Ordered Completion with Selection for Horn Clauses

Peter Stadelmeyer

Linz, November 1995

Contents

Chapter 1. Introduction 1

Chapter 2. Terminology and Classical Results 3
1. Syntax 3
2. Orderings 8
3. Equational Theories 10
4. Rewrite Systems 12
5. Semantics 14

Chapter 3. Standard Completion 17
1. Standard Completion Derivations 17
2. Proof Orderings 19
3. The Inference System $\mathcal{EC}$ for Equational Completion 22
4. The Inference System $\mathcal{CC}$ for Conditional Completion 26

Chapter 4. Ordered Completion 33
1. Equation and Clause orderings 36
2. The Inference System $\mathcal{OC}_S$ for Ordered Completion 37
3. Refutational Completeness of $\mathcal{OC}_S$ 39
4. Ordered Completion Derivations 45
5. Simplification and Deletion Techniques 46
6. Convergence 48

Chapter 5. Examples 51
1. Two Equational Examples 51
2. The Natural Numbers with $\leq$ 56
3. The Maximum Function 62
4. Lists 66
5. Refutational Theorem Proving 70

Appendix A. About **HCC** 77
1. Commands 77
2. Options 78

Appendix B. Program Listing of **HCC** 81

Bibliography 83

CHAPTER 1

Introduction

Term rewriting is one of the most successful techniques in reasoning about equational theories. It is based on a completion procedure, the Knuth–Bendix procedure [**Knuth and Bendix, 1970**], which tries to generate from a given set of equations a set of convergent[1] rewrite rules. Once, having such a set of rules, it can be used as a decision procedure for proving the validity of equations in the theory.

One may ask, if this method can be generalized to conditional equations or even to Horn clauses. In 1987 Kaplan [**Kaplan, 1987**] and Jouannaud, Waldman [**Jouannaud and Waldmann, 1987**] presented the first completion procedures for simplifying conditional rewrite systems. Unfortunately these procedures tend to fail much more often in constructing a convergent rewrite system than the Knuth–Bendix procedure. Ganzinger proposed in [**Ganzinger, 1991**] a completion procedure which is able to handle also non-simplifying conditional equations to a certain extent.

Combining the ideas of ordered completion and Paramodulation Bachmair, Ganzinger in [**Bachmair and Ganzinger, 1990a**] and Kounalis, Rusinowitch in [**Kounalis and Rusinowitch, 1991**] presented ordered (unfailing) completion procedures for Horn clauses. In addition, these procedures are refutationally complete and therefore can be used as a theorem prover.

In this book we describe completion procedures of all three types in a uniform way by inference systems called $\mathcal{EC}$ (standard equational completion), $\mathcal{CC}$ (standard conditional completion) and $\mathcal{OC}$ (ordered Horn clause completion). The correctness of each inference system is proved in its typical way. That is, the inference systems $\mathcal{EC}$ and $\mathcal{CC}$ are shown to be correct with the help of proof orderings, whereas the correctness of the inference system $\mathcal{OC}$ is proved using clause orderings. In order to outline the similarities between $\mathcal{EC}$ and $\mathcal{CC}$ the proof orderings are chosen in such a way that the proofs of $\mathcal{CC}$ can be seen as straight forward generalizations of those of $\mathcal{EC}$.

For practical experimentation the inference system $\mathcal{OC}$ together with some simplification and deletion techniques has been implemented in the program **HCC** [2], written in Common Lisp.

The terminology we use is basically the same as in the "Handbook of Theoretical Computer Science" [**Dershowitz and Jouannaud, 1990**] or in [**Bachmair, 1991**].

[1] By convergent we mean terminating and Church–Rosser.

[2] **HCC** stands for Horn Clause Completion. The overall structure of **HCC** was inspired by the Rewrite Rule Laboratory RRL [**Kapur and Zhang, 1989**], although they do not have much in common.

CHAPTER 2

Terminology and Classical Results

In the following we summarize the terminology used throughout this book and present some well known results in the field of term rewriting. All definitions and propositions are given for the case of *single-sorted* functions.

A few symbols will have a unique meaning throughout the book:

s, t, u, v, w	terms
σ, τ	substitutions
E	a set of (conditional) equations
R	a set of (conditional) rules
Γ, Δ, Φ, Ψ	multisets of equations
C, D	clauses
N	a set of clauses

1. Syntax

Let $\mathcal{F}$ be a countable set of *function symbols*. With each $f \in \mathcal{F}$ we associate a non-negative number, called its *arity*. Function symbols of arity zero are called *constants*, of which we make the assumption that there exists at least one. Further, let $\mathcal{V}$ be a countable infinite set of *variables* which is disjoint from $\mathcal{F}$.

1.1. Terms.

DEFINITION 2.1. *A* term *is either a variable or an expression of the form $f\,t_1 \ldots t_n$, where f is a function symbol of arity n and t_1,..., t_n are terms. By $\mathcal{T}(\mathcal{F}, \mathcal{V})$ we denote the set of all terms built from function symbols of $\mathcal{F}$ and variables of $\mathcal{V}$. The set consisting of all variables occuring in a term t is abbreviated by* $\mathrm{var}(t)$. *A term t is called a* ground term *if* $\mathrm{var}(t) = \emptyset$*, i.e. t is a variable-free term. The set of all ground terms built from function symbols of $\mathcal{F}$ is called the* Herbrand universe of $\mathcal{F}$ *and denoted by $\mathcal{T}(\mathcal{F})$.*

For readability reasons we will often use parentheses and infix notation for terms, e.g. the term $+\,x + \,inv\,xy$ will be written as $x + (inv(x) + y)$.

DEFINITION 2.2. *A term s is said to be a* subterm *of a term t, written as $t[s]$, if either $s = t$, or $t = f\,t_1 \ldots t_n$ and s is a subterm of some t_i, $1 \le i \le n$. We call s a* proper subterm *of t, if s is a subterm of t and $s \neq t$.*

DEFINITION 2.3. *The set of* positions *of a term t is denoted by* $\mathrm{pos}(t)$ *and recursively defined as:*

$$\mathrm{pos}(t) = \begin{cases} \{\lambda\} & \text{, if } t \in \mathcal{V} \text{ or } t \text{ is a constant;} \\ \{\lambda\} \cup \{i.p \mid p \in \mathrm{pos}(t_i), 1 \le i \le n\} & \text{, if } t = f\,t_1 \ldots t_n. \end{cases}$$

A position p is said to be below *a position q (or: q is* above *p) if $p = q.q'$, for some position q'. If $q' \neq \lambda$ then p is* strictly below *q (or: q is* strictly above *p). The positions p and q are said to be* disjoint *if none of them is below the other one.*

DEFINITION 2.4. *Let t be a term and p a position in t. By the* subterm of t at position p, *written as $t|_p$, we mean the term:*

$$t|_p = \begin{cases} t & \text{, if } p = \lambda; \\ t_i|_q & \text{, if } p = i.q \text{ and } t = f\,t_1 \ldots t_n. \end{cases}$$

For example, let $t = x + (inv(x) + y)$. The set of positions of t is $\mathrm{pos}(t) = \{\lambda, 1, 2, 2.1, 2.1.1, 2.2\}$ and the subterm at position 2.1 is $t|_{2.1} = inv(x)$.

DEFINITION 2.5. *Let s and t be terms and let p be a position in t. The new term obtained from t by* replacing its subterm at position p by the term *s is denoted by $t[s]_p$ and defined as:*

$$t[s]_p = \begin{cases} s & \text{, if } p = \lambda; \\ f\,t_1 \ldots t_{i-1} t_i[s]_q t_{i+1} \ldots t_n & \text{, if } p = i.q \text{ and } t = f\,t_1 \ldots t_n. \end{cases}$$

Sometimes we will use phrases like "replace $t[v]$ by $t[w]$" by which we mean that $t[w]$ is the (ambiguous) result of replacing the subterm v of t at some position p by the term w.

1.2. Substitutions.

DEFINITION 2.6. *A* substitution *σ is a mapping from $\mathcal{V}$ to $\mathcal{T}(\mathcal{F}, \mathcal{V})$ with $x\sigma = x$, for almost all $x \in \mathcal{V}$, where $x\sigma$ denotes the result of applying σ to x.*

By $\sigma = \{x_1 \mapsto t_n, \ldots, x_n \mapsto t_n\}$ we represent the substitution σ for which $x\sigma = t_i$, if $x = x_i$, for some $1 \leq i \leq n$, and $x\sigma = x$, otherwise. The empty substitution $\{\}$ will be abbreviated by ε.

The composition of two substitutions σ and τ, written as $\sigma\tau$, is defined as $x\sigma\tau = (x\sigma)\tau$, for all $x \in \mathcal{V}$. Setting $(f\,t_1 \ldots t_n)\sigma$ to $f\,t_1\sigma \ldots t_n\sigma$ extends substitutions uniquely to a mapping from $\mathcal{T}(\mathcal{F}, \mathcal{V})$ to $\mathcal{T}(\mathcal{F}, \mathcal{V})$.

