. The second answer of the rewrite program corresponds to the second answer of the Prolog program and all the proceeding ones.



$\{X \mapsto [a],\ Y \mapsto [c|Z]\}$
$\{X \mapsto [a,b,c],\ Z \mapsto [b|Y]\}$

Figure 8.2: Linear completion generates only two answers

We can see why this happens by instantiating the query by the answer substitutions. If the query is instantiated by the first answer substitution, it is rewritten to $append([\,], [b, c|Z], [b, c|Z])$ and then to $true$. If it is instantiated by the second answer or any of the proceeding ones, it is rewritten to $append([\,], [b|Y], [b|Y])$ and then to $true$. All the answers but the first one yield the same true fact if simplification is applied, i.e. if *append* is defined by equivalences. All those answers are equivalent to the second one with respect to the rewrite program so that they are not generated.

Since the intended definition of *append* is actually

$append([X|L1], Y, [X|L2])\ if\ and\ only\ if\ append(L1, Y, L2),$

the rewrite program seems closer to the intended meaning of the program than the Prolog program.

The interpretation of program units as logical equivalences may also help resolving some loops which may occur in Prolog. For example, consider the following Prolog program:

$\ldots$

$P(X, Y, Z) :- append(X, [b|Y], [a, b, c|Z]), non-member(a, X).$

$\ldots$

$append([\,], L, L).$

$append([X|L1], Y, [X|L2]) :- append(L1, Y, L2).$

...

with the query $?- P(X, Y, Z)$.

Prolog falls into an infinite loop when evaluating the first clause for P, since a is a member of X in all the solutions of $append(X, [b|Y], [a, b, c|Z])$. Such potential loops cannot even be prevented beforehand by using *cut*. The rewrite program does not loop and evaluates the second clause of P, since there are only two answers from evaluating the *append* subgoal and none of them satisfies the $non-member$ subgoal.

One may suspect that simplification may throw away too many answers and change the intended semantics. For instance, consider the following:

$Q(X, Y, Z) :- append(X, [b|Y], [a, b, c|Z]), size(X) > 3.$

with the query $?- Q(X, Y, Z)$.

Since in the two answers for the *append* subgoal generated in the previous examples the size of X is less than or equal to three, it seems that no solution would be provided. This is not the case. When $Q(X, Y, Z) \rightarrow answer(X, Y, Z)$ is given as the query, the execution generates first the two solutions to the *append* subgoal

$size([a]) > 3 \rightarrow answer([a], [c|Z], Z)$

$size([a, b, c]) > 3 \rightarrow answer([a, b, c], Y, [b|Y]),$

both of which fail to give any solution to the Q problem. Then the execution continues with the goal

$append(L1, [b|Y], L2), size([a, b, c, X'|L1]) > 3 \rightarrow answer([a, b, c, X'|L1], Y, [X'|L2]).$

Assuming that *size* is defined as desired, $size([a, b, c, X'|L1]) > 3$ simplifies to *true* and the goal is reduced to

$append(L1, [b|Y], L2) \rightarrow answer([a, b, c, X'|L1], Y, [X'|L2])$ (G2).

Note that this is the same goal (G2) generated in the previous execution for the *append* query. In that execution this goal is rewritten by its ancestor (G1) and it does not yield any answer. In the execution for the Q query all ancestors contain a *size* literal too, so that they cannot be applied to simplify the goal (G2), which yields a correct solution

$answer([a, b, c, X'], Y, [X', b|Y]) \rightarrow true.$

Then the computation halts as the new goal

$append(L1, [b|Y], L3) \rightarrow answer([a, b, c, X', X''|L1], Y, [X', X''|L3])$

is reduced by its ancestor (G2) to

$answer([a,b,c,X'|L1],Y,[X'|L3]) \leftrightarrow answer([a,b,c,X',X''|L1],Y,[X',X''|L3])$ [1].

So far we have seen examples where it is desirable to define predicates by equivalences. However, not all relations are meant to be defined by equivalences. Still, rewrite programs allow us to define predicates by implications. For example, for the *ancestor* relation

$ancestor(X,Y) :- parent(X,Y).$

$ancestor(X,Y) :- parent(Z,Y),\ ancestor(X,Z).$

both clauses are implications. As it was already pointed out in [30], implications can be written as bi-implications by recalling that $P :- Q$ is equivalent to $P \wedge Q \rightarrow Q$. If we add some facts and a query we get:

$parent(jb,lc) \rightarrow true$

$parent(jb,gg) \rightarrow true$

$parent(gg,wm) \rightarrow true$

$ancestor(X,Y),\ parent(X,Y) \rightarrow parent(X,Y)$

$ancestor(X,Y),\ parent(Z,Y),\ ancestor(X,Z) \rightarrow parent(Z,Y),\ ancestor(X,Z)$

$ancestor(jb,Z) \rightarrow answer(Z).$

This program gives the same answers as Prolog, but the computation is optimized by simplification of goals as shown in Fig. 8.3.

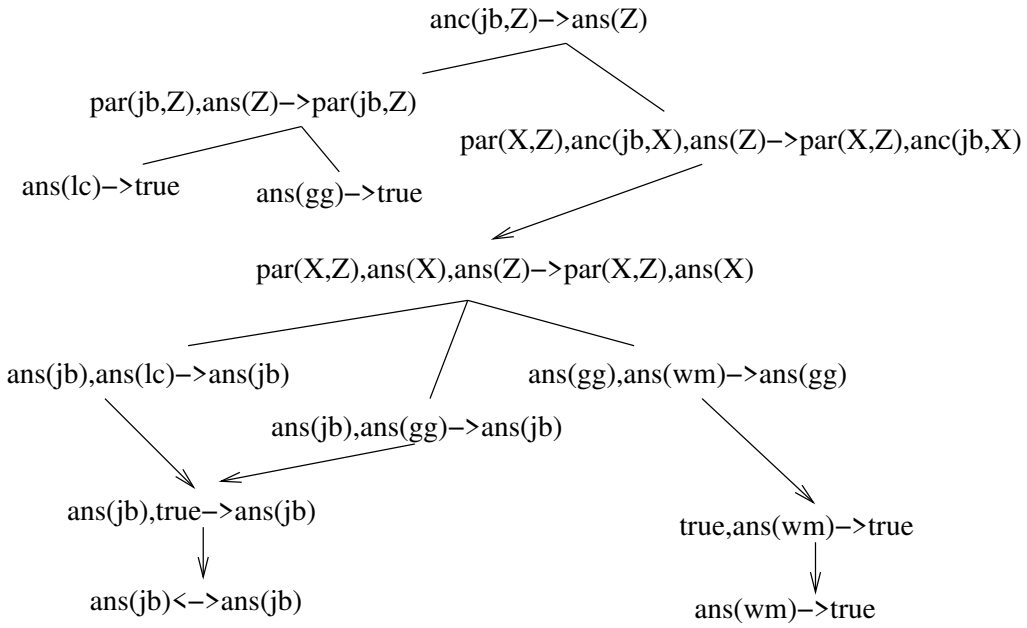
The generation of the first two answers is the same as in Prolog. The third answer is different. Having the goal $parent(X,Z), ancestor(jb,X)$, Prolog first generates $ancestor(jb,jb)$ twice, fails twice, then generates the goal $ancestor(jb,gg)$, which yields the answer $z \mapsto wm$, and a third failing computation of the goal $ancestor(jb,jb)$. These failing paths are pruned by rewriting. Simplification by previously generated answers reduces the number of recursive applications of the definition of *ancestor* and the amount of backtracking performed by the interpreter.

8.2 Linear Completion: inference rules and search plan

As we have seen in the previous section, rewrite programs allow us to define predicates by either equivalences or implications. A predicate is *mutually exclusively defined* if it is defined by a set of clauses such that no two heads unify[2]. If a predicate is mutually exclusively defined, its rewrite rules are equivalences, otherwise implications. More precisely, if a predicate A is defined by a set of clauses

[1] We shall see in the following that the inference mechanism of Linear Completion includes overlap between the current goal and previously generated answers. In this case the current goal is oriented from right to left and the previously generated answer does not overlap on the greater side. Therefore no further step is possible.

[2] According to this definition, the clauses $p(x) :- x > 0 \ldots$ and $p(x) :- x \leq 0 \ldots$ are not mutually exclusive and clearly this is not entirely satisfactory. However, this definition is sufficient for the purposes of this discussion. A more general notion of mutually exclusive clauses can be found in [25].



$\{Z \mapsto lc\}$
$\{Z \mapsto gg\}$
$\{Z \mapsto wm\}$

Figure 8.3: Optimization of the computation by simplification

$A(\bar{t}_1).$

$\ldots$

$A(\bar{t}_m).$

$A(\bar{t}_{m+1}) :- B_{11} \ldots B_{1p_1}.$

$\ldots$

$A(\bar{t}_{m+n}) :- B_{n1} \ldots B_{np_n}.$

its rewrite program contains the rules

$A(\bar{t}_1) \to true.$

$\ldots$

$A(\bar{t}_m) \to true.$

$A(\bar{t}_{m+1}) \to B_{11} \ldots B_{1p_1}.$

$\ldots$

$A(\bar{t}_{m+n}) \to B_{n1} \ldots B_{np_n}.$

if A is mutually exclusively defined and $\forall i, 1 \leq i \leq n, A(\bar{t}_{m+i}) \succ B_{i1} \ldots B_{ip_i}$, where $\succ$ is a complete simplification ordering. Otherwise A is transformed into

$A(\bar{t}_1) \to true.$

$\ldots$

$A(\bar{t}_m) \to true.$

$A(\bar{t}_{m+1})B_{11} \ldots B_{1p_1} \to B_{11} \ldots B_{1p_1}.$

$\ldots$

$A(\bar{t}_{m+n})B_{n1} \ldots B_{np_n} \to B_{n1} \ldots B_{np_n}.$

We assume a complete simplification ordering $\succ$ on terms and literals such that $P \succ answer(\bar{t}) \succ true$ for all atoms P whose predicate symbol is different from *answer* and *true* and for all terms $\bar{t}$. We call the rewrite rules representing facts, implications and bi-implications *fact rules*, *if-rules* and *iff-rules*. A *rewrite program* is a rewrite system of if-rules, iff-rules and fact rules. If a program has only if-rules and fact rules, then we also call it an *if-program*. Note that every Prolog program can be transformed into an if-program. Otherwise it is called an *iff-program*.

Rewrite programs are interpreted by *Linear Completion*. The input data for Linear Completion are a rewrite program, that is the presentation, and a query, that is the target or goal.

A query $\exists \bar{x} Q_1 \ldots Q_m$ is negated into $Q_1 \ldots Q_m \to false$ and written as a *query rule* $Q_1 \ldots Q_m \to answer(\bar{x})$, where $\bar{x}$ contains all the free variables in $Q_1 \ldots Q_m$. When a rule in the form $answer(\bar{x})\sigma \to true$ is deduced, $\bar{x}\sigma$ is a solution to the query. The *answer* literal was introduced originally in [42] and used in [30].

An answer rule means a contradiction in the refutational sense, since *answer* means $false$ logically. The interpreter builds a refutation starting with the query and ending with a contradiction:

$E \cup \{Q_1 \ldots Q_m \to answer(\bar{x})\} \vdash_{LC} answer(\bar{x})\sigma \to true$

We also denote it by $E \vdash_{LC} Q_1 \ldots Q_m\sigma$, meaning that $Q_1 \ldots Q_m\sigma$ is proved from program E by Linear Completion. If a ground query is given, the answer substitution is empty and we write $E \vdash_{LC} Q_1 \ldots Q_m$.