DEFINITION 2.7. *A term s is called an* instance *of a term t (or: t* subsumes *s) if there exists a substitution σ such that $s = t\sigma$. If t is also an instance of s the two terms are called* literally similar, *i.e. they are equal up to renaming of variables. Otherwise, we say that s is a* proper instance *of t (or: t* properly subsumes *s).*

We call an instance s of a term t a ground instance *if s is a ground term.*

The relation "t is instance of s" is a quasi-ordering[1] on terms. It is a special case of the *encompassment ordering* $\trianglerighteq$, which is also a quasi-ordering on terms and is defined as: $t \trianglerighteq s$ if and only if some subterm of t is an instance of s.

By $\triangleright$ we denote the strict part of $\trianglerighteq$ (i.e. $t \triangleright s$ if and only if $t \trianglerighteq s$ and $s \not\trianglerighteq t$).

[1]A *quasi-ordering* is a binary reflexive, transitive relation. See Definition 2.18.

DEFINITION 2.8. *Two terms s and t are said to be* unifiable *if there exists a substitution σ, called a* unifier, *such that $s\sigma = t\sigma$.*

Let s and t be two unifiable terms. A unifier τ of s and t is called a most general unifier of s and t *if any other unifier σ of s and t can be written as $\sigma = \tau\sigma'$, where σ' is a substitution.*

If two terms s and t are unifiable then there exists a unique (up to renaming of variables) most general unifier denoted by $\mathrm{mgu}(s,t)$[2].

The first algorithm for computing the most general unifier of two terms was described by Robinson in [**Robinson, 1965**]. In 1982 Martelli and Montanari presented an improved unification algorithm which is nearly linear in time. They view unification as a step-by-step process of transforming multisets of equations into a "solved form" from which the most general unifier can easily be extracted. For more details about their algorithm see [**Martelli and Montanari, 1982**].

Let $s = inv(1) + x$ and $t = y + inv(z)$ be two terms. The substitution $\sigma = \{x \mapsto inv(1), y \mapsto inv(1), z \mapsto 1\}$ is a unifier of s and t. The terms s and t have $\tau = \mathrm{mgu}(s,t)$ is $\tau = \{x \mapsto inv(z), y \mapsto inv(1)\}$ as their most general unifier and, hence, σ can be written as $\tau\sigma'$ with $\sigma' = \{z \mapsto 1\}$.

1.3. Rewrite Relations.

DEFINITION 2.9. *A binary relation $\to$ on $\mathcal{T}(\mathcal{F}, \mathcal{V})$ is called a* rewrite relation *if the following two properties hold:*

(i) *$s \to t$ implies $u[s]_p \to u[t]_p$, for all terms s, t, u and positions p in u.*
(ii) *$s \to t$ implies $s\sigma \to t\sigma$, for all terms s, t and substitutions σ.*

In other words, rewrite relations are relations which are closed under context application ("replacement property") and substitution ("fully invariant property").

[2] The decidability and uniqueness result of the most general unifier for first-order terms does not hold for higher-order terms with function variables (see [**Huet, 1976**]).

DEFINITION 2.10. *For any binary relation $\rightarrow$ we can define the following:*

$\leftarrow$	*inverse*	$\rightarrow^{-1}$
$\leftrightarrow$	*symmetric closure*	$\rightarrow \cup \leftarrow$
$\rightarrow^{+}$	*transitive closure*	$\rightarrow \cdots \rightarrow$
$\rightarrow^{*}$	*transitive-reflexive closure*	$= \cup \rightarrow^{+}$
$\leftrightarrow^{*}$	*symmetric-transitive-reflexive closure*	$(\leftrightarrow)^{*}$

DEFINITION 2.11. *Let $\rightarrow$ be a rewrite relation.*

(i) *If there is no term t such that $s \rightarrow t$, the term s is called* irreducible.
(ii) *A term t is called a* normal form *of s, $s \rightarrow^{!} t$, if $s \rightarrow^{*} t$ and t is irreducible.*
(iii) *Two terms s and t are called* joinable, *written as $s \downarrow^{*} t$, if there exists a term u such that $s \rightarrow^{*} u$ and $t \rightarrow^{*} u$.*
(iv) *Two terms s and t are called* normal-joinable, *written as $s \downarrow^{!} t$, if there exists an irreducible term u such that $s \rightarrow^{!} u$ and $t \rightarrow^{!} u$.*
(v) *The rewrite relation $\rightarrow$ is said to be* terminating *or* well-founded *if there exists no infinite rewrite sequence $t_0 \rightarrow t_1 \rightarrow \cdots$.*
(vi) *If $s \leftrightarrow^{*} t$ implies $s \downarrow^{*} t$, for all $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, the rewrite relation $\rightarrow$ is said to be* Church–Rosser *(or: $\rightarrow$ has the Church–Rosser property).*
(vii) *If $\rightarrow$ is Church–Rosser and terminating it is called* convergent.[3]

It can easily be shown from the above definitions that convergent rewrite relations define *unique* normal-forms.

DEFINITION 2.12. *Let $\rightarrow$ be a convergent rewrite relation. We define the* normal-form operator $\mathrm{nf}_{\rightarrow}()$ of $\rightarrow$ *as*

$$\mathrm{nf}_{\rightarrow}(s) = t \quad , \textit{where } s \rightarrow^{!} t.$$

LEMMA 2.13. *Let $\rightarrow$ be a convergent rewrite relation and s and t be terms. The following are equivalent:*

(i) $s \leftrightarrow^{*} t$
(ii) $s \downarrow^{*} t$
(iii) $s \downarrow^{!} t$
(iv) $\mathrm{nf}_{\rightarrow}(s) = \mathrm{nf}_{\rightarrow}(t)$

PROOF. obvious from the definitions. □

1.4. Equations and Clauses.

DEFINITION 2.14. *A* multiset *M over a set X is a function from X to the non-negative integers. The value $M(x)$, for $x \in X$, is the number of occurrences of x in M. We say x is an* element *of M if $M(x) > 0$.*

A multiset M_1 is a submultiset *of M_2, written as $M_1 \subseteq M_2$, if $M_1(x) \leq M_2(x)$, for all $x \in X$. The* union *and* intersection *of two multisets are defined as $(M_1 \cup M_2)(x) = M_1(x) + M_2(x)$ and $(M_1 \cap M_2)(x) = \min(M_1(x), M_2(x))$, respectively. If $M(x) = 0$ for all but finitely many $x \in X$, M is called a* finite *multiset.*

By $M = \{x, x, x\}$ we denote the multiset M for which $M(x) = 3$ and $M(y) = 0$ for $y \neq x$. To simplify notation we also define some operations between sets and multisets.

[3] Some authors call Church–Rosser and terminating rewrite relations also *canonical*.

DEFINITION 2.15. *Let S be a set and M a multiset. We write $M \subseteq S$ if every element of M is also an element of S. We define the union and intersection between sets and multisets as: $M \cup S$ is the multiset where $(M \cup S)(x) = M(x) + 1$, if $x \in S$, and $(M \cup S)(x) = M(x)$, otherwise; and $M \cap S$ is the set $\{x \in S \mid M(x) > 1\}$.*

DEFINITION 2.16. *An* equation *$s \approx t$ is considered as the multiset consisting of the two terms s and t. A* clause *is a pair of multisets of equations, written as $\Gamma \Rightarrow \Phi$. The multisets Γ and Φ are called the* antecedent *and* succedent*, respectively.*

We will write $\mathcal{M}(\Gamma)$, where $\Gamma = \{s_1 \approx t_1, \ldots, s_n \approx t_n\}$ is a multiset of equations, to denote the multiset of terms $\mathcal{M}(\Gamma) = \{s_1, t_1, \ldots, s_n, t_n\}$.

The expression Γ_1, Γ_2 will be used as an abbreviation for $\Gamma_1 \cup \Gamma_2$ and Γ, A or A, Γ as an abbreviation for $\Gamma \cup \{A\}$, where Γ ,Γ_1, Γ_2 are multisets of equations and A is an equation. Let A_i, for $1 \leq i \leq m$, and B_j, for $1 \leq j \leq n$ be equations. By an expression like $A_1, \ldots, A_m \Rightarrow B_1, \ldots, B_n$ we mean the clause $\{A_1, \ldots, A_m\} \Rightarrow \{B_1, \ldots, B_n\}$, which represents the *universally quantified logical formula* $A_1 \wedge \ldots \wedge A_m \Longrightarrow B_1 \vee \ldots \vee B_n$.

A *Horn clause* is a clause with at most one equation in its succedent. If a clause has exactly one equation in its succedent the clause can also be thought of as a *conditional equation*, i.e. an equation which only holds if all conditions in its antecedent are true. Clauses with at most one equation at all are called *unit clauses*. If the equation of a unit clause occurs in the succedent it is called a *positive unit clause*, i.e. an equation. For the sake of convenience we will abbreviate positive unit clauses $\{\} \Rightarrow s \approx t$ by $s \approx t$.

DEFINITION 2.17. *Let $A = s \approx t$ be an equation and $C = A_1, \ldots, A_m \Rightarrow B_1, \ldots, B_n$ a clause. We define $A\sigma$ as $s\sigma \approx t\sigma$ and $C\sigma$ as $A_1\sigma, \ldots, A_m\sigma \Rightarrow B_1\sigma, \ldots, B_n\sigma$. Again, $A\sigma$ and $C\sigma$ are called* instances *of A and C, respectively. If $A\sigma$ or $C\sigma$ consists only of ground terms it is called a* ground instance.

2. Orderings

DEFINITION 2.18. *Let S be a set. A* quasi-ordering *on S is a reflexive, transitive binary relation on S, whereas an* ordering *on S is an irreflexive, transitive binary relation on S. We call an ordering $\succ$* terminating *or* well-founded *if it admits no infinite descending sequence like $x_1 \succ x_2 \succ \cdots$.*

Each ordering $\succ$ on a set S defines a quasi-ordering $\succeq$ on S in the following way: Let $x, y \in S$. $x \succeq y$ if and only if $x \succ y$ or $x = y$.

DEFINITION 2.19. *Let $\succ_1$ and $\succ_2$ be orderings on sets S_1 and S_2, respectively. The* lexicographic combination of $\succ_1$ and $\succ_2$, *written as $\succ_{1,2}$, is an ordering on $S_1 \times S_2$ defined by:*

$$(x_1, x_2) \succ_{1,2} (y_1, y_2) \text{ iff either } x_1 \succ_1 y_1, \text{ or } x_1 = y_1 \text{ and } x_2 \succ y_2.$$

The lexicographic combination of more than two orderings is defined analogously. As a special case, we can use the lexicographic combination to order n-tuples over S.

Let $\succ$ be an ordering on S. The lexicographic ordering $\succ_{lex}$ of n-tuples over S is defined by:

$$(x_1, \ldots, x_n) \succ_{lex} (y_1, \ldots, y_n) \text{ iff there exists } k,\ 1 \leq k \leq n \text{ such that } x_i = y_i, \text{ for all } i < k \text{ and } x_k \succ y_k.$$

In the next definition we show how an ordering $\succ$ on a set S can be extended to an ordering on finite multisets over S. The idea is, that any element of a multiset replaces any finite number of smaller (with respect to $\succ$) elements.

DEFINITION 2.20. *The ordering $\succ$ on a set S defines an ordering $\succ_{mul}$ on finite multisets over S as follows:*

$$M \succ_{mul} N \text{ iff } M \neq N \text{ and, whenever } N(x) > M(x) \text{ then } M(y) > N(y) \text{ for some } y \succ x.$$

For example, let S be the set of natural numbers and $\succ$ the the "greater than" ordering. We have $\{4, 4, 5, 1\} \succ_{mul} \{4, 3, 3, 2, 2, 2, 5\}$.

In [**Dershowitz and Manna, 1979**] it is shown that the multiset ordering $\succ_{mul}$ is well-founded and/or total if and only if the ordering $\succ$ is well-founded and/or total.

DEFINITION 2.21. *Let S be a set or a multiset, $\succ$ an ordering on S. We say that x is* maximal *in S if there is no $y \in S$ with $y \succ x$, and* strictly maximal *if there is no $y \in S$ with $y \succeq x$.*

DEFINITION 2.22. *An ordering $\succ$ on $\mathcal{T}(\mathcal{F}, \mathcal{V})$ is called a* rewrite ordering *if $\succ$ is also a rewrite relation. Well-founded rewrite orderings are called* reduction orderings. *If a reduction ordering $\succ$ satisfies also the* subterm property *(i.e. $u[s] \succ s$, for all terms u and proper subterms s of u) it is called a* simplification ordering.

Note, the transitive closure $\rightarrow^+$ of any rewrite relation $\rightarrow$ defines a rewrite ordering.

A well known example of a reduction ordering (which is also a simplification ordering) is the *lexicographic recursive path ordering*. It is based on a well-founded ordering $\succ_{\mathcal{F}}$ on the set of function symbols $\mathcal{F}$, called *precedence*, and is recursively defined by:

$$s = f(s_1, \ldots, s_m) \succ g(t_1, \ldots, t_n) = t$$

if and only if

$$s_i \succeq t, \text{ for some } i \text{ with } 1 \leq i \leq m,$$

or

$$f \succ_{\mathcal{F}} g \text{ and } s \succ t_i, \text{ for all } i \text{ with } 1 \leq i \leq n,$$

or

$$f = g \text{ and } (s_1, \ldots, s_m) \succ_{lex} (t_1, \ldots, t_n)$$

$$\text{and } s \succ t_i, \text{ for all } i \text{ with } 1 \leq i \leq n.$$

For more details about rewrite orderings we refer to [**Dershowitz, 1982**] and [**Dershowitz, 1987**].

DEFINITION 2.23. *A reduction ordering $\succ$ is said to be* complete *if it is total on ground terms (i.e. $s \succ t$ or $t \succ s$, for all distinct ground terms s and t).*

LEMMA 2.24. *Let $\succ$ be the lexicographic recursive path ordering based on a well-founded precedence $\succ_{\mathcal{F}}$. If $\succ_{\mathcal{F}}$ is total then $\succ$ is well-founded and complete.*

PROOF. [**Bachmair, 1991**] $\square$

3. Equational Theories

In this section we define what we mean by the equational theory generated by a set of conditional equations.

DEFINITION 2.25. *Let $\sim$ be an equivalence relation on $\mathcal{T}(\mathcal{F}, \mathcal{V})$. We say that $\sim$ is a* congruence *relation on $\mathcal{T}(\mathcal{F}, \mathcal{V})$, if $s \sim t$ implies $u[s]_p \sim u[t]_p$ holds, for all terms s, t, u and positions p in u.*

In other words congruence relations are equivalence relations which are closed under context application. For example, the symmetric-transitive-reflexive closure $\leftrightarrow^*$ of a rewrite relation $\to$ is a congruence relation.

PROPOSITION 2.26. *Given a set E of conditional equations. There exists a smallest congruence relation $\equiv_E$ on $\mathcal{T}(\mathcal{F}, \mathcal{V})$ such that $s\sigma \equiv_E t\sigma$ whenever $s_1\sigma \equiv_E t_1\sigma, \ldots, s_n\sigma \equiv_E t_n\sigma$, for all $s_1 \approx t_1, \ldots, s_n \approx t_n \Rightarrow s \approx t \in E$ and substitutions σ. We call $\equiv_E$ the congruence generated by E.*

PROOF. We only sketch the idea of the proof. The missing details can be found in [**Kaplan, 1984**] and [**Kaplan, 1983**].

Let X be the complete lattice of congruences on $\mathcal{T}(\mathcal{F}, \mathcal{V})$ partially ordered by inclusion $\subseteq$. We define the function $\theta : X \to X, \sim_1 \mapsto \sim_2$ by:

(i) $s \sim_1 t$ implies $s \sim_2 t$, for all $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$.
(ii) $\sim_2$ is the smallest congruence relation such that $s\sigma \sim_2 t\sigma$ whenever $s_1\sigma \sim_1 t_1\sigma, \ldots, s_n\sigma \sim_1 t_n\sigma$, for all $s_1 \approx t_1, \ldots, s_n \approx t_n \Rightarrow s \approx t \in E$ and substitutions σ.

The function θ is well-defined, monotone and continuous. Using the Fixpoint Theorem we may infer that θ has a least fix point $\equiv_E$ which is equal to $\bigcup_{n\in\mathbb{N}} \theta^n(=)$. □

Given a set E of conditional equations. We will denote the congruence class of the term t (with respect to $\equiv_E$) by $[t]_E$ and the set of all congruence classes by $\mathcal{T}(\mathcal{F}, \mathcal{V})/_{\equiv_E}$.

DEFINITION 2.27. *Let E be a set of conditional equations and $\equiv_E$ the congruence generated by E. Two terms s and t are said to be* equal with respect to E *if and only if $s \equiv_E t$. The* equational theory of E, $\mathbf{Th}(E)$, *is the set of equations which are equal with respect to E. I.e. $\mathbf{Th}(E) = \{s \approx t \mid s \equiv_E t\}$.*

Another way of defining the equational theory generated by a set of conditional equations is based on the idea of "replacing equals by equals".

DEFINITION 2.28. *Let E be a set of conditional equations and s and t terms. We define the binary relation $\leftrightarrow_E$ as:*

$s \leftrightarrow_E t$ if there exists $\Delta \Rightarrow v \approx w \in E$,
a position p in s, and
a substitution σ,
such that $v'\sigma \leftrightarrow_E^* w'\sigma$, for all $v' \approx w' \in \Delta$, and
$s|_p = v\sigma$, $t = s[w\sigma]_p$.

We write $E \vdash s \approx t$ if and only if $s \leftrightarrow_E^* t$.

If E consists only of unconditional equations it is quite clear that $\equiv_E = \leftrightarrow_E^*$. Kaplan showed that this also holds for the case of conditional equations.

PROPOSITION 2.29. *Given a set E of conditional equations.*

$$s \equiv_E t \text{ if and only if } s \leftrightarrow_E^* t,$$

for all terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$.