A computation by Linear Completion has the form

$(E; Q_1 \ldots Q_m \to answer(\bar{x}); \emptyset) \vdash_{LC} \ldots \vdash_{LC} (E_i; \bar{L}_i \to \bar{R}_i; S_i) \vdash_{LC} \ldots$

where E is the program, $\bar{L}_i \to \bar{R}_i$ is the current goal and S_i contains all the *ancestors* of the current goal. We use $\bar{L}$, $\bar{R}$ to denote conjunctions of atoms. A computation step applies one of the following inference rules:

Simplification:

$$\frac{(E; L_1 \ldots L_l \to R_1 \ldots R_r; S)}{(E; L'_1 \ldots L'_n \to R'_1 \ldots R'_s; S)}$$

Overlap:

$$\frac{(E; L_1 \ldots L_l \to R_1 \ldots R_r; S)}{(E; L'_1 \ldots L'_n \to R'_1 \ldots R'_s; S \cup \{L_1 \ldots L_l \to R_1 \ldots R_r\})}$$

Answer:

$$\frac{(E; answer(\bar{x})\sigma \rightarrow true; S)}{(E \cup \{answer(\bar{x})\sigma \rightarrow true\}; —; S)}$$

Delete:

$$\frac{(E; L_1 \dots L_l \leftrightarrow L_1 \dots L_l; S)}{(E; —; S)}$$

In *Simplification*, $L_1 \dots L_l \rightarrow R_1 \dots R_r$ is simplified into a new goal $L'_1 \dots L'_n \rightarrow R'_1 \dots R'_s$ by using $E \cup S$ and $L_1 \dots L_l \rightarrow R_1 \dots R_r$ is discarded. Simplification of goals by goals is the basic difference between our definition of Linear Completion and the one in [30]. In *Overlap* a new goal is generated by overlapping the current goal with a rule in E. The new goal replaces the current one, which is moved to the ancestors set. In *Answer* an answer is found and the answer rule is added to the program. In *Delete* a goal which is an identity is deleted.

No overlap between two program rules, no overlap between two goal rules and no simplification of program rules are used: all inference steps are target inference steps and therefore all derivations are linear input derivations. The name Linear Completion emphasizes this property.

We describe a computation by Linear Completion as a process of visiting an I-tree rooted at $(E; Q_1 \dots Q_m \rightarrow answer(\bar{x}); \emptyset)$, where I is the above set of inference rules.

In theorem proving a completion procedure is applied to a *validity* problem, such as $E_0 \models \forall \bar{x} s_0 \simeq t_0$, given in input as $(E_0; \hat{s}_0 \simeq \hat{t}_0)$. Therefore, the computation halts successfully as soon as a pair $(E_k; true)$ is reached. A successful computation is a path in the I-tree from the root $(E_0; \hat{s}_0 \simeq \hat{t}_0)$ to a node $(E_k; true)$.

In logic programming, a query $\exists \bar{x} Q_1 \dots Q_m$ is a *satisfiability* problem. If we are interested in just one answer, a successful computation is a path in the I-tree from the root $(E; \bar{Q} \rightarrow answer(\bar{x}); \emptyset)$ to a node $(E; answer(\bar{x})\sigma \rightarrow true; S_k)$. However, in practice a programmer is interested in extracting all the answers from the given data. Therefore, the computation halts successfully when all the answers have been discovered, that is when all the nodes $(E_k; answer(\bar{x})\sigma \rightarrow true; S_k)$ in the I-tree have been reached. A successful computation is a finite subtree of the I-tree, such that the leaves of the subtree are all and only the answers to the problem. The search plan determines how such a subtree is generated. If the computation reaches a stage where no inference rule applies, it backtracks. This is the case for instance when the target is empty, such as after an *Answer* or *Delete* step.

A search plan for theorem proving may include backtracking as well. It may pursue a path in the I-tree, backtrack for one or more steps and choose another branch. But the final result of a successful computation is a single path. Backtracking may be used to determine such path. A search plan for logic programming needs backtracking because the final result of a successful computation is a subtree with all the answers.

A search plan for Linear Completion tries the inference rules in the following order: *Delete, Answer, Simplification* and *Overlap*. Therefore, the current goal is always fully simplified before the next overlap step is performed. This choice ensures that the interpreter performs an overlap step and expands the search space only if the current search space has been pruned first as much

as possible.

The search plan is required to backtrack only on the goal and the ancestors set. Modifications on the program are never undone. Since the only modification to the program is the addition of answer rules, it follows that the interpreter does not forget the already computed answers and applies them as rules while visiting other paths in the tree. The *ancestor* example in Section 8.1 shows that keeping answers rules around is necessary to generate all the intended answer substitutions. In addition, simplification by answers rules contribute to prune the search tree and therefore to optimize the computation.

The overlap steps in LC are similar to the resolution steps in Prolog. Given a goal rule $A(\bar{u})L_1 \dots L_l \to R_1 \dots R_r$ and a program rule, one of the three overlapping inferences can be used according to the type of the program rule:

Overlap with an if-rule:

$$\frac{A(\bar{s})B_1 \dots B_n \to B_1 \dots B_n,\ A(\bar{u})L_1 \dots L_l \to R_1 \dots R_r}{(B_1 \dots B_n L_1 \dots L_l)\sigma \leftrightarrow (B_1 \dots B_n R_1 \dots R_r)\sigma}$$

Overlap with an iff-rule:

$$\frac{A(\bar{s}) \to B_1 \dots B_n,\ A(\bar{u})L_1 \dots L_l \to R_1 \dots R_r}{(B_1 \dots B_n L_1 \dots L_l)\sigma \leftrightarrow (R_1 \dots R_r)\sigma}$$

Overlap with a fact rule:

$$\frac{A(\bar{s}) \to true,\ A(\bar{u})L_1 \dots L_l \to R_1 \dots R_r}{(L_1 \dots L_l)\sigma \leftrightarrow (R_1 \dots R_r)\sigma}$$

where σ is the most general unifier of $A(\bar{s})$ and $A(\bar{u})$ and the generated goal is oriented according to the assumed simplification ordering. Here and in the following we assume that the leftmost literal in a goal is selected[3]. An overlap step replaces a literal in the goal by a proper instance of its predicate's definition. If the defining formula is an implication, the new set of subgoals is added to both sides of the goal equation. If it is a bi-implication, it is added to the left side only. An overlap with a fact deletes a literal in the goal list.

No overlap on an atom different from the head of a rule needs to be considered. If the atom $A(\bar{u})$ in the goal rule $A(\bar{u})\bar{L} \to \bar{R}$ unifies with mgu σ with an atom $A(\bar{v})$ in an if-rule $CA(\bar{v})\bar{B} \to A(\bar{v})\bar{B}$ the overlap step generates the goal

$$(C\bar{B}\bar{R})\sigma \to (A(\bar{v})\bar{B}\bar{L})\sigma$$

which is reduced by its predecessor $A(\bar{u})\bar{L} \to \bar{R}$ to

$$(C\bar{B}\bar{R})\sigma \to (\bar{B}\bar{R})\sigma.$$

A following overlap on literal C between this new goal and the same program rule $CA(\bar{v})\bar{B} \to A(\bar{v})\bar{B}$ will lead to the identity $(A(\bar{v})\bar{B}\bar{R})\sigma \leftrightarrow (A(\bar{v})\bar{B}\bar{R})\sigma$.

[3]It is very well known that computations differing in the order of selection of literals give the same answers up to renaming of variables [24, 67].

The key feature of rewrite programs is *simplification* of the current goal by program rules, ancestor goal rules and answer rules. If an if-rule $AB_1 \dots B_n \to B_1 \dots B_n$ or a fact $A \to true$ simplifies a query $Q_1 \dots Q_m$, then the atom Q_j such that $A\sigma = Q_j$ is deleted. If an iff-rule $A \to B_1 \dots B_n$ applies, the atom Q_j is replaced by $(B_1 \dots B_n)\sigma$. The rewrite rules $x \cdot true \to x$ and $x \cdot x \to x$ are also implicitly applied. They allow us to delete any repeated atom and any occurrence of *true* in a conjunction. Since the product is associative, commutative and idempotent, we regard conjunctions of atoms as *sets* of atoms: the left side $\bar{L}$ of an if-rule or of an ancestor goal rule matches a side $\bar{R}$ of the current goal rule if there is a subset of $\bar{R}$ which is an instance of $\bar{L}$. No AC-matching is needed, since all the matching operations occur below the product, between pairs of literals.

8.3 A fixpoint characterization of rewrite programs

Rewrite programs have operational, model theoretic and fixpoint semantics. Let E be a rewrite program, $\mathcal{B}$ be its Herbrand base and $\mathcal{P}(\mathcal{B})$ be the set of all subsets of $\mathcal{B}$, i.e. the set of all the Herbrand interpretations. The operational semantics of E is its *success set*, $\{G \mid G \in \mathcal{B},\ E \vdash_{LC} G\}$. The model theoretic semantics is the set $\{G \mid G \in \mathcal{B},\ E^* \models G \simeq true\}$, where $E^* = E \cup \{x \cdot x \to x,\ x \cdot true \to x\}$. If $E \vdash_{LC} Q_1 \dots Q_m\sigma$, we say that σ is a *correct answer substitution* for the query $Q_1 \dots Q_m$ if $E^* \models Q_1 \dots Q_m\sigma \simeq true$.

We define a *least fixpoint semantics* of rewrite programs on the lattice $\mathbb{IB} = \{I' \mid I' = I \cup \{true\}, I \subseteq \mathcal{B}\}$. $\mathbb{IB}$ is $\mathcal{P}(\mathcal{B})$ where the element *true* is added to each subset of $\mathcal{B}$. The order relation on $\mathbb{IB}$ is set inclusion $\subseteq$, the greatest lower bound operation is intersection $\bigcap$ and the least upper bound operation is union $\bigcup$. The bottom element is $\{true\}$ and the top element is $\mathcal{B} \cup \{true\}$. A function $T_E : \mathbb{IB} \to \mathbb{IB}$ is associated to a program E as follows:

Definition 8.3.1 *Given a rewrite program E, its associated function is the function $T_E : \mathbb{IB} \to \mathbb{IB}$ such that $P \in T_E(I)$ if and only if there exists in E a rule $A_1 \dots A_n \leftrightarrow B_1 \dots B_m$ $(n \geq 1, m \geq 1)$ such that $P = A_i\sigma$ and $\{A_1\sigma, \dots, A_{i-1}\sigma, A_{i+1}\sigma, \dots, A_n\sigma, B_1\sigma, \dots, B_m\sigma\} \subseteq I$ for some i, $1 \leq i \leq n$, and some ground substitution σ. (The double arrow $\leftrightarrow$ means there is no distinction between the left hand side and the right hand side.)*

Lemma 8.3.1 *Given a rewrite program E, T_E is continuous, that is for every non decreasing chain $X_1 \subseteq X_2 \subseteq \dots$ of elements in $\mathbb{IB}$, $T_E(\bigcup\{X_i \mid i < \omega\}) = \bigcup\{T_E(X_i) \mid i < \omega\}$.*

Proof: let $X_1 \subseteq X_2 \subseteq \dots$ be a chain in $\mathbb{IB}$.

1. $T_E(\bigcup\{X_i \mid i < \omega\}) \subseteq \bigcup\{T_E(X_i) \mid i < \omega\}$:

   $P \in T_E(\bigcup\{X_i \mid i < \omega\})$ if and only if there exists a ground instance $A_1\sigma \dots A_{j-1}\sigma P A_{j+1}\sigma \dots A_n\sigma \leftrightarrow B_1\sigma \dots B_m\sigma$ of a rule in E and $\{A_1\sigma \dots A_{j-1}\sigma, A_{j+1}\sigma \dots A_n\sigma, B_1\sigma \dots B_m\sigma\} \subseteq \bigcup\{X_i \mid i < \omega\}$. Then $\{A_1\sigma \dots A_{j-1}\sigma, A_{j+1}\sigma \dots A_n\sigma, B_1\sigma \dots B_m\sigma\} \subseteq X_i$ for some i, so that $P \in T_E(X_i)$ and $P \in \bigcup\{T_E(X_i) \mid i < \omega\}$.

2. $\bigcup\{T_E(X_i) \mid i < \omega\} \subseteq T_E(\bigcup\{X_i \mid i < \omega\})$:

$P \in \bigcup\{T_E(X_i) \mid i < \omega\}$ if and only if $P \in T_E(X_i)$ for some i if and only if there exists a ground instance $A_1\sigma \ldots A_{j-1}\sigma P A_{j+1}\sigma \ldots A_n\sigma \leftrightarrow B_1\sigma \ldots B_m\sigma$ and $\{A_1\sigma \ldots A_{j-1}\sigma, A_{j+1}\sigma \ldots A_n\sigma, B_1\sigma \ldots B_m\sigma\} \subseteq X_i$. Since $X_i \subseteq \bigcup\{X_i \mid i < \omega\}$, it follows that $P \in T_E(\bigcup\{X_i \mid i < \omega\})$. □

It follows that T_E has the properties of continuous functions on a lattice. Namely the least fixpoint of T_E is an ordinal power of T_E:

$$lfp(T_E) = T_E \uparrow \omega$$

where ordinal powers are defined on $\mathbb{B}$ in the usual way:

$$T \uparrow 0 = \{true\}$$

$$T \uparrow n = \begin{cases} T(T \uparrow (n-1)) & \text{if } n \text{ is a successor ordinal} \\ \bigcup\{T \uparrow k \mid k < n\} & \text{if } n \text{ is a limit ordinal.} \end{cases}$$

Example 8.3.1 *The least fixpoint of the rewrite program for the ancestor relation is obtained as follows:*

$T_E(\{true\}) = \{true, parent(jb, lc), parent(jb, gg), parent(gg, wm)\}$

$T_E^2(\{true\}) = T_E(\{true\}) \cup \{ancestor(jb, lc), ancestor(jb, gg), ancestor(gg, wm)\}$

$lfp(T_E) = T_E^3(\{true\}) = T_E^2(\{true\}) \cup \{ancestor(jb, wm)\}$.

8.4 Equivalence of proof theoretic, model theoretic and fix point semantics

We show that $lfp(T_E)$ is a fixpoint characterization of both the operational and model theoretic semantics of a program E. These results are obtained through a few steps. The first one is the following lemma:

Lemma 8.4.1 *For all conjunctions of atoms $Q_1 \ldots Q_m$, $Q_1 \ldots Q_m \leftrightarrow_{E^*}^* true$ if and only if $\forall j, 1 \leq j \leq m$, $Q_j \leftrightarrow_{E^*}^* true$.*

Proof:

$\Rightarrow$) The proof is by induction on the length i of the chain $Q_1 \ldots Q_m \leftrightarrow_{E^*}^i true$.

Base: if $i = 1$ then $m = 1$ and the thesis is trivially true.

Induction hypothesis: $\forall i, 1 < i \leq l$, $Q_1 \ldots Q_m \leftrightarrow_{E^*}^i true$ implies $\forall j, 1 \leq j \leq m$ $Q_j \leftrightarrow_{E^*}^* true$.

Induction step: if $i = l + 1$, then we have

either case 1: $Q_1 \ldots Q_m \rightarrow_{E^*} C \leftrightarrow_{E^*}^l true$

or case 2: $Q_1 \ldots Q_m \leftarrow_{E^*} C \leftrightarrow_{E^*}^l true$:

1. if a fact rule $A \rightarrow true$ such that $Q_1 = A\sigma$ is applied, then C is $Q_2 \ldots Q_m$. The atom Q_1 is rewritten to *true* by the applied fact rule. The induction hypothesis applies to the remaining atoms $Q_2 \ldots Q_m$.

Similarly, if an if-rule $AB_1 \dots B_n \rightarrow B_1 \dots B_n$ is applied to Q_1, C is $Q_2 \dots Q_m$. The induction hypothesis applies to the atoms $Q_2 \dots Q_m$. For Q_1 we have $Q_1 \leftrightarrow^*_{E^*} Q_1 true \dots true \leftrightarrow^*_{E^*} Q_1 Q_2 \dots Q_m \leftrightarrow^*_{E^*} true$ so that Q_1 is E^*-equivalent to $true$.

If an iff-rule $A \rightarrow B_1 \dots B_n$ is applied to Q_1, C is $(B_1 \dots B_n)\sigma Q_2 \dots Q_m$. The induction hypothesis applies to all the atoms in C, so that $\forall k, 2 \leq k \leq m, Q_k \leftrightarrow^*_{E^*} true$ and $Q_1 \rightarrow_{E^*} (B_1 \dots B_n)\sigma \leftrightarrow^*_{E^*} true$.

2. If C is rewritten to $Q_1 \dots Q_m$ by either a fact rule or an if-rule, then by induction hypothesis, $\forall j, 1 \leq j \leq m\ Q_j \leftrightarrow^*_{E^*} true$, since $\{Q_1 \dots Q_m\} \subset C$.

   If an iff-rule $A \rightarrow B_1 \dots B_n$ is applied to C, then if C is $D_1 \dots D_p$, $Q_1 \dots Q_m$ is $(B_1 \dots B_n)\sigma D_2 \dots D_p$. The induction hypothesis applies to the atoms $D_2 \dots D_p$. We are left with the atoms in $(B_1 \dots B_n)\sigma$. The atom D_1 has a mutually exclusively defined predicate, since it is rewritten by an iff-rule. It follows that the same iff-rule $A \rightarrow B_1 \dots B_n$ has to be applied to reduce D_1 in the equality chain between C and $true$. The product $(B_1 \dots B_n)\sigma$ is reduced to $true$ by a path shorter than l steps, so that the induction hypothesis applies to these atoms as well.

$\Leftarrow$) Trivial. □

The following theorem establishes the soundness of Linear Completion, i.e. that all the answers given by Linear Completion are correct answer substitutions:

Theorem 8.4.1 *If $E \vdash_{LC} Q_1 \dots Q_m\sigma$ then $E^* \models Q_1 \dots Q_m\sigma \simeq true$.*

Proof: soundness of Linear Completion follows from soundness of replacing equals by equals, since the inference rules of Linear Completion are applications of equational replacement.

$E \vdash_{LC} Q_1 \dots Q_m\sigma$ stands for $E \cup \{Q_1 \dots Q_m \rightarrow answer(\bar{x})\} \vdash_{LC} answer(\bar{x})\sigma \rightarrow true$. The equation $answer(\bar{x})\sigma \rightarrow true$ is derived by LC from E and $Q_1 \dots Q_m \rightarrow answer(\bar{x})$. Since the *answer* literal is just a place holder for $Q_1 \dots Q_m$, this actually means that the equation $Q_1 \dots Q_m\sigma \rightarrow true$ is derived from E by equational replacement, or $Q_1 \dots Q_m\sigma \leftrightarrow^*_{E^*} true$. Then, $Q_1 \dots Q_m\sigma \leftrightarrow^*_{E^*} true$ implies $E^* \models Q_1 \dots Q_m\sigma \simeq true$ by the soundness of replacing equals by equals. □

We now restrict our attention to ground queries. First we prove a lemma which allows us to split the problem of proving a ground conjunction into the subproblems of proving its single atoms. Intuitively this result holds because since the query is ground, there is no computation of an answer substitution and the processes of reducing to *true* the literals in the query are independent processes.

Lemma 8.4.2 *$\forall Q_1 \dots Q_m \in \mathcal{B}$, $E \vdash_{LC} Q_1 \dots Q_m$ if and only if $\forall j$, $1 \leq j \leq m$, $E \vdash_{LC} Q_j$.*

Proof:

$\Leftarrow$) The proof is done by way of contradiction. Assume that $E \vdash_{LC} Q_1 \dots Q_m$ does not hold. This means that all the paths in the computation tree generated by LC starting from the query

$Q_1 \ldots Q_m$ halt with a goal rule which is not an answer rule and such that no further step can be performed. Such a goal contains at least one ground literal A which is neither *answer*( ) nor *true*. Note that since the query is ground, all the *answer* literals are identical to *answer*( ) and each goal can contain at most one of them since $x \cdot x \to x$ is applied.

Since all the goal rules are ground, a simplification step of a goal by a goal does not generate new instances. It follows that such a literal A has been generated by a sequence of steps by program rules. Then, A is also generated in the computation tree starting from a query Q_j for some j, $1 \leq j \leq m$. Since $E \vdash_{LC} Q_j$, the program solves A contradicting our assumption that A cannot be solved.

$\Rightarrow$) The proof is by induction on the length i of the derivation $E \vdash^i_{LC} Q_1 \ldots Q_m$.

Base: if $i = 1$, then $m = 1$ and the result follows.

Induction hypothesis: $\forall i$, $1 < i \leq l$, $E \vdash^i_{LC} Q_1 \ldots Q_m$ implies $\forall j$, $1 \leq j \leq m$, $E \vdash_{LC} Q_j$.

Induction step: if $i = l + 1$, then a program rule must be applied:

1. if an iff-rule $A \to B_1 \ldots B_n$ such that $A\sigma = Q_1$ is applied, $E \vdash^l_{LC} (B_1 \ldots B_n)\sigma Q_2 \ldots Q_m$. By induction hypothesis we get $E \vdash_{LC} Q_j$ for $2 \leq j \leq m$ and $E \vdash_{LC} B_k\sigma$ for $1 \leq k \leq n$. Since an iff-rule applies to Q_1, its predicate is mutually exclusively defined. It follows that if Q_1 alone is given as a query it is necessarily reduced to $(B_1 \ldots B_n)\sigma$. Since all the $B_k\sigma$ are solved, $E \vdash_{LC} (B_1 \ldots B_n)\sigma$ and $E \vdash_{LC} Q_1$ follows.

2. If an if-rule $AB_1 \ldots B_n \to B_1 \ldots B_n$ such that $A\sigma = Q_1$ is applied, it means $Q_1 \ldots Q_m$ is $(AB_1 \ldots B_n)\sigma Q_{n+2} \ldots Q_m$ and $E \vdash^l_{LC} (B_1 \ldots B_n)\sigma Q_{n+2} \ldots Q_m$. By induction hypothesis we get $E \vdash_{LC} Q_j$ for $n+2 \leq j \leq m$ and $E \vdash_{LC} B_k\sigma$ for $1 \leq k \leq n$. Only Q_1 is missing. However, if Q_1 alone is given as a query there exists a computation which starts by overlapping Q_1 with $AB_1 \ldots B_n \to B_1 \ldots B_n$ reducing it to the query $(B_1 \ldots B_n)\sigma$. Since all the $B_k\sigma$ are solved, $E \vdash_{LC} (B_1 \ldots B_n)\sigma$ and $E \vdash_{LC} Q_1$ follows. □

We can now apply Lemma 8.4.2 to show that all ground queries proved by Linear Completion from E are equivalent to *true*.

Theorem 8.4.2 *$\forall Q_1 \ldots Q_m \in \mathcal{B}$, $E \vdash_{LC} Q_1 \ldots Q_m$ if and only if $Q_1 \ldots Q_m \leftrightarrow^*_{E^*} true$.*

Proof:

$\Rightarrow$) See the soundness theorem (Theorem 8.4.1).

$\Leftarrow$) The proof is by induction on the length i of the chain $Q_1 \ldots Q_m \leftrightarrow^i_{E^*} true$.

Base: if $i = 1$, then $m = 1$ and there is a fact rule $A \to true$ such that $Q_1 = A\sigma$. It follows that $E \vdash_{LC} Q_1$ in one step.

Induction hypothesis: $\forall i, 1 < i \leq l$, $Q_1 \ldots Q_m \leftrightarrow^i_{E^*} true$ implies $E \vdash_{LC} Q_1 \ldots Q_m$.

Induction step: if $i = l + 1$, then $Q_1 \ldots Q_m \leftrightarrow_{E^*} \bar{C} \leftrightarrow^l_{E^*} true$ and we consider two cases depending on the direction of the first step:

1. if $Q_1 \ldots Q_m \rightarrow_{E^*} \bar{C} \leftrightarrow^l_{E^*} true$, then the LC interpreter can derive the goal $\bar{C}$ from the query $Q_1 \ldots Q_m$ by applying this rewrite step. Since $E \vdash_{LC} \bar{C}$ holds by induction hypothesis, $E \vdash_{LC} Q_1 \ldots Q_m$ follows.

2. If $Q_1 \ldots Q_m \leftarrow_{E^*} \bar{C} \leftrightarrow^l_{E^*} true$, then there are two more cases.

   If an if-rule or a fact rule or $x \cdot x \rightarrow x$ or $x \cdot true \rightarrow x$ reduces $\bar{C}$ to $Q_1 \ldots Q_m$, all the atoms Q_j, $1 \leq j \leq m$, occur in $\bar{C}$. By induction hypothesis we have that $E \vdash_{LC} \bar{C}$. It follows by Lemma 8.4.2 that $E \vdash_{LC} Q_j$, $1 \leq j \leq m$ and $E \vdash_{LC} Q_1 \ldots Q_m$.