PROOF. [**Kaplan, 1984**] □

4. Rewrite Systems

As mentioned in the last section, reasoning in equational theories can be seen as replacement of an instance of one side of an equation by the corresponding instance of the other side of the equation. The central idea in rewriting is now to "orient equations into rules" and use those rules only in "one direction". When we are using rules for replacement we do no longer have the choice which side we replace by the other one, but we are only allowed to replace instances of the left-hand side by instances of the right-hand side.

DEFINITION 2.30. *A* rule *is an ordered pair of terms, written as $s \to t$. A* conditional rule $\Gamma \Rightarrow s \to t$ *is a pair consisting of a multiset* Γ *of equations, the condition or antecedent, and a rule $s \to t$, the succedent.*

Given a conditional equation $A = \Gamma \Rightarrow s \approx t$. We can orient A in two different ways into a rule, namely as: $\Gamma \Rightarrow s \to t$ or $\Gamma \Rightarrow t \to s$. On the other hand we can associate to each conditional rule $\Gamma \Rightarrow s \to t$ the conditional equation $\Gamma \Rightarrow s \approx t$.

Next, we define what we mean by the rewrite relation $\to_R$ generated by a set R of conditional rules. Note, that the definition of $\to_R$ is quite similar to the definition of $\leftrightarrow_E$ in Section 3, accept that the conditions are now checked using $\downarrow_R^*$ instead of using $\leftrightarrow_E^*$.

DEFINITION 2.31. *Let R be a set of conditional rules. The* rewrite relation $\to_R$ generated by R *is defined by:*

$$\begin{aligned} s \to_R t \textit{ if there exists } & \Delta \Rightarrow v \to w \in R, \\ & \textit{a position } p \textit{ in } s, \textit{ and} \\ & \textit{a substitution } \sigma, \\ & \textit{such that } v'\sigma \downarrow_R^* w'\sigma, \textit{ for all } v' \approx w' \in \Delta, \textit{ and} \\ & s|_p = v\sigma,\ t = s[w\sigma]_p. \end{aligned}$$

If we are mainly interested in the rewrite relation $\to_R$ generated by a set R of rules we also speak of the rewrite system R.

PROPOSITION 2.32. *Let R be a set of conditional rules and E be the set of corresponding conditional equations.*

(i) *$s \leftrightarrow_R^* t$ implies $s \leftrightarrow_E^* t$, for all $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$.*
(ii) *If $\to_R$ is Church–Rosser, then $\downarrow_R^* = \leftrightarrow_R^* = \leftrightarrow_E^*$.*

PROOF. [**Kaplan, 1984**] □

Hence, only if $\to_R$ is Church–Rosser we can hope to use rewriting successfully as a technique for reasoning in conditional equational theories.

In conditional rewriting problems occur which do not exist in purely equation rewriting. Consider the following two sets of rules:

$$R_1 = \{f(f(x)) \approx a \Rightarrow f(x) \to a\}$$

$$R_2 = \{f(x, y, z) \approx a \Rightarrow g(x, y) \to h(z)\}$$

It is not possible to rewrite the term $f(a)$ by R_1, because one would get into an infinite loop while checking the condition.

On the other hand rewriting the term $g(a, b)$ with respect to R_2 requires solving the equation $f(a, b, z) \approx a$.

In order to avoid such problems and to guarantee the well-foundedness of $\rightarrow_R$ we have to impose several restrictions on the rules.

DEFINITION 2.33. *Let $\succ$ be a reduction ordering. A conditional rule $\Gamma \Rightarrow s \rightarrow t$ or a conditional equation $\Gamma \Rightarrow s \approx t$ is called* simplifying *(with respect to $\succ$) if $\{s\} \succ_{mul} \{t\} \cup \mathcal{M}(\Gamma)$.*

The above definition is equivalent to:
A conditional rule $\Gamma \Rightarrow s \rightarrow t$ or a conditional equation $\Gamma \Rightarrow s \approx t$ is simplifying (with respect to $\succ$) if

(i) $s \succ t$ and
(ii) $s \succ v$ and $s \succ w$, for all $v \approx w \in \Gamma$.

Condition (i) is well known from equational rewriting, whereas condition (ii) ensures that while checking the conditions no bigger terms than s have to be considered, therefore no loops can occur and no equation solving is necessary.

These restrictions ensure that conditional rewriting becomes manageable and is well-founded as shown in the next Lemma.

LEMMA 2.34. *Given a reduction ordering $\succ$ and a set R of simplifying conditional rules, then:*

(i) *$\rightarrow_R$ is decidable and the normal form is computable.*
(ii) *$\rightarrow_R$ is well-founded.*

PROOF. [**Kaplan, 1987**] $\square$

5. Semantics

DEFINITION 2.35. *An* $\mathcal{F}$-algebra *is a pair* $\mathfrak{A} = (A, \mathfrak{a})$*, where* A *is a non-empty set of values, the* universe *or* domain *of* $\mathfrak{A}$*, and* $\mathfrak{a}$ *is a map defined on* $\mathcal{F}$*, called* operator-map*, satisfying: For every* n*-ary function symbol* $f \in \mathcal{F}$*,* $\mathfrak{a}(f)$ *is an* n*-ary function on* A*.*

DEFINITION 2.36. *An* $\mathcal{F}$-interpretation *is a pair* $\mathfrak{I} = (\mathfrak{A}, \beta)$ *consisting of an* $\mathcal{F}$*-algebra* $\mathfrak{A} = (A, \mathfrak{a})$ *and a map* β*, called* assignment*, from the set of variables* $\mathcal{V}$ *into the domain* A*.*

DEFINITION 2.37. *Given an* $\mathcal{F}$*-interpretation* $\mathfrak{I} = (\mathfrak{A}, \beta)$*. We attach to each term* $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ *an element* $\mathfrak{I}(t)$ *from the domain, called its* meaning*, which is defined by:*

$$\mathfrak{I}(t) = \begin{cases} \beta(t) & \text{, if } t \in \mathcal{V}; \\ \mathfrak{a}(f)(\mathfrak{I}(t_1), \ldots, \mathfrak{I}(t_n)) & \text{, if } t = f\, t_1 \ldots t_n. \end{cases}$$

Further, let $s \approx t$ *be an equation,* Δ *a multiset of equations,* $\Gamma \Rightarrow \Phi$ *a clause and* N *a set of clauses. We define the relation* $\mathfrak{I}$ is a model of N, $\mathfrak{I} \models N$*, as:*

$\mathfrak{I} \models s \approx t$	*if and only if*	$\mathfrak{I}(s) = \mathfrak{I}(t)$.
$\mathfrak{I} \models \Delta$	*if and only if*	$\mathfrak{I} \models s \approx t$*, for all* $s \approx t \in \Delta$.
$\mathfrak{I} \models \Gamma \Rightarrow \Phi$	*if and only if*	$\mathfrak{I} \models \Gamma$ *implies* $\mathfrak{I} \models \Phi$.
$\mathfrak{I} \models N$	*if and only if*	$\mathfrak{I} \models \Gamma \Rightarrow \Phi$*, for all* $\Gamma \Rightarrow \Phi \in N$.

If $\mathfrak{I} \models N$ we also say that $\mathfrak{I}$ *satisfies* N or N *is true in* $\mathfrak{I}$. Using the notion of satisfaction we can state whether a clause is a consequence of a set of clauses or not.

DEFINITION 2.38. *Let* N *be a set of clauses. A clause* C *is called a* consequence *of* N*, written as* $N \models C$ *if and only if every interpretation which is a model of* N *is also a model of* C*.*

Let E be a set of unconditional equations. It is quite obvious that "replacement of equals by equals" as defined in Section 3, is *sound*, i.e. $E \vdash s \approx t$ implies $E \models s \approx t$, for all terms s and t. In 1935 Birkhoff showed in [**Birkhoff, 1935**] that the other direction of the above implication is also true ("replacement of equals by equals" is also *complete*).

Again, this theorem can be generalized to the case where E is a set of conditional equations.

THEOREM 2.39. *For any set* E *of conditional equations and for any terms* s *and* t*,*

$$E \vdash s \approx t \text{ if and only if } E \models s \approx t.$$

PROOF. see [**Kaplan, 1983**]. □

Therefore, in conditional equational theories the syntactic notion of $\vdash$ and the semantic notion of $\models$ coincide. In general one is interested in the question: "Given a set E of conditional equations and terms s and t. Is $s \approx t$ a consequence of E?". According to Theorem 2.39 it suffices to check whether $E \vdash s \approx t$ or not. If we are able to find a convergent rewrite system R such that $\leftrightarrow_E^* = \leftrightarrow_R^*$ we are done. In Chapter 3 we will show how such a convergent set of rewrite rules can be constructed with the help of Standard Completion. Since $E \models s \approx t$ is semi-decidable, we will not always be able to find such a set of rewrite rules which means that the completion procedure will not always succeed.

Another question which often arises is: "Given a set N of clauses. Does there exist a model at all?". If N consists only of conditional equations the answer is "yes", since $\mathcal{T}(\mathcal{F})/_{\equiv_N}$ is a model of N. But if N consists of arbitrary clauses this need not be case.

DEFINITION 2.40. *A clause C is* satisfiable *if and only if there exists an interpretation which is a model of C. A set N of clauses is satisfiable if and only if there is an interpretation which is a model for all clauses in N.*

Among all $\mathcal{F}$-algebras the *ground-term algebra* plays a special role, because it is the initial element in the set of all $\mathcal{F}$-algebras. Its universe A is the Herbrand universe $\mathcal{T}(\mathcal{F})$ and its operator-map $\mathfrak{a}$ is defined by $\mathfrak{a}(f) = f$, for all function symbols $f \in \mathcal{F}$. The importance of the ground-algebra lies in the fact that it is unique and that all other $\mathcal{F}$-algebras are homomorphic images of it. So the ground-term algebra is the most "abstract" algebra. Interpretations based on the ground term algebra are called Herbrand-interpretations.

DEFINITION 2.41. *Let $\mathfrak{I} = (\mathfrak{A}, \beta)$ an $\mathcal{F}$-interpretation, where $\mathfrak{A} = (A, \mathfrak{a})$. The interpretation $\mathfrak{I}$ is called a* Herbrand interpretation *or* H-interpretation *if $A = \mathcal{T}(\mathcal{F})$ and $\mathfrak{a}(f) = f$, for all function symbols $f \in \mathcal{F}$.*

Since we are only dealing with equations (there are no predicates or logical connectives), H-interpretations can also be thought of as congruences on ground terms.

In the case where N consists only of conditional equations we have shown that there always exists a congruence $\equiv_N$ such that $s \equiv_N t$ if and only if $N \models s \approx t$. Therefore the quotient $\mathcal{T}(\mathcal{F})/_{\equiv_N}$ of the ground-term algebra $\mathcal{T}(\mathcal{F})$ and the congruence $\equiv_N$ is a model of N. It can be shown that this model is the initial element in the set of all models of N and is called the *initial algebra*.

As said above, an H-interpretation can also be interpreted as a congruence on ground terms which itself can be represented by a ground-convergent rewrite system. Given a satisfiable set N of clauses. If there exists a ground-convergent rewrite system R, such that $N \models s \approx t$ if and only if $s \leftrightarrow_R^* t$, for all ground terms s and t, the initial algebra is isomorphic to the *normal-form algebra of R* ([**Goguen, 1987**]), i.e. the normal forms are chosen as representatives for the congruence classes of $\mathcal{T}(\mathcal{F})/_{\equiv_N}$.

DEFINITION 2.42. *Let R be a ground-convergent rewrite system. We define the* normal-form algebra $\mathfrak{I}_R$ induced by R, *as*

$$\mathfrak{I}_R(f\,t_1 \ldots t_n) = \mathrm{nf}_{\to_R}(f\,t_1 \ldots t_n)$$

In other words: $\mathfrak{I}_R \models s \approx t$ if and only if $s \downarrow_R^! t$. As an abbreviation we will just say "$s \approx t$ is true in R" instead of "$s \approx t$ is true in the normal form algebra induced by R".

CHAPTER 3

Standard Completion

In this chapter we discuss the well known Knuth–Bendix completion procedure ([**Knuth and Bendix, 1970**]), which we call *Standard Equational Completion*, and its extension to conditional equations ([**Kaplan, 1987**] and [**Jouannaud and Waldmann, 1987**]) called *Standard Conditional Completion*. Both procedures will be described by systems of inference rules, called $\mathcal{EC}$ and $\mathcal{CC}$, respectively. Since in Standard Completion we strictly distinguish between equations and rules the inference rules are acting on pairs of sets of equations and sets of rules.

Starting from a set E of equations a standard completion procedure tries to generate a convergent set R of rules such that $\leftrightarrow_E^* = \leftrightarrow_R^*$. For this purpose it may be necessary that new equations, so called *critical pairs*, have to be deduced (by *superposition*). On the other hand, equations may be replaced by other equations which are logically equivalent but "smaller" with respect to some reduction ordering, whereas trivial equations can be deleted at all.

Standard Completion need not succeed always and may fail for the following two reasons:

- It may happen that infinitely many rules are generated by superposition (i.e. the set of rules does not become Church–Rosser).
- It may be the case that an equation is generated which cannot be oriented into a simplifying rule (i.e. termination of $\rightarrow_R$ is longer guaranteed).

1. Standard Completion Derivations

In Standard Completion (unlike in Ordered Completion) we define the inference systems on pairs of sets of equations and sets of rules.

DEFINITION 3.1. *An* inference rule *for Standard Completion is a binary relation on pairs of the form (E, R), where E is a set of conditional equations and R is a set of conditional rules. A set of inference rules for Standard Completion is called a* standard inference system.

One can think of inference rules to be the elementary computation steps of completion. We can distinguish three kinds of such rules, namely rules which add new equations (*deduction rules*), rules which delete equations (*deletion rules*) and rules which replace equations by logically equivalent but "smaller" ones (*simplification rules*).

DEFINITION 3.2. *Let $\mathcal{I}$ be a standard inference system. A sequence $(E_0, R_0) \vdash_{\mathcal{I}} (E_1, R_1) \vdash_{\mathcal{I}} \cdots$, where E_i are sets of conditional equations and R_i are sets of conditional rules, for all $i \in \mathbb{N}$, is a* derivation in $\mathcal{I}$ *if each pair (E_{i+1}, R_{i+1})*

can be obtained from (E_i, R_i) *by one or more applications of inference rules of* $\mathcal{I}$.

The set $(E_\infty, R_\infty) = (\bigcup_i \bigcap_{j\geq i} E_j, \bigcup_i \bigcap_{j\geq i} R_j)$ *is called the* limit *of the derivation. Equations in* E_∞ *and rules in* R_∞ *are called* persisting.

DEFINITION 3.3. *A* standard completion procedure for $\mathcal{I}$ *is a program which takes a set of equations* E *as input and generates a derivation* $(E_0, R_0) \vdash_{\mathcal{I}} (E_1, R_1) \vdash_{\mathcal{I}} \cdots$ *according to some standard inference system* $\mathcal{I}$.

The procedure succeeds *for a given input if* $E_\infty = \emptyset$ *and* R_∞ *is convergent. We say that the procedure has failed if* $E_\infty \neq \emptyset$ *and we call it* correct *if* R_∞ *is convergent whenever* $E_\infty = \emptyset$.

2. Proof Orderings

We will show correctness of standard completion with the help of *proof orderings*. In order to be able to do so we first have to define what we mean by a conditional proof.

DEFINITION 3.4. *Let E be a set of equations and R be a set of rules. A* conditional proof $P = (P_1, \ldots, P_n)$ in (E, R) *of an equation $t_0 \approx t_n$, for $n \geq 0$, is any finite sequence of* proof steps P_i in (E, R), *of the form*

$$\frac{Q_{11} \ldots Q_{1k_1}}{t_0\, \vartheta_1\, t_1}, \frac{Q_{21} \ldots Q_{2k_2}}{t_1\, \vartheta_2\, t_2}, \ldots, \frac{Q_{n1} \ldots Q_{nk_n}}{t_{n-1}\, \vartheta_n\, t_n}.$$

Each proof step $P_i = \dfrac{Q_{i1} \ldots Q_{ik_i}}{t_{i-1}\, \vartheta_i\, t_i}$, for $1 \leq i \leq n$, is of one of the following three types:

- $\dfrac{Q_1 \ldots Q_k}{s \leftrightarrow_C^{p,\sigma} t}$, *where $C = v_1 \approx w_1, \ldots, v_k \approx w_k \Rightarrow v \approx w \in E$, p is a position in s and σ is a substitution such that $s|_p = v\sigma$, $t = s[w\sigma]_p$ and Q_j is a proof in (E, R) of $v_j\sigma \approx w_j\sigma$, for all $1 \leq j \leq k$.*
- $\dfrac{Q_1 \ldots Q_k}{s \rightarrow_C^{p,\sigma} t}$, *where $C = v_1 \approx w_1, \ldots, v_k \approx w_w \Rightarrow v \rightarrow w \in R$, p is a position in s and σ is a substitution such that $s|_p = v\sigma$, $t = s[w\sigma]_p$ and Q_j is a proof in (E, R) of $v_j\sigma \approx w_j\sigma$, for all $1 \leq j \leq k$.*
- $\dfrac{Q_1 \ldots Q_k}{s \leftarrow_C^{p,\sigma} t}$, *where $C = v_1 \approx w_1, \ldots, v_k \approx w_w \Rightarrow v \rightarrow w \in R$, p is a position in s and σ is a substitution such that $t|_p = v\sigma$, $s = t[w\sigma]_p$ and Q_j is a proof in (E, R) of $v_j\sigma \approx w_j\sigma$, for all $1 \leq j \leq k$.*

By () we will denote the empty proof which is a proof of the trivial equation $t \approx t$, for any term t.

The depth *of a proof is one more than the maximum depth of proofs for its conditions. Unconditional proofs have depth 0.*

We introduce the following abbreviations:

Unconditional proof steps will be written as

$$s \leftrightarrow_{v \approx w}^{p,\sigma} t,\ s \rightarrow_{v \approx w}^{p,\sigma} t \text{ or } s \leftarrow_{v \approx w}^{p,\sigma} t,$$

respectively.