   If an iff-rule $A \rightarrow B_1 \ldots B_n$ reduces $\bar{C}$ to $Q_1 \ldots Q_m$, $\bar{C}$ is $C_1 \ldots C_p$, $C_1 = A\sigma$ and $Q_1 \ldots Q_m$ is $(B_1 \ldots B_n)\sigma C_2 \ldots C_p$. By induction hypothesis we have that $E \vdash_{LC} \bar{C}$. It follows by Lemma 8.4.2 that $E \vdash_{LC} C_q$, $1 \leq q \leq p$. Since $C_1 = A\sigma$ and A is the head of an iff-rule, the predicate of C_1 is mutually exclusively defined. It follows that in the computation $E \vdash_{LC} C_1$ this rule must be applied and $E \vdash_{LC} B_1 \ldots B_n\sigma$. It follows by Lemma 8.4.2 that $E \vdash_{LC} B_k\sigma$, $1 \leq k \leq n$. Now all atoms in $Q_1 \ldots Q_m$ are proved so that by applying again Lemma 8.4.2 we get $E \vdash_{LC} Q_1 \ldots Q_m$. □

These results allow us to relate the fixpoint semantics of a rewrite program E to its operational semantics:

Theorem 8.4.3 *$\forall Q_1 \ldots Q_m \in \mathcal{B}$, $Q_1 \ldots Q_m \leftrightarrow^*_{E^*} true$ if and only if $\forall i$, $1 \leq i \leq m$, $Q_i \in lfp(T_E)$.*

Proof:

$\Leftarrow$) We prove that if $Q_i \in lfp(T_E)$, then $Q_i \leftrightarrow^*_{E^*} true$. Then $Q_1 \ldots Q_m \leftrightarrow^*_{E^*} true$ follows.

If $Q_i \in lfp(T_E)$, then there exists a $j \geq 1$ such that $Q_i \in T^j_E(\{true\})$ and $Q_i \notin T^k_E(\{true\})$ for all $k < j$, i.e. j is the minimum power such that Q_i is included. The proof is done by induction on this power j.

Base: if $j = 1$, then there exists a fact rule $A \rightarrow true$ such that $Q_i = A\sigma$ for some ground substitution σ. It follows that $Q_i \leftrightarrow_{E^*} true$.

Induction hypothesis: $\forall j, 1 < j \leq l$, $Q_i \in T^j_E(\{true\})$ implies $Q_i \leftrightarrow^*_{E^*} true$.

Induction step: if $j = l+1$, then there exists a ground instance $A_1\sigma \ldots A_{k-1}\sigma Q_i A_{k+1}\sigma \ldots A_n\sigma \leftrightarrow B_1\sigma \ldots B_m\sigma$ of a rule in E such that $\{A_1\sigma \ldots A_{k-1}\sigma, A_{k+1}\sigma \ldots A_n\sigma, B_1\sigma \ldots B_m\sigma\} \subseteq T^l_E(\{true\})$. By induction hypothesis all the atoms in this set are E^*-equivalent to $true$. It follows that $Q_i \leftrightarrow^*_{E^*} true$ as well.

$\Rightarrow$) The proof is done by induction on the length j of $Q_1 \ldots Q_m \leftrightarrow^j_{E^*} true$.

Base: if $j = 1$, then $m = 1$ and $Q_1 = A\sigma$ for a fact rule $A \rightarrow true$. It follows that $Q_1 \in T_E(\{true\})$ and $Q_1 \in lfp(T_E)$.

Induction hypothesis: $\forall j, 1 < j \leq l$, $Q_1 \ldots Q_m \leftrightarrow^j_{E^*} true$ implies $Q_i \in lfp(T_E)$ for $1 \leq i \leq m$.

Induction step: if $j = l+1$, then $Q_1 \ldots Q_m \leftrightarrow_{E^*} \bar{C} \leftrightarrow^l_{E^*} true$ and we have two cases as in the proof of Theorem 8.4.2.

1. Let us consider $Q_1 \ldots Q_m \rightarrow_{E^*} \bar{C} \leftrightarrow^l_{E^*} true$. If $x \cdot x \rightarrow x$ or $x \cdot true \rightarrow x$ reduces $Q_1 \ldots Q_m$ to $\bar{C}$, all the atoms Q_i, $1 \leq i \leq m$ occur in $\bar{C}$ and so by induction hypothesis all of them are in $lfp(T_E)$. If $A \rightarrow true$ reduces $Q_1 \ldots Q_m$ to $\bar{C}$, the atom Q_k matched by A is in $T_E(\{true\})$ and so in $lfp(T_E)$. All the remaining atoms $Q_2 \ldots Q_m$ are in $\bar{C}$ and so in $lfp(T_E)$ by induction hypothesis. If $AB_1 \ldots B_n \rightarrow B_1 \ldots B_n$ applies, then $(AB_1 \ldots B_n)\sigma = Q_1 Q_2 \ldots Q_{n+1}$ with $n+1 < m$ and C is $Q_2 \ldots Q_m$. The atoms $Q_2 \ldots Q_m$ are in $lfp(T_E)$ by induction hypothesis. Since $(B_1 \ldots B_n)\sigma = Q_2 \ldots Q_{n+1}$, the atoms $B_1\sigma \ldots B_n\sigma$ are in $lfp(T_E)$. Let p be the minimum power such that all the $B_k\sigma$'s are in $T^p_E(\{true\})$. It follows that $Q_1 \in T^{p+1}_E(\{true\})$ and so in $lfp(T_E)$. If an iff-rule $A \rightarrow B_1 \ldots B_n$ reduces $Q_1 \ldots Q_m$ to $\bar{C}$, then $Q_1 = A\sigma$ and $\bar{C}$ is $(B_1 \ldots B_n)\sigma Q_2 \ldots Q_m$. By induction hypothesis all the atoms in $\bar{C}$ are in $lfp(T_E)$. Let p be the minimum power such that all the $B_k\sigma$'s are in $T^p_E(\{true\})$. It follows that $Q_1 \in T^{p+1}_E(\{true\})$ and so in $lfp(T_E)$.

2. If $Q_1 \ldots Q_m \leftarrow_{E^*} \bar{C} \leftrightarrow^l_{E^*} true$ and an if-rule or a fact rule or $x \cdot x \rightarrow x$ or $x \cdot true \rightarrow x$ reduces $\bar{C}$ to $Q_1 \ldots Q_m$, all the atoms Q_i, $1 \leq i \leq m$, occur in $\bar{C}$. By induction hypothesis all of them are in $lfp(T_E)$. If an iff-rule $A \rightarrow B_1 \ldots B_n$ applies, then $\bar{C}$ is $C_1 \ldots C_h$, $C_1 = A\sigma$ and $Q_1 \ldots Q_m$ is $(B_1 \ldots B_n)\sigma C_2 \ldots C_h$. By induction hypothesis we have that all the atoms in $\bar{C}$ are in $lfp(T_E)$. Only the atoms in $(B_1 \ldots B_n)\sigma$ are missing. However, since an iff-rule applies, it means that the predicate of C_1 is mutually exclusively defined. It follows that in the chain $\bar{C} \leftrightarrow^l_{E^*} true$ this rule must be applied, i.e. the proof has the form $\bar{C} \leftrightarrow^{j_1}_{E^*} \ldots (B_1 \ldots B_n)\sigma \ldots \leftrightarrow^{j_2}_{E^*} true$, where $j_2 < l$. By applying the induction hypothesis to the last subchain we get that all the $B_k\sigma$'s are in $lfp(T_E)$ as well. □

Therefore, the fixpoint semantics $lfp(T_E)$ captures both the operational and model theoretic semantics of the rewrite program E:

Theorem 8.4.4 *$\forall G \in \mathcal{B}$, $G \in lfp(T_E)$ if and only if $E \vdash_{LC} G$.*

Proof: $G \in lfp(T_E)$ if and only if $G \leftrightarrow^*_{E^*} true$ by Theorem 8.4.3 if and only if $E \vdash_{LC} G$ by Theorem 8.4.2. □

Theorem 8.4.5 *$\forall G \in \mathcal{B}$, $G \in lfp(T_E)$ if and only if $E^* \models G \simeq true$.*

Proof: $G \in lfp(T_E)$ if and only if $G \leftrightarrow^*_{E^*} true$ by Theorem 8.4.3 if and only if $E^* \models G \simeq true$ by completeness of replacing equals by equals (Birkhoff's theorem). □

8.5 Denotational semantics of rewrite programs

The above fixpoint characterization of rewrite programs is basically equivalent to the fixpoint characterization of Prolog programs. The fixpoint semantics of a Prolog program P is $lfp(T_P) = T_P \uparrow \omega$, where T_P is the function $T_P : \mathcal{P}(\mathcal{B}) \rightarrow \mathcal{P}(\mathcal{B})$ such that $A \in T_P(I)$ if and only if there exists in P a clause $A' :- B_1 \ldots B_m$ $(m \geq 0)$ such that $A = A'\sigma$ and $\{B_1\sigma \ldots B_m\sigma\} \subseteq I$ for some ground substitution σ [64, 6]. The following theorem shows that the two semantics are indeed

the same. We write $E \equiv P$ and we say that the rewrite program E corresponds to the Prolog program P if they define the same predicates.

Theorem 8.5.1 *If $E \equiv P$, then $lfp(T_E) = lfp(T_P)$.*

Proof:

1. $lfp(T_E) \subseteq lfp(T_P)$.

   If $G \in lfp(T_E)$, then there exists an $i \geq 1$ such that $G \in T_E^i(\{true\})$ and $\forall j < i, G \notin T_E^j(\{true\})$. The proof is done by induction on the power i.

   Base: if $i = 1$, then there exists a fact rule $A \rightarrow true$ in E such that $G = A\sigma$ for some ground substitution σ. Since $E \equiv P$, the fact $A.$ belongs to the Prolog program P and $G \in lfp(T_P)$.

   Induction hypothesis: $\forall i, 1 < i \leq l$, $G \in T_E^i(\{true\})$ implies $G \in lfp(T_P)$.

   Induction step: if $i = l + 1$, there exists a ground instance $C_1\sigma \ldots C_{j-1}\sigma G C_{j+1}\sigma \ldots C_p\sigma \leftrightarrow D_1\sigma \ldots D_q\sigma$ of a rule in E whose atoms but G are in $T_E^l(\{true\})$. By induction hypothesis all the atoms in this set belong to $lfp(T_P)$. It follows that there is some $k \geq 1$ such that $\{C_1\sigma \ldots C_{j-1}\sigma, C_{j+1}\sigma \ldots C_p\sigma, D_1\sigma \ldots D_q\sigma\} \subseteq T_P^k(\emptyset)$. The above ground rule is either an instance of an if-rule $AB_1 \ldots B_n \rightarrow B_1 \ldots B_n$ or an instance of an iff-rule $A \rightarrow B_1 \ldots B_n$. Since $P \equiv E$, in both cases the Prolog program P contains the corresponding clause $A :- B_1 \ldots B_n$. In the if-rule case, if $G = B_h\sigma$ for some $h, 1 \leq h \leq n$, then it is in $lfp(T_P)$ by induction hypothesis, because each B_h occurs twice in the rule. If $G = A\sigma$, then $G \in T_P^{k+1}(\emptyset)$ and $G \in lfp(T_P)$. In the iff-rule case, if $G = A\sigma$, then $G \in T_P^{k+1}(\emptyset)$ and $G \in lfp(T_P)$. If $G = B_h\sigma$ for some $h, 1 \leq h \leq n$, then $A\sigma$ is already in $T_P^k(\emptyset)$. Since the A's predicate is mutually exclusively defined, $A\sigma$ must have been included in $T_P^k(\emptyset)$ because of the clause $A :- B_1 \ldots B_n$ and $G = B_h\sigma$ must be in $T_P^z(\emptyset)$ for some $z < k$, so that $G \in lfp(T_P)$.

2. $lfp(T_P) \subseteq lfp(T_E)$.