The proof

$$\frac{Q_{11} \ldots Q_{1k_1}}{t_0\, \vartheta_1\, t_1}, \frac{Q_{21} \ldots Q_{2k_2}}{t_1\, \vartheta_2\, t_2}, \ldots, \frac{Q_{n1} \ldots Q_{nk_n}}{t_{n-1}\, \vartheta_n\, t_n}$$

will be written as

$$\frac{Q_{11} \ldots Q_{1k_1} Q_{21} \ldots Q_{2k_2} \ldots Q_{n1} \ldots Q_{nk_n}}{t_0\, \vartheta_1\, t_1\, \vartheta_2\, t_2 \cdots t_{n-1}\, \vartheta_n\, t_n}.$$

If all proof steps of a proof are of the same kind we will write

$$\frac{Q_1 \ldots Q_k}{t_0 \rightarrow^* t_n} \quad \text{for} \quad \frac{Q_1 \ldots Q_k}{t_0 \rightarrow_{C_1}^{p_1,\sigma_1} t_1 \ldots t_{n-1} \rightarrow_{C_n}^{p_n,\sigma_n} t_n}$$

and

$$\frac{Q_1 \ldots Q_k}{t_0 \leftarrow^* t_n} \quad \text{for} \quad \frac{Q_1 \ldots Q_k}{t_0 \leftarrow_{C_1}^{p_1,\sigma_1} t_1 \ldots t_{n-1} \leftarrow_{C_n}^{p_n,\sigma_n} t_n}.$$

DEFINITION 3.5. *A proof is called a* rewrite proof *if it has the form*

$$\frac{Q_1 \ldots Q_k}{s \rightarrow^* u \leftarrow^* t}$$

where all Q_i, $0 \leq i < k$ are also rewrite proofs.

Hence, rewrite proofs contain no proof steps which use equations and they have no *peaks* of the form $\dfrac{Q_1 \ldots Q_k}{s \leftarrow^* u \rightarrow^* t}$ as subproofs.

DEFINITION 3.6. *Let P be the proof*

$$\frac{Q_{11} \ldots Q_{1k_1} \ldots Q_{n1} \ldots Q_{nk_n}}{t_0 \leftrightarrow_{C_1}^{p_1,\sigma_1} t_1 \cdots t_{n-1} \leftrightarrow_{C_n}^{p_n,\sigma_n} t_n}.$$

We define P^{-1} to be the proof

$$\frac{Q_{n1} \ldots Q_{nk_n} \ldots Q_{11} \ldots Q_{1k_1}}{t_n \leftrightarrow_{C_n}^{p_n,\sigma_n} t_{n-1} \cdots t_1 \leftrightarrow_{C_1}^{p_1,\sigma_1} t_0},$$

$P\sigma$, where σ is a substitution, to be the proof

$$\frac{Q_{11}\sigma \ldots Q_{1k_1}\sigma \ldots Q_{n1}\sigma \ldots Q_{nk_n}\sigma}{t_0\sigma \leftrightarrow_{C_1}^{p_1,\sigma_1\sigma} t_1\sigma \cdots t_{n-1}\sigma \leftrightarrow_{C_n}^{p_n,\sigma_n\sigma} t_n\sigma}$$

and $u[P]_p$, where p is a position in u, to be the proof

$$\frac{Q_{11} \ldots Q_{1k_1} \ldots Q_{n1} \ldots Q_{nk_n}}{u[t_0]_p \leftrightarrow_{C_1}^{p.p_1,\sigma_1} u[t_1]_p \cdots u[t_{n-1}]_p \leftrightarrow_{C_n}^{p.p_n,\sigma_n} u[t_n]_p}.$$

A proof P' is a subproof of P, written as $P[P']$, if either P' is of the form

$$\frac{Q_{i1} \ldots Q_{ik_1} \ldots Q_{j1} \ldots Q_{jk_n}}{t_{i-1} \leftrightarrow_{C_i}^{p_i,\sigma_i} t_i \cdots t_{j-1} \leftrightarrow_{C_j}^{p_j,\sigma_j} t_j},$$

for some $1 \leq i \leq j \leq n$ or P' is a subproof of some Q_{ij}, for $1 \leq i \leq n$ and $1 \leq j \leq k_i$.

These definitions are generalized in the obvious way to proofs consisting of all types of proof steps.

It is clear that if P is a proof in (E, R), then P^{-1}, $P\sigma$, $u[P]_p$ and P', where P' is a subproof of P, are also proofs in (E, R).

We can now define what a proof ordering is.

DEFINITION 3.7. *A* proof ordering $>$ *is an ordering on proofs which satisfies the properties:*[1]

(i) $P > Q$ *implies* $u[P]_p > u[Q]_p$*, for all proofs* P *and* Q*, terms* u *and positions* p *in* u*.*

(ii) $P > Q$ *implies* $P\sigma > Q\sigma$*, for all proofs* P *and* Q *and substitutions* σ*.*

(iii) $Q > Q'$ *implies* $P[Q] > P[Q']$*, for all proofs* P*, subproofs* Q *of* P *and all proofs* Q'*, which prove the same equation as* Q*.*

If a proof ordering is a well-founded ordering, it is called a proof reduction ordering.

DEFINITION 3.8. *A proof ordering* $>$ *is said to be* compatible *with an inference system* $\mathcal{I}$ *if* $(E,R) \vdash_{\mathcal{I}} (E',R')$ *implies that for any proof* P *in* (E,R) *there exists a proof* P' *in* (E',R') *such that* $P \geq P'$*.*

To show correctness of standard completion we will use proof orderings which are based on a *complexity function* c from the set of proof steps to some set X.

Let P be the proof

$$\frac{Q_{11}\ldots Q_{1k_1}\ldots Q_{n1}\ldots Q_{nk_n}}{t_0\,\vartheta_1\,t_1\cdots t_{n-1}\,\vartheta_n\,t_n}.$$

The complexity $c^p(P)$ of the proof P is defined as the multiset-union of the complexities of its proof steps, i.e.

$$c^p(P) = c^s(\frac{Q_{11}\ldots Q_{1k_1}}{t_0\,\vartheta_1\,t_1}) \cup \ldots \cup c^s(\frac{Q_{n1}\ldots Q_{nk_n}}{t_{n-1}\,\vartheta_n\,t_n}).$$

Where the complexity $c^s(P_i)$ of a proof step $P_i = \dfrac{Q_{i1}\ldots Q_{ik_i}}{t_{i-1}\,\vartheta_i\,t_i}$ is the multiset-union

$$c^s(P_i) = \{c(P_i)\} \cup c^p(Q_{i1}) \cup \ldots \cup c^p(Q_{ik_i}).$$

Hence, if $>^c$ is an ordering on X, its multiset extension $>^c_{mul}$ is an ordering on complexities of proof steps and proofs. The *proof ordering* $>$ *based on the complexity function* c is then defined by:

$$P > Q \text{ if and only if } c^p(P) >^c_{mul} c^p(Q).$$

[1] Compare this definition to the definitions of rewrite relation and rewrite ordering.

3. The Inference System $\mathcal{EC}$ for Equational Completion

Let $\succ$ be a reduction ordering. The inference system $\mathcal{EC}$ for *equational completion* is given by the inference rules below [**Bachmair et al., 1986**]:

Superposition:

$$\frac{(E, R)}{(E \cup \{v[t]\sigma \approx w\sigma\}, R)} \quad \text{if } s \to t, v[s'] \to w \in R, \; \sigma = \mathrm{mgu}(s, s') \text{ and } s' \text{ is not a variable.}$$

Orientation:

$$\frac{(E \cup \{s \approx t\}, R)}{(E, R \cup \{s \to t\})} \quad \text{if } s \succ t.$$

Deletion:

$$\frac{(E \cup \{s \approx s\}, R)}{(E, R)}$$

Simplification:

$$\frac{(E \cup \{s \approx t\}, R)}{(E \cup \{s \approx u\}, R)} \quad \text{if } t \to_R u.$$

Composition:

$$\frac{(E, R \cup \{s \to t\})}{(E, R \cup \{s \to u\})} \quad \text{if } t \to_R u.$$

Collapse:

$$\frac{(E, R \cup \{s \to t\})}{(E \cup \{u \approx t\}, R)} \quad \text{if } v \to w \in R, \; s \rhd v \text{ and } s \to_{\{v \to w\}} u.$$

It is assumed that the rules used in Superposition have no variables in common (if necessary the variables of one rule have to be renamed).

To show correctness of equational standard completion, we do not necessarily need the inference rules Composition and Collapse. The inference system consisting only of Superposition, Orientation, Deletion and Simplification is known as *basic equational completion* and is also correct but, when implemented, less efficient than $\mathcal{EC}$.