   The proof is analogous to the previous one, applying the same induction argument on the power of T_P and exploiting the correspondence between the two programs. □

This theorem establishes that a Prolog program and a rewrite program defining the same predicates have the same model. For a ground atom G, $E \vdash_{LC} G$, $G \leftrightarrow_{E^*}^* true$, $E^* \models G \simeq true$, $G \in lfp(T_E)$, $G \in lfp(T_P)$, $P \models G$ and $P \vdash_{Prolog} G$ are all equivalent. The behaviour of the two programs P and E may be different, though. The rewrite program E may generate less answers than the corresponding Prolog program P. However, for all answers given by P there is an answer given by E which is E-equivalent. In order to obtain this result, we first prove that all the answers given by Linear Completion are also given by Prolog. This result follows from completeness of SLD-resolution.

Theorem 8.5.2 *If $E \equiv P$ and $E \vdash_{LC} Q_1 \ldots Q_m\sigma$, there exists an answer θ, $P \vdash_{Prolog} Q_1 \ldots Q_m\theta$, such that $\sigma = \theta\rho$ for some substitution ρ.*

Proof: if $E \vdash_{LC} Q_1 \ldots Q_m\sigma$, then σ is a correct answer substitution by soundness of Linear Completion. Then by completeness of SLD-resolution there exists a computed answer substitution θ, $P \vdash_{Prolog} Q_1 \ldots Q_m\theta$ and a substitution ρ, such that $\sigma = \theta\rho$. □

We now prove that Linear Completion is also complete, by proving that all Prolog answers are represented by some answers given by Linear Completion.

In the following we assume that all queries are single literal queries. There is no loss of generality because a query $Q_1 \ldots Q_m \rightarrow answer(\bar{x})$ can be written as a single literal query by introducing a new predicate symbol N, a new program rule $N \rightarrow Q_1 \ldots Q_m$ and the query $N \rightarrow answer(\bar{x})$.

First of all we prove that rewrite programs and Prolog programs give the same answers if Linear Completion is restricted to overlap steps only, i.e. no simplification is performed. This is straightforward, since overlap steps and resolution steps clearly correspond:

Theorem 8.5.3 *Let LC' be a subset of the Linear Completion interpreter performing overlap steps only. If $E^* \models G\theta \simeq true$, i.e. θ is a correct answer for G, there exists a computed answer σ, $E \vdash_{LC'} G\sigma$, such that $\theta = \sigma\rho$ for some substitution ρ.*

Proof: we prove that if $E \vdash_{LC'} G\sigma$ then $P \vdash_{Prolog} G\sigma$ and vice versa. Completeness of LC' then follows from completeness of SLD-resolution. Overlap steps correspond to resolution steps and vice versa. The same unifiers are generated at each step. Since an answer is given by the composition of the unifiers generated by overlap/resolution steps, the two programs generate the same answer substitutions. This holds regardless of whether a Prolog step corresponds to an overlap step by an if-rule or to an overlap step by an iff-rule. If an iff-rule overlaps the current goal, the body of the rule is added to the left hand side of the new goal rule only, whereas an if-rule overlap adds the body to both sides of the new goal rule. However, having an atom occurring on both sides of a goal is irrelevant as far as overlap steps and generation of unifiers are concerned. When the left occurrence of an atom is eliminated by an overlap step, its right occurrence is instantiated by the mgu of the overlap step and will be eventually eliminated by an overlap step whose mgu does not affect the variables in the goal. Note that overlaps with answer rules do not occur, since *answer* atoms may occur on the left hand side of a goal only as effect of previous simplification by goals, which is not performed by LC'. □

In order to prove an analogous result for Linear Completion with simplification we first need to prove two more lemmas.

Lemma 8.5.1 *If Linear Completion generates a computation path $(E; G \rightarrow answer(\bar{x}); \emptyset) \vdash^*_{LC} (E; \bar{W} \rightarrow \bar{V}; \_) \vdash^*_{LC} (E; H; \_) \vdash_{LC} (E; \bar{R} \leftrightarrow \bar{R}; \_)$, where the goal H is simplified to an identity by its ancestor $\bar{W} \rightarrow \bar{V}$, then H is $\bar{Z}(\bar{W}\lambda) \rightarrow \bar{Z}(\bar{V}\lambda)$ for some substitution λ and literals $\bar{Z}$. (The hyphen $\_$ in $(E; H; \_)$ means that the third component is not relevant.)*

Proof: let H be $\bar{S}\bar{X} \rightarrow \bar{S}\bar{Y}$, where $\bar{S}$ are the literals occurring on both sides, the literals in $\bar{X}$ occur on the left side only and the literals in $\bar{Y}$ occur on the right side only. If $\bar{W} \rightarrow \bar{V}$ applies to both sides of H, then $\bar{W}$ matches a subset of $\bar{S}$, i.e. $\bar{S} = (\bar{W}\lambda)\bar{Z}$ for some substitution λ and

$\bar{W} \to \bar{V}$ rewrites H to $(\bar{V}\lambda)\bar{Z}\bar{X} \to (\bar{V}\lambda)\bar{Z}\bar{Y}$ against the hypothesis that $\bar{W} \to \bar{V}$ rewrites H to an identity. It follows that $\bar{W} \to \bar{V}$ applies to one side of H only.

We assume that $\bar{W} \to \bar{V}$ applies to the left side of H. The case where it applies to the right side is symmetrical. It follows that H is reduced to $\bar{S}\bar{Y} \to \bar{S}\bar{Y}$. If $\bar{W}$ matches only a subset of $\bar{X}$, then the unmatched subset of $\bar{X}$ would still appear on the left side and not on the right side after the simplification step, contradicting the hypothesis that $\bar{W} \to \bar{V}$ rewrites H to an identity. It follows that $\bar{W}$ must match $\bar{X}$ completely and possibly a subset of $\bar{S}$: we have $\bar{W} = \bar{W}_1\bar{W}_2$, $\bar{S} = (\bar{W}_1\lambda)\bar{Z}$ and $\bar{X} = (\bar{W}_2\lambda)$. The left side of H is $(\bar{W}_1\lambda)\bar{Z}(\bar{W}_2\lambda)$ and it is rewritten to $\bar{Z}(\bar{V}\lambda)$. Furthermore, $\bar{Z}(\bar{V}\lambda) = \bar{S}\bar{Y}$, since the left side of H must be rewritten to its right side. Since $\bar{S} = \bar{(}W_1\lambda)\bar{Z}$, it follows that $(\bar{V}\lambda) = (\bar{W}_1\lambda)\bar{Y}$, i.e. $\bar{V} = \bar{W}_1\bar{W}_3$, where $\bar{W}_3\lambda = \bar{Y}$. It follows that $\bar{W} \to \bar{V}$ is $\bar{W}_1\bar{W}_2 \to \bar{W}_1\bar{W}_3$ and H is $\bar{Z}(\bar{W}_1\bar{W}_2\lambda) \to \bar{Z}(\bar{W}_1\bar{W}_3\lambda)$. □

The following lemma shows that Linear Completion with simplification is refutationally complete. First, we introduce some terminology used in the proof: a computation path starting at the root, i.e. the query, and halting with an answer is a *successful path.* A computation path starting at the root and halting with a goal such that no inference rule applies is an *unsuccessful path.* There are three kinds of unsuccessful path in an LC tree:

- unsuccessful paths where the goal is reduced to an identity, which is next deleted by a *Delete* step,
- unsuccessful paths where the goal is reduced to an equation containg only *answer* literals to which no inference rule applies and
- unsuccessful paths where the last goal still contains literals whose predicate is not *answer*, but no program rule applies to them.

The first two kinds of unsuccessful paths are determined by simplification and therefore they do not occur in computations by LC'. The third kind corresponds to the notion of *finite failure* in Prolog and is the only kind of unsuccessful path in an LC' computation. We use Δ to indicate either an identity or an equation containg only *answer* literals to which no inference rule applies.

Lemma 8.5.2 *Given a query G, if there exists an answer θ generated by LC', i.e. $E \vdash_{LC'} G\theta$, then there exists an answer σ generated by LC, i.e. $E \vdash_{LC} G\sigma$.*

Proof: the proof is done by way of contradiction. Assuming that LC does not generate any answer, we consider the tree T_{LC} generated by LC for the query G. Since LC does not generate any answer, all paths in T_{LC} are unsuccessful. LC differs from LC' because of simplification of goals by program rules, by their ancestors and by previously generated answers. Simplification by rules in the program is irrelevant since a simplification step by a program rule is an overlap step where the unifier is a matching substitution. Simplification and overlap with previously generated answers do not apply because we assume that LC does not generate any answer for the query G. Therefore, we only have to consider the case where all the successful paths generated by LC' are pruned by simplification by ancestor goal rules in LC. More precisely, all the successful paths

generated by LC' are replaced as an effect of simplification by ancestors by unsuccessful paths ending with a Δ goal:

$(E; G \to answer(\bar{x}); \emptyset) \vdash^*_{LC} (E; H'; \_) \vdash^*_{LC} (E; H; \_) \vdash_{LC} (E; \Delta; \_)$,

where H' simplifies H to Δ.

For any such path α, we denote by $I(\alpha)$ the set of all the successful paths in the computation tree generated by LC' which have the subpath $(E; G \to answer(\bar{x}); \emptyset) \vdash^*_{LC'} (E; H'; \_) \vdash^*_{LC'} (E; H; \_)$ as a prefix, i.e. all the paths in the form:

$(E; G \to answer(\bar{x}); \emptyset) \vdash^*_{LC'} (E; H'; \_) \vdash^*_{LC'} (E; H; \_) \vdash^*_{LC'} (E; answer(\bar{x})\theta \to true; \_)$.

Given a set of paths A, we denote by $min(A)$ the shortest path in the set A. If it is not unique, we just select one among the shortest paths. Then $min(I(\alpha))$ is the shortest path in the set $I(\alpha)$. We consider the path α^*, defined as follows:

$\alpha^* = min(\{min(I(\alpha)) | \alpha \in T_{LC}\})$.

α^* has the form

$(E; G \to answer(\bar{x}); \emptyset) \vdash^*_{LC'} (E; H'; \_) \vdash^*_{LC'} (E; H; \_) \vdash^*_{LC'} (E; answer(\bar{x})\theta \to true; \_)$.

By our assumptions, α^* is pruned by goal simplification in the tree T_{LC}. We show that if this is the case, then LC' generates a successful path shorter than α^*, which is a contradiction. There are two cases:

case 1: the goal H is reduced by its ancestor H' to an identity,

case 2: the goal H is reduced by its ancestor H' to an equation containing only *answer* literals to which no inference rule applies.

1. By Lemma 8.5.1, it follows that if H' is $\bar{W} \to \bar{V}$, then H is $\bar{Z}(\bar{W}\lambda) \to \bar{Z}(\bar{V}\lambda)$ for some substitution λ and literals $\bar{Z}$. We know that $(E; \bar{Z}(\bar{W}\lambda) \to \bar{Z}(\bar{V}\lambda); \_) \vdash^*_{LC'} (E; answer(\bar{x})\theta \to true; \_)$. The literals $\bar{Z}$ must be eliminated in order to achieve a solution. Since the order of literals selection for overlap/resolution does not affect the answers generated by LC' or Prolog, we can rearrange the order of steps in the path α^* in such a way that the literals in $\bar{Z}$ are eliminated first. This does not modify the length of the path. It follows that α^* is $(E; G \to answer(\bar{x}); \emptyset) \vdash^*_{LC'} (E; \bar{W} \to \bar{V}; \_) \vdash^*_{LC'} (E; \bar{Z}(\bar{W}\lambda) \to \bar{Z}(\bar{V}\lambda); \_) \vdash^*_{LC'} (E; \bar{W}\lambda \to \bar{V}\lambda; \_) \vdash^*_{LC'} (E; answer(\bar{x})\theta \to true; \_)$. Such a path cannot be the shortest path to a solution. If we unify a literal $P(\bar{t})\lambda$ in $\bar{W}\lambda$ with a head of a rule $P(\bar{s})$, we can also unify $P(\bar{t})$ in $\bar{W}$ with $P(\bar{s})$. It follows that there is another successful path $(E; G \to answer(\bar{x}); \emptyset) \vdash^*_{LC'} (E; \bar{W} \to \bar{V}; \_) \vdash^*_{LC'} (E; answer(\bar{x})\theta' \to true; \_)$ which is obtained by applying to $\bar{W} \to \bar{V}$ the same clauses applied to $\bar{W}\lambda \to \bar{V}\lambda$ to get the solution. This path is shorter than α^*.

2. Let us assume now that H' reduces H to an equation containing only *answer* literals. It follows that H' has the form $\bar{W} \to answer(\_)$ and H is $\bar{W}\lambda \to answer(\_)$ or $\bar{W}\lambda answer(\_) \to \bar{W}\lambda$. There may be more than one *answer* literal in H' and H, but this is irrelevant. It follows that α^* is $(E; G \to answer(\bar{x}); \emptyset) \vdash^*_{LC'} (E; \bar{W} \to answer(\_); \_) \vdash^*_{LC'} (E; \bar{W}\lambda \to answer(\_); \_) \vdash^*_{LC'} (E; answer(\bar{x})\theta \to true; \_)$. By the same argument applied in the previous case, there is a shorter successful path $(E; G \to answer(\bar{x}); \emptyset) \vdash^*_{LC'} (E; \bar{W} \to answer(\_); \_) \vdash^*_{LC'} (E; answer(\bar{x})\theta' \to true; \_)$. □

We can finally state our completeness result:

Theorem 8.5.4 *Let E be an iff-program. If $E^* \models G\theta \simeq true$, i.e. θ is a correct answer for G, there exists a computed answer σ, $E \vdash_{LC} G\sigma$, such that $G\theta \leftrightarrow^*_{E^*} G\sigma\rho$ for some substitution ρ.*

Proof: if $E^* \models G\theta \simeq true$, then by completeness of LC' there exists a computed answer τ, $E \vdash_{LC'} G\tau$, such that $G\theta = G\tau\rho$. By Lemma 8.5.2, there exists an answer for the same query computed by LC: $E \vdash_{LC} G\sigma$. $E \vdash_{LC'} G\tau$ implies that $G\tau \leftrightarrow^*_{E^*} true$. Similarly, $E \vdash_{LC} G\sigma$ implies $G\sigma \leftrightarrow^*_{E^*} true$. It follows that $G\tau \leftrightarrow^*_{E^*} G\sigma$. Then $G\tau\rho \leftrightarrow^*_{E^*} G\sigma\rho$ or $G\theta \leftrightarrow^*_{E^*} G\sigma\rho$. □

Theorem 8.5.5 *If $E \equiv P$, if $P \vdash_{Prolog} G\theta$, then there exists an answer σ given by Linear Completion, $E \vdash_{LC} G\sigma$, such that $G\theta \leftrightarrow^*_{E^*} G\sigma\rho$ for some substitution ρ.*

Proof: if $P \vdash_{Prolog} G\theta$, then θ is a correct answer substitution by soundness of SLD-resolution. Then by Theorem 8.5.4 there exist a computed answer substitution σ, $E \vdash_{LC} G\sigma$ and a substitution ρ, such that $G\theta \leftrightarrow^*_{E^*} G\sigma\rho$. □

8.6 Comparisons with related work: some loop checking mechanisms in Prolog

Rewrite systems have been applied in both functional and logic programming and there is a vast literature on this subject. In this chapter we have restricted our attention to a special class of rewrite systems interpreted by Linear Completion. Our rewrite rules are *equivalences* between conjunctions of atoms. Therefore, our language is relational like pure Prolog. The inference system of Linear Completion includes simplification and overlap. An overlap step intuitively corresponds to a resolution step and therefore we have carefully avoided to call it *narrowing*. The name narrowing was introduced first in [75] and it refers to a paramodulation step by an oriented equation whose sides are terms. Our method is not related to those using narrowing since we do not use equations or rewrite rules between first order terms. It is closer instead to those given in [29, 30, 31, 73]. The approach proposed in the first three papers does not allow simplification, thus cannot fully utilize the power of rewriting. Our approach is similar to [73], although they did not study the case where a predicate is defined by implications and not by logical equivalences. Neither did they explain the semantics of their method.

One of the positive effects of simplification in executing rewrite programs is loop avoidance. Techniques for loop detection and avoidance in the execution of Prolog programs have received considerable attention. Here we focuse on the results in [7, 17] since their loop checking mechanisms are based on *subsumption of goals by ancestors* and therefore they may yield pruning effects similar to those of simplification.

Loop checking mechanisms are described in [7, 17] according to the following terminology: a loop checking mechanism L is said to be

- *complete*, if it prunes all infinite derivations,

- *sound*, if no answer substitution is lost, that is, for all answers σ to a query G generated by Prolog, there is an answer σ' generated by Prolog with L such that $G\sigma = G\sigma'\rho$ for some substitution ρ,
- *weakly sound*, if whenever there are answers generated by Prolog to a query G, there is at least one answer to G generated by Prolog with L.

Completeness is a very strong requirement and indeed it is proved in [7] that no weakly sound and complete loop check exists for general Prolog programs, even in the absence of function symbols.

Weak soundness does not seem to be a sufficiently significant property: a weakly sound loop check is guaranteed to yield at least one answer if there are any, but no information is given about how the generated substitution is possibly related to those which are not given. Our Theorem 8.5.5 instead, establishes more than weak soundness, since it says that for all Prolog answers σ to a query G, there exists an answer θ by Linear Completion such that $G\sigma \leftrightarrow^*_{E^*} G\theta\rho$ for some substitution ρ.

Three kinds of loop checking mechanisms are introduced in [7, 17]. The first one is called *Contains a Variant/Instance of Atom (CVA/CIA)* check. The basic idea is to interrupt a derivation if the current goal contains an atom which is subsumed by an atom occurring in an ancestor goal. As observed in [7, 17], the CVA/CIA check is not even weakly sound. Take the *append* examples which we presented in Section 8.1: Prolog with CVA/CIA would interrupt the infinite derivation of the first *append* example, but, unlike Linear Completion, it would also halt with no answer the computation for the query

$$?- append(X, [b|Y], [a, b, c|Z]), size(X) > 3$$

of the third example. This behaviour is not correct and it shows that a loop check mechanism which is not weakly sound is unacceptable. Intuitively, the CVA/CIA checks are not correct because they check single atoms out of their contexts: the condition to halt the derivation applies to the *append* literal without taking the *size* literal into consideration. On the other hand, Linear Completion answers the above query because simplification keeps the context into account, since it requires that the entire left hand side of the ancestor matches the current goal.

The second family of loop checks introduced in [7, 17] is obtained by replacing "atom" by "goal": a derivation is interrupted if the current goal is subsumed by one of its ancestors. These checks are called *Equal Variant of Goal (EVG)*, *Equal Instance of Goal (EIG)*, *Subsumed as Variant of Goal (SVG)* and *Subsumed as Instance of Goal (SIG)*, where "equal" means that the current goal is exactly an instance of one of its ancestors. These four loop checks are all proved to be weakly sound [7, 17] and they are complete for very restricted classes of programs defined in [7, 17]. Weak soundness just guarantees that at least one answer is generated. The following example from [7] shows that weak soundness is not satisfactory: given the program

$$p(a).$$
$$p(Y) :- p(Z).$$

and the query

$?- p(X).$,

Prolog generates first the answer $\{X \mapsto a\}$, then the answer $\{X \mapsto Y\}$ and then it loops forever. Prolog with any of the above weakly sound checks is able to find only the answer a as shown in Fig. 8.4, where the computation tree below $?- p(Z)$ is pruned because $p(Z)$ is subsumed by $p(X)$.

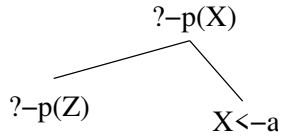


Figure 8.4: Prolog finds only answer a

If the current goal is subsumed by an ancestor, simplification applies. Therefore, one may expect that also Linear Completion loses all answers but $\{X \mapsto a\}$. The result is instead very different: the rewrite program

$p(a) \rightarrow true.$

$p(Y)p(Z) \rightarrow p(Z).$

with the query

$p(X) \rightarrow answer(X).$,

generates the answers $\{X \mapsto a\}$ and $\{X \mapsto X\}$ and then halts as shown in Fig. 8.5.

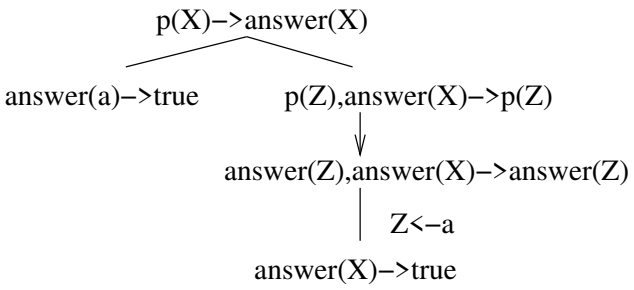


Figure 8.5: Linear completion generates $\{X \mapsto a\}$, $\{X \mapsto X\}$, and halts

Linear Completion correctly infers that $p(a)$ is true and that $p(X)$ is true for all X, since the second clause in the program says that if there exists an element Z such that $p(Z)$ is true, then $p(Y)$ is true for all Y. Operationally, the reason for this different behaviour is that simplification is not a mere pruning mechanism, but an additional inference rule: if a goal is simplified the computation does not halt, but continues with a reduced goal.

The third type of loop checking mechanisms in [7, 17] is based on *resultants*: if G_0 is the query, G_i is the current goal and σ_i is the partial answer substitution computed on the path from G_0 to G_i, the resultant associated to G_i is $G_0\sigma_i$. According to the loop checks based on resultants, a derivation is interrupted if the current goal and its associated resultant are susbumed respectively by an ancestor goal and its resultant. These loop checks are called *EVR, EIR, SVR* and *SIR*.

The resultant based loop checks are proved in [7, 17] to be sound in general and complete for very restricted classes of programs.

Loop checks based on variants rather than instances may be undesirably weak: for instance, the EVR and SVR checks would not prune the infinite derivation in our first *append* example, since the goals are variants, but the resultants are not. The EIR and SIR checks would prune it like Linear Completion.

The SIR loop check seems to be the one whose effects are the closest to those of simplification in Linear Completion. Intuitively, the reason for this similarity is that Linear Completion embeds some check on the resultants by recording partial answer substitutions in the *answer* literals. If the query has the form $G_0(\bar{x}) \rightarrow answer(\bar{x})$, the literal $answer(\bar{x}\sigma_i)$ in the goal G_i is exactly the resultant $G_0(\bar{x}\sigma_i)$. This is not the case if the query has the form $G_0(c[\bar{x}]) \rightarrow answer(\bar{x})$ for some context c, but such a query can be reformulated as $G_0(c[\bar{x}]) \rightarrow answer(c[\bar{x}])$, should it turn out to be convenient to carry more information in the *answer* literal. If an *answer* literal occurs in the left hand side of the ancestor applied as simplifier, simplification automatically checks resultants as well. This explains intuitively why in the following example from [17] the loop check whose behaviour is the closest to Linear Completion is SIR. Given the program

$a(0).$

$a(Y) :- a(0), c(Y).$

$b(1).$

$c(Z) :- b(Z), a(W).$

and the query $?- a(X).$,

the Prolog execution is pruned by the loop checks in [7, 17] as shown in Fig. 8.6, where each frame delimits the portion of the computation tree generated by Prolog augmented with the indicated loop check.

In particular, the SIR check halts the derivation after the answers $\{X \mapsto 0\}$ and $\{X \mapsto 1\}$ are derived because the goal $?- a(0), c(W)$ and its resultant $a(1)$ are respectively a variant and an instance of the ancestor $?- a(0), c(X)$ and its resultant $a(X)$. Linear Completion generates the two answers $\{X \mapsto 0\}$ and $\{X \mapsto 1\}$ and then it halts as shown in Fig. 8.7.

The SIG/SVG and EIG/EVG loop checks halt the computation too early and inhibit the generation of the answer $\{X \mapsto 1\}$. The SVR, EIR and EVR loop checks give both solutions like Linear Completion and SIR, but they are less efficient, since they prune the computation much later. It is remarkable that Linear Completion cuts out of the Prolog tree exactly those paths which lead to a solution.

To summarize, any comparison between Linear Completion and Prolog augmented with these loop checks is limited by the observation that the two approaches are very different in nature. The approach in [7, 17] consists in limiting the inference system of Prolog by adding loop checks. The concept of a loop check is purely procedural: the inference system is not modified, but just restricted by the addition of an external control. This approach seems therefore somewhat artificial.