LEMMA 3.9. *Let $(E, R) \vdash_{\mathcal{EC}} (E', R')$ and s and t be terms. There exists a proof P in (E, R) of $s \approx t$ if and only if there exists a proof P' in (E', R') of $s \approx t$.*

PROOF. A proof can be found in [**Bachmair, 1991**]. □

3.1. A Proof Ordering for $\mathcal{EC}$. We next define an ordering $>_{\mathcal{EC}}$ on equational proofs based on the complexity function $c_{\mathcal{EC}}$:

$$c_{\mathcal{EC}}(s \to^{p,\sigma}_{v \approx w} t) = (\{s, t\}, \bot, \bot)$$
$$c_{\mathcal{EC}}(s \to^{p,\sigma}_{v \to w} t) = (\{s\}, v, w)$$
$$c_{\mathcal{EC}}(s \leftarrow^{p,\sigma}_{v \to w} t) = (\{t\}, v, w)$$

The symbol $\perp$ denotes a new symbol not belonging to $\mathcal{T}(\mathcal{F},\mathcal{V})$ with $t \succ \perp$ and $t \rhd \perp$, for all $t \in \mathcal{T}(\mathcal{F},\mathcal{V})$. The ordering $>^{c\mathcal{EC}}$ on the 3-tuples (x_1, x_2, x_3) is the lexicographic combination of $\succ_{mul}$, $\rhd$ and $\succ$.

Let P and Q be proofs, we define: $P >_{\mathcal{EC}} Q$ if and only if $c^p_{\mathcal{EC}}(P) >^{c\mathcal{EC}}_{mul} c^p_{\mathcal{EC}}(Q)$.

LEMMA 3.10. *The ordering $>_{\mathcal{EC}}$ is a proof reduction ordering and compatible with the inference system $\mathcal{EC}$.*

PROOF. It quite clear that $>_{\mathcal{EC}}$ is a proof ordering and that it is well-founded (see remarks about multiset orderings). So it remains to show that $>_{\mathcal{EC}}$ is compatible with $\mathcal{EC}$.

Let P be a proof in (E, R) and let (E', R') be such that $(E, R) \vdash_{\mathcal{EC}} (E', R')$. We will show how a proof Q in (E', R') can be constructed which proves the same equation as P but is smaller than P with respect to $>_{\mathcal{EC}}$. We distinguish the following cases (one case for each inference rule in $\mathcal{EC}$):

(i) *Superposition:* nothing has to be done, because P is also a proof in (E', R').

(ii) *Orientation:*
$$s \rightarrow^{\lambda,\epsilon}_{s\approx t} t \quad >_{\mathcal{EC}} \quad s \rightarrow^{\lambda,\epsilon}_{s\rightarrow t} t,$$
since $\{(\{s,t\},\perp,\perp)\} >^{c\mathcal{EC}}_{mul} \{(\{s\},s,t)\}$.

(iii) *Deletion:*
$$s \rightarrow^{\lambda,\epsilon}_{s\approx s} s \quad >_{\mathcal{EC}} \quad ()$$
since $\{(\{s,s\},\perp,\perp)\} >^{c\mathcal{EC}}_{mul} \{\}$.

(iv) *Simplification:*
$$s \rightarrow^{\lambda,\epsilon}_{s\approx t} t \quad >_{\mathcal{EC}} \quad s \rightarrow^{\lambda,\epsilon}_{s\approx u} u \leftarrow^{p,\sigma}_{v\rightarrow w} t,$$
since $\{(\{s,t\},\perp,\perp)\} >^{c\mathcal{EC}}_{mul} \{(\{s,u\},\perp,\perp),(\{t\},v,w)\}$.

(v) *Composition:*
$$s \rightarrow^{\lambda,\epsilon}_{s\rightarrow t} t \quad >_{\mathcal{EC}} \quad s \rightarrow^{\lambda,\epsilon}_{s\rightarrow u} u \leftarrow^{p,\sigma}_{v\rightarrow w} t,$$
since $\{(\{s\},s,t)\} >^{c\mathcal{EC}}_{mul} \{(\{s\},s,u),(\{t\},v,w)\}$.

(vi) *Collapse:*
$$s \rightarrow^{\lambda,\epsilon}_{s\rightarrow t} t \quad >_{\mathcal{EC}} \quad s \rightarrow^{p,\sigma}_{v\rightarrow w} u \rightarrow^{\lambda,\epsilon}_{u\approx t} t,$$
since $\{(\{s\},s,t)\} >^{c\mathcal{EC}}_{mul} \{(\{s\},v,w),(\{u,t\},\perp,\perp)\}$, because $s \rhd v$.

Let Q be the result of replacing those subproofs of P which use equations and rules that are no longer in (E', R') in the way as shown above. By transitivity of $>_{\mathcal{EC}}$ it follows that $P >_{\mathcal{EC}} Q$. □

3.2. Correctness of $\mathcal{EC}$. Given a standard theorem proving derivation $(E_0, R_0) \vdash_{\mathcal{I}} (E_1, R_1) \vdash_{\mathcal{I}} (E_2, R_2) \cdots$ we next derive conditions such that, for any non-rewrite proof P in (E_i, R_i) of $s \approx t$ there exists a proof Q in (E_j, R_j) of $s \approx t$ with $P >_{\mathcal{EC}} Q$, for some $j \geq i$.

If P is not a rewrite proof it either must contain a proof step of the form $s \leftrightarrow^{p,\sigma}_{v\approx w} t$ (an application of an equation) or a subproof like $s \leftarrow^{p_1,\sigma_1}_{v_1\rightarrow w_1} u \rightarrow^{p_2,\sigma_2}_{v_2\rightarrow w_2} t$ (a peak). Proofs of the first kind can be made smaller by orienting or simplifying the equation. To reduce proofs of the second kind it suffices to deduce so called *critical pairs* (which is done by *superposition*).

LEMMA 3.11 (**Critical Pair Lemma, [Knuth and Bendix, 1970]**). *If P is a peak $s \leftarrow_R u \rightarrow_R t$, then either there exists a rewrite proof $s \rightarrow_R^* u' \leftarrow_R^* t$ or, for some critical pair $v \approx w$ between rules in R, a proof of the form $s \leftrightarrow_{v \approx w}^{p;\sigma} t$.*

DEFINITION 3.12. *A non-failing derivation $(E_0, R_0) \vdash_{\mathcal{EC}} (E_1, R_1) \vdash_{\mathcal{EC}} \cdots$ in $\mathcal{EC}$ is called* fair *if the set of all critical pairs between rules in R_∞ is a subset of $\bigcup_j E_j$.*

That means, in fair derivations all possible critical pairs between persisting rules have been computed.

LEMMA 3.13. *Let $(E_0, R_0) \vdash_{\mathcal{EC}} (E_1, R_1) \vdash_{\mathcal{EC}} \cdots$ be a non-failing and fair derivation and let P be a proof in (E_i, R_i), for some i. If P is not a rewrite proof, then there exists, for some $j \geq i$, a proof Q in (E_j, R_j) such that $P >_{\mathcal{EC}} Q$.*

PROOF. [**Bachmair et al., 1986**]

Let P be a proof in (E_i, R_i) and assume that P is not a rewrite proof. Then there are two possibilities:

(i) There exists a proof step P_k in P which uses an equations.
(ii) The proof P contains a subproof P' which is a peak.

Ad (i): Let $P_k = s \leftrightarrow_{v \approx w}^{p;\sigma} t$. Since $(E_0, R_0) \vdash_{\mathcal{EC}} (E_1, R_1) \vdash_{\mathcal{EC}} \cdots$ is a non-failing derivation, the equation $v \approx w$ will be formed into a rule, simplified or deleted. All cases result in a smaller proof of $s \approx t$ in (E_j, R_j), for some $j > i$.

Ad (ii): Let $P' = s \leftarrow_{v_1 \rightarrow w_1}^{p_1, \sigma_1} u \rightarrow_{v_2 \rightarrow w_2}^{p_2, \sigma_2} t$. If any of the two rules involved is simplified we get a smaller proof and are done. Therefore, let us assume that both rules persist. By the Critical Pair Lemma we know that either there exists a rewrite proof of $s \approx t$ (which is of course smaller than P'), or there exists a proof of the form $s \leftrightarrow_{v \approx w}^{p;\sigma} t$ (which is also smaller), where $v \approx w$ is a critical pair. Because $(E_0, R_0) \vdash_{\mathcal{EC}} (E_1, R_1) \vdash_{\mathcal{EC}} \cdots$ is assumed to be a fair derivation, there exists a $j > i$, such that $v \approx w \in E_j$ and hence $s \leftrightarrow_{v \approx w}^{p;\sigma} t$ is a proof in (E_j, R_j) of $s \approx t$.

In both cases the subproof for $s \approx t$ can be replaced by a smaller proof and therefore we can construct a proof Q in (E_j, R_j) such that $P >_{\mathcal{EC}} Q$. □

THEOREM 3.14 ([**Huet, 1981**]). *If a non-failing derivation in $\mathcal{EC}$ is fair, then the limit rewrite system R_∞ is convergent.*

PROOF. The limit rewrite system R_∞ is contained in $\succ$ and hence terminating. We have to show that for each proof P in R_∞ of $s \approx t$ there also exists a rewrite proof Q in R_∞ of $s \approx t$. From the well-foundedness of $>_{\mathcal{EC}}$, see Lemma 3.10, it follows that Lemma 3.13 can only be applied finitely many times. Therefore, we can transform P by finitely many steps into a rewrite proof Q in R_∞. □

As a direct consequence of the above theorem we get Corollary 3.15.