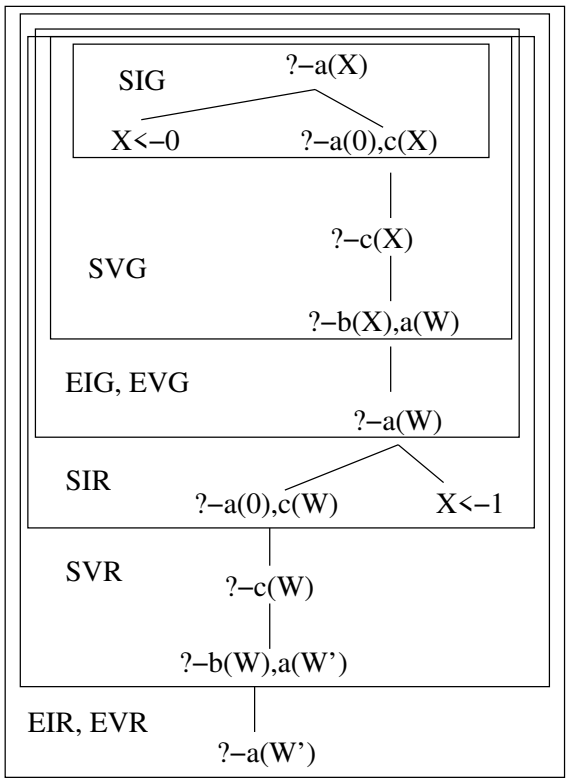


Figure 8.6: Loop checks prune the Prolog execution

In our approach, simplification is a natural consequence of writing program units as equations. Simplification is not a pruning mechanism added to the inference rules, but an inference rule itself, with the capability of reducing the search space. Therefore, if simplification applies, the derivation is not interrupted, but it continues with the reduced goal. For instance, in our basic *append* example, Prolog with EIR or SIR does not loop, because of the external loop checking mechanism, whereas Linear Completion does not loop because *append* is defined by equivalences rather than by implications. In all the examples proposed in [7, 17], Linear Completion behaves as desired.

Furthermore, simplification in Linear Completion uses uniformly program rules, answer rules and ancestor goals as simplifiers and it allows us to optimize executions where no infinite derivation occurs, as shown by the *ancestor* example in Section 8.1.

$a(0) \rightarrow true.$
$a(Y), a(0), c(Y) \rightarrow a(0), c(Y).$
$b(1) \rightarrow true.$
$c(Z) \rightarrow b(Z), a(W).$
$a(X) \rightarrow answer(X).$

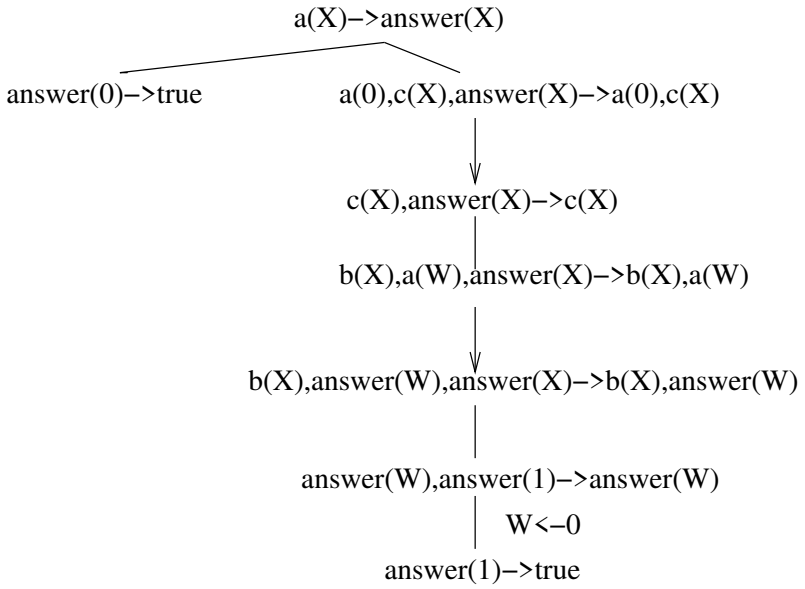


Figure 8.7: Linear completion generates $\{X \mapsto 0\}$, $\{X \mapsto 1\}$, and halts

Chapter 9

Summary and directions for future research

We have given a new abstract framework for the study of Knuth-Bendix type completion procedures, where all the fundamental concepts are uniformly defined in terms of *proof reduction* with respect to a well founded proof ordering. In order to do this, we have given a new, more general notion of proof ordering, such that proofs of different theorem can also be compared.

A completion procedure is given by a set of *inference rules* and a *search plan*. We have emphasized the distinction between these two components throughout our work. This distinction is often overlooked in the literature, where most theorem proving strategies are presented by giving the set of inference rules only, whereas the search plan is what ultimately turns a set of inference rules into a procedure.

Inference rules are either *expansion* inference rules or *contraction* inference rules. Each inference step either reduces some proofs or eliminates *redundant* sentences. We have compared our expansion/contraction scheme with the deduction/deletion scheme in [16] and our notion of redundancy with those in [74, 16]. Our scheme deals more satisfactorily with contraction rules and our notion of redundancy seems closer to the intuitive notion of redundancy.

If the inference rules are *refutationally complete* and the search plan is *fair*, a completion procedure is a semidecision procedure for theorem proving. The key part of this result is the notion of *fairness*. Our definition of fairness is the first definition of fairness for completion procedures which addresses the theorem proving problem. It is new in three ways: it is *target-oriented*, that is it keeps the theorem to be proved into consideration, it is explicitly stated as a property of the search plan and it is defined in terms of proof reduction, so that expansion inferences and contraction inferences are treated uniformly.

If the search plan of a completion procedure is *uniformly fair*, the limit of a derivation by completion is *saturated*. If additional conditions are satisfied, a saturated presentation is a *decision procedure* and therefore completion acts as a *generator of decision procedures*. Saturation and semidecision are related by our *general Knuth-Bendix-Huet theorem*, which is the first generalization of the classical result for equational logic. Previous works on saturated presentations in Horn logic with equality [63, 16] failed to generalize the Knuth-Bendix-Huet theorem, because

they did not relate the saturation process with the semidecision process and regarded saturation as a compilation process.

We have presented some equational completion procedures based on Unfailing Knuth-Bendix completion, which include the AC-UKB procedure with Cancellation laws, the S-strategy and the Inequality Ordered Saturation strategy. These extensions of UKB had not been presented in a unified framework for completion before.

Two more applications of completion are disproving of inductive conjectures and logic programming. For the first one, we have shown that the so called *inductionless induction* method is a semidecision process. For the second one, we have given an operational and denotational semantics of rewrite programs interpreted by *Linear Completion*. Rewrite programs seem to capture the intended meaning of mutually exclusively defined predicates more accurately than Prolog programs. Also, simplification allows to take advantage of these definitions to prevent certain infinite loops which are otherwise unavoidable in pure Prolog.

The research reported in this report can be further pursued in several directions. We have developed this framework with the intent of applying it to the design of new, more efficient, completion procedures. This goal leads to the study of new, more powerful contraction inference rules and of fair and efficient search plans.

We are interested in efficient completion procedures for theorem proving both in the special case of pure equational logic and in the general case of full first order logic with equality. Therefore we expect to achieve a full extension of our approach to completion procedures for Horn and first order logic with equality. This includes the study of specific proof orderings for first order logic with equality and further work on the notions of saturated set, linear input reducing derivation and failure in first order logic with equality.

Finally, our view of completion as problem transformation might be extended to a notion of *completion modulo a decision algorithm*. If a decision algorithm is known for a certain theory, completion of a theorem proving problem $(S;\varphi)$ does not need to proceed up to a stage $(S_k;\varphi_k)$ where $S_k \models \varphi_k$ is trivial, such as when φ_k is an equation $s \simeq s$. The process halts at stage k if $S_k \models \varphi_k$ is decidable by the given algorithm. This approach opens the way to the study of completion in connection with decision procedures for special theories.

Bibliography

[1] S.Anantharaman and J.Mzali, Unfailing Completion modulo a set of equations, Technical Report, LRI, Université de Paris Sud, 1989.

[2] S.Anantharaman, J.Hsiang and J.Mzali, SbReve2: A Term Rewriting Laboratory with (AC)-Unfailing Completion, in N.Dershowitz (ed.), *Proceedings of the Third International Conference on Rewriting Techniques and Applications*, Chapel Hill, NC, USA, April 1989, Springer Verlag, Lecture Notes in Computer Science 355, 533–537, 1989.

[3] S.Anantharaman and J.Hsiang, Automated Proofs of the Moufang Identities in Alternative Rings, *Journal of Automated Reasoning*, Vol. 6, No. 1, 76–109, 1990.

[4] S.Anantharaman and N.Andrianarivelo, Heuristical Criteria in Refutational Theorem Proving, in A.Miola (ed.), *Proceedings of the Symposium on the Design and Implementation of Systems for Symbolic Computation*, 184–193, Capri, Italy, April 1990.

[5] S.Anantharaman, N.Andrianarivelo, M.P.Bonacina and J.Hsiang, SBR3: A Refutational Prover for Equational Theorems, to appear in M.Okada, S.Kaplan (eds.), *Proceedings of the Second International Workshop on Conditional and Typed Rewriting Systems*, Montreal, Canada, June 1990.

[6] K.R.Apt and M.H.Van Emden, Contributions to the Theory of Logic Programming, *Journal of the ACM*, Vol. 29, No. 3, 841–862, July 1982.

[7] K.R.Apt, R.N.Bol and J.W.Klop, On the Safe Termination of PROLOG programs, in G.Levi, M.Martelli (eds.), *Proceedings of the Sixth International Conference on Logic Programming*, 353–368, MIT Press, Cambridge MA, 1989.

[8] L.Bachmair, N.Dershowitz and J.Hsiang, Orderings for Equational Proofs, in *Proceedings of the First Annual IEEE Symposium on Logic in Computer Science*, 346–357, Cambridge, MA, June 1986.

[9] L.Bachmair and N.Dershowitz, Completion for rewriting modulo a congruence, in P.Lescanne (ed.), *Proceedings of the Second International Conference on Rewriting Techniques and Applications*, Bordeaux, France, May 1987, Springer Verlag, Lecture Notes in Computer Science 256, 192–203, 1987.

[10] L.Bachmair and N.Dershowitz, Inference Rules for Rewrite-Based First-Order Theorem Proving, in *Proceedings of the Second Annual Symposium on Logic in Computer Science*, Ithaca, New York, June 1987.

[11] L.Bachmair, Proofs Methods for Equational Theories, Ph.D. thesis, Department of Computer Science, University of Illinois, Urbana, IL.,1987.

[12] L.Bachmair and N.Dershowitz, Critical Pair Criteria for Completion, *Journal of Symbolic Computation*, Vol. 6, 1–18, 1988.

[13] L.Bachmair, Proof by consistency in equational theories, in *Proceedings of the Third Annual IEEE Symposium on Logic in Computer Science*, 228–233, Edinburgh, Scotland, July 1988.

[14] L.Bachmair, N.Dershowitz and D.A.Plaisted, Completion without failure, in H.Ait-Kaci, M.Nivat (eds.), *Resolution of Equations in Algebraic Structures*, Vol. II: Rewriting Techniques, 1–30, Academic Press, New York, 1989.

[15] L.Bachmair and H.Ganzinger, On Restrictions of Ordered Paramodulation with Simplification, in *Proceedings of the Tenth International Conference on Automated Deduction*, Kaiserslautern, Germany, July 1990.

[16] L.Bachmair and H.Ganzinger, Completion of First-Order Clauses with Equality by Strict Superposition, to appear in M.Okada, S.Kaplan (eds.), *Proceedings of the Second International Workshop on Conditional and Typed Rewriting Systems*, Montreal, Canada, June 1990.

[17] R.N.Bol, K.R.Apt and J.W.Klop, On the Power of Subsumption and Context Checks, in A.Miola ed., *Proceedings of the International Symposium on Design and Implementation of Symbolic Computation Systems*, 131–140, Capri, Italy, April 1990.

[18] M.P.Bonacina and G.Sanna, KBlab: An Equational Theorem Prover for the Macintosh, in N.Dershowitz (ed.), *Proceedings of the Third International Conference on Rewriting Techniques and Applications*, Chapel Hill, NC, USA, April 1989, Springer Verlag, Lecture Notes in Computer Science 355, 548–550, 1989.

[19] M.P.Bonacina and J.Hsiang, Completion procedures as Semidecision procedures, to appear in M.Okada, S.Kaplan (eds.), *Proceedings of the Second International Workshop on Conditional and Typed Rewriting Systems*, Montreal, Canada, June 1990.

[20] M.P.Bonacina and J.Hsiang, Operational and Denotational Semantics of Rewrite Programs, in S.Debray, M.Hermenegildo (eds.), *Proceedings of the North American Conference on Logic Programming*, Austin, TX, October 1990, MIT Press, 449–464, 1990.

[21] M.P.Bonacina and J.Hsiang, On fairness of completion-based theorem proving strategies, to appear in R.V.Book (ed.), *Proceedings of the Fourth International Conference on Rewriting Techniques and Applications*, Como, Italy, April 1991.

[22] M.P.Bonacina and J.Hsiang, The Knuth-Bendix-Huet theorem and its extensions, in preparation.

[23] R.M.Burstall, D.B.MacQueen and D.T.Sannella, HOPE: an experimental applicative language, in *Conference Record of the 1980 LISP Conference*, 136–143, Stanford, California, 1980.

[24] C.L.Chang and R.C.Lee, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, New York, 1973.

[25] S.K.Debray and D.S.Warren, Functional Computations in Logic Programs, *ACM Transactions on Programming Languages and Systems*, Vol. 11, No. 3, 451–481, July 1989.

[26] N.Dershowitz and Z.Manna, Proving termination with multisets orderings, *Communications of the ACM*, Vol. 22, No. 8, 465–476, August 1979.

[27] N.Dershowitz, Orderings for term-rewriting systems, *Theoretical Computer Science*, Vol. 17, No. 3, 279–301, 1982.

[28] N.Dershowitz, J.Hsiang, N.A.Josephson and D.A.Plaisted, Associative-commutative rewriting, in A.Bundy ed., *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, 940–944, Karlsrue, Germany, 1983.

[29] N.Dershowitz and N.A.Josephson, Logic Programming by Completion, in *Proceedings of the Second International Conference on Logic Programming*, 313–320, Uppsala, Sweden, 1984.

[30] N.Dershowitz, Computing with Rewrite Systems, *Information and Control*, Vol. 65, 122–157, 1985.

[31] N.Dershowitz and D.A.Plaisted, Logic Programming Cum Applicative Programming, in *Proceedings of the IEEE Symposium on Logic Programming*, 54–66, Boston, MA, 1985.

[32] N.Dershowitz, Termination of Rewriting, *Journal of Symbolic Computation*, Vol. 3, No. 1 & 2, 69–116, February/April 1987.

[33] N.Dershowitz, Completion and its Applications, in *Proceedings of Conference on Resolution of Equations in Algebraic Structures*, Lakeway, Texas, May 1987.

[34] N.Dershowitz and J.-P.Jouannaud, Rewrite Systems, Chapter 15 of Volume B of *Handbook of Theoretical Computer Science*, North-Holland, 1989.

[35] N.Dershowitz and J.-P.Jouannaud, Notations for Rewriting, Rapport de Recherche 478, LRI, Université de Paris Sud, January 1990.

[36] N.Dershowitz, A Maximal-Literal Unit Strategy for Horn Clauses, to appear in M.Okada, S.Kaplan (eds.), *Proceedings of the Second International Workshop on Conditional and Typed Rewriting Systems*, Montreal, Canada, June 1990.

[37] F.Fages, Associative-commutative unification, in R.Shostak (ed.), *Proceedings of the Seventh International Conference on Automated Deduction*, Napa Valley, CA, USA, 1984, Springer Verlag, Lecture Notes in Computer Science 170, 1984.

[38] L.Fribourg, A Strong Restriction to the Inductive Completion Procedure, in *Proceedins of the Thirteenth International Conference on Automata Languages and Programming*, Rennes, France, July 1986, Springer Verlag, Lecture Notes in Computer Science 226, 1986.

[39] K.Futatsugi, J.A.Goguen, J.P.Jouannaud and J.Meseguer, Principles of OBJ2, in *Conference Record of the 12th Annual ACM Symposium on Principles of Programming Languages*, New Orleans, LA, 1985.

[40] J.A.Goguen, How to prove algebraic inductive hypotheses without induction, in W.Bibel and R.Kowalski (eds.), *Proceedings of the Fifth International Conference on Automated Deduction*, 356–373, Les Arcs, France, 1980, Springer Verlag, Lecture Notes in Computer Science 87, 1980.

[41] J.A.Goguen and J.Meseguer, Equality, types, modules and (why not?) generics for logic programming, *Journal of Logic Programming*, Vol. 1, No. 2, 179–210, 1984.

[42] C.C.Green, The Application of Theorem-proving to Question-answering, Ph.D. Thesis, Dept. of Computer Science, Stanford University, Stanford, California, 1969.

[43] C.M.Hoffmann and M.J.O'Donnell, Programming with Equations, *ACM Transactions on Programming Languages and Systems*, Vol. 4. No. 1, 83–112, January 1982.

[44] J.Hsiang and N.Dershowitz, Rewrite Methods for Clausal and Nonclausal Theorem Proving, in *Proceedings of the Tenth International Conference on Automata, Languages and Programming*, Barcelona, Spain, July 1983, Springer Verlag, Lecture Notes in Computer Science 154, 1983.

[45] J.Hsiang, Refutational Theorem Proving Using Term Rewriting Systems, *Artificial Intelligence*, Vol. 25, 255–300, 1985.

[46] J.Hsiang and M.Rusinowitch, A New Method for Establishing Refutational Completeness in Theorem Proving, in J.Siekmann (ed.), *Proceedings of the Eighth Conference on Automated Deduction*, Oxford, England, July 1986, Springer Verlag, Lecture Notes in Computer Science 230, 141–152, 1986.

[47] J.Hsiang, Rewrite Method for Theorem Proving in First Order Theories with Equality, *Journal of Symbolic Computation*, Vol. 3, 133–151, 1987.

[48] J.Hsiang and M.Rusinowitch, On word problems in equational theories, in Th.Ottman (ed.), *Proceedings of the Fourteenth International Conference on Automata, Languages and Programming*, Karlsruhe, Germany, July 1987, Springer Verlag, Lecture Notes in Computer Science 267, 54–71, 1987.

[49] J.Hsiang, M.Rusinowitch and K.Sakai, Complete Inference Rules for the Cancellation Laws, in *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milano, Italy, August 1987, 990–992, 1987.

[50] J.Hsiang and M.Rusinowitch, Proving Refutational Completeness of Theorem Proving Strategies: the Transfinite Semantic Tree Method, to appear in *Journal of the ACM*, 1990.

[51] G.Huet, Confluent reductions: abstract properties and applications to term rewriting systems, *Journal of the ACM*, Vol. 27, 797–821, 1980.

[52] G.Huet, A Complete Proof of Correctness of the Knuth-Bendix Completion Algorithm, *Journal of Computer and System Sciences*, Vol. 23, 11–21, 1981.

[53] G.Huet and J.M.Hullot, Proofs by Induction in Equational Theories with Constructors, Journal of Computer and System Sciences, Vol. 25, 239–266, 1982.

[54] J.-P.Jouannaud and E.Kounalis, Proofs by induction in equational theories without constructors, in *Proceedings of the First Annual IEEE Symposium on Logic in Computer Science*, 358–366, Cambridge, MA, June 1986.

[55] J.-P.Jouannaud and H.Kirchner, Completion of a set of rules modulo a set of equations, *SIAM Journal of Computing*, Vol. 15, 1155–1194, November 1986.

[56] J.-P.Jouannaud and E.Kounalis, Automatic proofs by induction in equational theories without constructors, *Information and Computation*, 1989.

[57] J.-P.Jouannaud and C.Kirchner, Solving Equations in Abstract Algebras: A Rule-Based Survey of Unification, Rapport de Recherche, LRI, Université de Paris Sud, November 1989.

[58] S.Kamin and J.-J.Lévy, Two generalizations of the recursive path ordering, Unpublished note, Department of Computer Science, University of Illinois, Urbana, Illinois, February 1980.

[59] D.Kapur and P.Narendran, An equational approach to theorem proving in first order predicate calculus, in *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 1146–1153, Los Angeles, CA, August 1985.

[60] D.Kapur and D.R.Musser, Proof by consistency, *Artificial Intelligence*, Vol. 31, No. 2, 125–157, February 1987.

[61] D.Kapur, P.Narendran and H.Zhang, Proof by induction using test sets, in J.Siekmann (ed.), *Proceedings of the Eighth Conference on Automated Deduction*, Oxford, England, July 1986, Springer Verlag, Lecture Notes in Computer Science 230, 99–117, 1986.

[62] D.E.Knuth and P.B.Bendix, Simple Word Problems in Universal Algebras, in J.Leech (ed.), *Proceedings of the Conference on Computational Problems in Abstract Algebras*, Oxford, England, 1967, Pergamon Press, Oxford, 263–298, 1970.

[63] E.Kounalis and M.Rusinowitch, On Word Problems in Horn Theories, in E.Lusk, R.Overbeek (eds.), *Proceedings of the Ninth International Conference on Automated Deduction*, 527–537, Argonne, Illinois, May 1988, Springer Verlag, Lecture Notes in Computer Science 310, 1988.

[64] R.A.Kowalski and M.H.Van Emden, Predicate Logic as a Programming Language, *Journal of the ACM*, Vol. 4, No. 23, 1976.

[65] D.S.Lankford, Canonical inference, Memo ATP-32, Automatic Theorem Proving Project, University of Texas, Austin, TX, May 1975.

[66] D.S.Lankford, A simple explanation of inductionless induction, Technical report MTP-14, Mathematics Department, Louisiana Technical University, Ruston, LA, 1981.

[67] J.W.Lloyd, *Foundations of Logic Programming*, second edition, Springer Verlag, Berlin, 1987.

[68] D.Musser, On proving inductive properties of abstract data types, in *Proceedings of the Seventh ACM Symposium on Principles of Programming Languages*, 154–162, Las Vegas, Nevada, 1980.

[69] A.Ohsuga and K.Sakai, Refutational Theorem Proving for Equational Systems Based on Term Rewriting, Technical Report COMP86–40, ICOT, 1986.

[70] G.E.Peterson and M.E.Stickel, Complete sets of reductions for some equational theories, *Journal of the ACM*, Vol. 28, No. 2, 233–264, 1981.

[71] G.E.Peterson, A Technique for Establishing Completeness Results in Theorem proving with Equality, *SIAM Journal of Computing*, Vol. 12, No. 1, 82–100, 1983.

[72] D.A.Plaisted, Semantic confluence tests and completion methods, *Information and Control*, Vol. 65, 182–215, 1985.

[73] P.Rety, C.Kirchner, H.Kirchner and P.Lescanne, NARROWER: a new algorithm for unification and its application to logic programming, in J.P.Jouannaud ed., *Proceedings of the First International Conference on Rewrite Techniques and Applications*, Dijon, France, May 1985.

[74] M.Rusinowitch, Theorem-proving with resolution and superposition: an extension of Knuth and Bendix procedure as a complete set of inference rules, Thèse d'Etat, Université de Nancy, 1987.

[75] J.R.Slagle, Automated theorem proving with simplifiers, commutativity and associativity, *Journal of the ACM*, Vol. 21, 622–642, 1974.

[76] M.E.Stickel, Unification Algorithms for Artificial Intelligence Languages, Ph.D. thesis, Carnegie Mellon University, 1976.

[77] M.E.Stickel, A unification algorithm for associative-commutative functions, *Journal of the ACM*, Vol. 28, No. 3, 423–434, 1981.

[78] H.Zhang and D.Kapur, First Order Theorem Proving Using Conditional Rewrite RUles, in E.Lusk, R.Overbeek (eds.), *Proceedings of the Ninth International Conference on Automated Deduction*, 1–20, Argonne, Illinois, May 1988, Springer Verlag, Lecture Notes in Computer Science 310, 1988.