COROLLARY 3.15. *A terminating rewrite system R is Church-Rosser if and only if all critical pairs deduced from R are joinable.*

3.3. An Equational Completion Procedure. In the following we describe the completion procedure of Huet [**Huet, 1981**] as an example of a completion procedure for $\mathcal{EC}$ which produces fair derivations.

given: A set E of equations.
find: A convergent set R of rules such that $\leftrightarrow_E^* = \leftrightarrow_R^*$.

begin
  $input(E)$;
  $R := Completion(E, \emptyset, 0)$;
  $output(R)$;
end

function $Completion(E, R, n)$
if $E \neq \emptyset$ **then**
  select an equation $s \approx t$ in E.
  $s' := Normalize(s)$;
  $t' := Normalize(t)$;
  if $s' = t'$ **then**
    $Completion(E \setminus \{s \approx t\}, R, n)$;
  else
    $s'' \to t'' := Orient(s' \approx t')$;
    $R := R \cup \{n : s'' \to t''\}$;
    $K := \{k : v \to w \mid k : v \to w \in R, v$ is reducable w.r.t. $R \setminus \{k : v \to w\}\}$;
    $E := E \setminus \{s \approx t\} \cup \{v' \approx w \mid k : v \to w \in K, v' := Normalize(v)\}$;
    $R' := \{k : v \to w' \mid k : v \to w \in R \setminus K, w' := Normalize(w)\}$;
    $Completion(E, R', n + 1)$;
  fi
elseif $\exists\, j : s \to t \in R$ unmarked **then**
  mark $j : s \to t$.
  $E :=$ set of all critical pairs of $j : s \to t$ and $k : v \to w \in R$, with $k \leq j$;
  $Completion(E, R, n)$;
else
  return(R) %success
fi

The completion procedure leads to one of the following situations:

- It terminates with a convergent rewrite system R which is equivalent to the given set E of equations.
- It fails inside the orientation procedure $Orient$.
- It generates infinitely many rules.

4. The Inference System $\mathcal{CC}$ for Conditional Completion

Let $\succ$ be a reduction ordering. The inference system $\mathcal{CC}$ for *conditional completion* consists of the following inference rules:

Superposition:

$$\frac{(E, R)}{(E \cup \{\Gamma\sigma, \Delta\sigma \Rightarrow v[t]\sigma \approx w\sigma\}, R)}$$

if $\Gamma \Rightarrow s \to t \in R$, $\Delta \Rightarrow v[s'] \to w \in R$, $\sigma = \mathrm{mgu}(s, s')$ and s' is not a variable.

Orientation:

$$\frac{(E \cup \{\Gamma \Rightarrow s \approx t\}, R)}{(E, R \cup \{\Gamma \Rightarrow s \to t\})}$$

if $\{s\} \succ \{t\} \cup \mathcal{M}(\Gamma)$.

Deletion:

$$\frac{(E \cup \{\Gamma \Rightarrow s \approx s\}, R)}{(E, R)}$$

Simplification:

$$\frac{(E \cup \{\Gamma \Rightarrow s \approx t\}, R)}{(E \cup \{\Gamma \Rightarrow s \approx u\}, R)}$$

if $t \to_R u$.

Composition:

$$\frac{(E, R \cup \{\Gamma \Rightarrow s \to t\})}{(E, R \cup \{\Gamma \Rightarrow s \to u\})}$$

if $t \to_R u$.

Collapse:

$$\frac{(E, R \cup \{\Gamma \Rightarrow s \to t\})}{(E \cup \{\Gamma \Rightarrow u \approx t\}, R)}$$

if $\Delta \Rightarrow v \to w \in R$, $s \rhd v$ and $s \to_{\{\Delta \Rightarrow v \to w\}} u$.

Deletion of condition in equation:

$$\frac{(E \cup \{\Gamma, s' \approx s' \Rightarrow s \approx t\}, R)}{(E \cup \{\Gamma \Rightarrow s \approx t\}, R)}$$

Deletion of condition in rule:

$$\frac{(E \cup \{\Gamma, s' \approx s' \Rightarrow s \to t\}, R)}{(E \cup \{\Gamma \Rightarrow s \to t\}, R)}$$

Simplification of condition in equation:

$$\frac{(E \cup \{\Gamma, s' \approx t' \Rightarrow s \approx t\}, R)}{(E \cup \{\Gamma, u' \approx t' \Rightarrow s \approx t\}, R)}$$

if $s \succ s'$ or $t \succ s'$, and $s' \to_R u'$.

Simplification of condition in rule:

$$\frac{(E \cup \{\Gamma, s' \approx t' \Rightarrow s \to t\}, R)}{(E \cup \{\Gamma, u' \approx t' \Rightarrow s \to t\}, R)}$$

if $s' \to_R u'$.

It is assumed that the rules used for Superposition have no variables in common (if necessary the variables of one rule have to be renamed).

The inference system consisting of the rules Superposition, Orientation, Deletion and Simplification is called *basic conditional completion* and describes also a correct completion procedure, but is less efficient.

The same Lemma as we have had it for $\mathcal{EC}$ is also true for the inference system $\mathcal{CC}$.

LEMMA 3.16. *Let $(E_1, R_1) \vdash_{\mathcal{CC}} (E_2, R_2)$ and s and t be terms. There exists a proof P_1 in (E_1, R_1) of $s \approx t$ if and only if there exists a proof P_2 in (E_2, R_2) of $s \approx t$.*

PROOF. The proof is a straight forward generalization of the proof of Lemma 3.9. □

4.1. A Proof Ordering for $\mathcal{CC}$. The complexity function $c_{\mathcal{CC}}$ of proof steps in $\mathcal{CC}$ is defined as:

$$c_{\mathcal{CC}}(\frac{Q_1 \dots Q_k}{s \leftrightarrow^{p,\sigma}_{\Delta \Rightarrow v \approx w} t}) = (\{s,t\} \cup \mathcal{M}(\Delta\sigma), \bot, \{\})$$

$$c_{\mathcal{CC}}(\frac{Q_1 \dots Q_k}{s \rightarrow^{p,\sigma}_{\Delta \Rightarrow v \rightarrow w} t}) = (\{s\}, v, \{w\} \cup \mathcal{M}(\Delta))$$

$$c_{\mathcal{CC}}(\frac{Q_1 \dots Q_k}{s \leftarrow^{p,\sigma}_{\Delta \Rightarrow v \rightarrow w} t}) = (\{t\}, v, \{w\} \cup \mathcal{M}(\Delta))$$

The ordering $>^{ccc}$ on the 3-tuples (x_1, x_2, x_3) is the lexicographic combination of $\succ_{mul}$, $\rhd$ and $\succ_{mul}$.

Let P and Q be proofs, we define: $P >_{\mathcal{CC}} Q$ if and only if $c^p_{\mathcal{CC}}(P) >^{ccc}_{mul} c^p_{\mathcal{CC}}(Q)$.

When proving the compatibility of $>_{\mathcal{CC}}$ with the inference system $\mathcal{CC}$ we need the following technical Lemma.

LEMMA 3.17. *Let R be a set of simplifying conditional rules and s, t be terms.*

(i) *If $s \rightarrow_R t$, then there exists a proof P in R of $s \approx t$ such that*

$$\{(\{s, \bot\}, \bot, \{\})\} >^{ccc}_{mul} c^p_{\mathcal{CC}}(P)$$

If in addition $s \rhd v$, where $\Delta \Rightarrow v \rightarrow w \in R$ is the rule which is applied to rewrite s, it even holds that

$$\{(\{s\}, s, \{\})\} >^{ccc}_{mul} c^p_{\mathcal{CC}}(P).$$

(ii) *If $s \downarrow^*_R t$, then there exists a proof P in R of $s \approx t$ such that*

$$\{(\{s, t\}, \bot, \{\})\} >^{ccc}_{mul} c^p_{\mathcal{CC}}(P)$$

PROOF. We will prove this Lemma by induction on the term s with respect to the reduction ordering $\succ$.

Ad (i): Let $C = v_1 \approx w_1, \dots, v_n \approx w_n \Rightarrow v \rightarrow w$ be the rule which rewrites s into t, i.e. there exists a position p in s and a substitution σ such that $s|_p = v\sigma$, $t = s[w\sigma]_p$ and $v_i\sigma \downarrow^*_R w_i\sigma$, for all $1 \leq i \leq n$.

Since C is simplifying the terms $v_i\sigma$ and $w_i\sigma$, for $1 \leq i \leq n$, are smaller than $v\sigma$, which itself is smaller than or equal to s. Using the induction hypothesis we may infer that there exist proofs Q_i in R of $v_i\sigma \approx w_i\sigma$, such that

$$(*) \qquad \{(\{s\}, \bot, \{\})\} >^{ccc}_{mul} \{(\{v_i\sigma, w_i\sigma\}, \bot, \{\})\} >^{ccc}_{mul} c^p_{\mathcal{CC}}(Q_i),$$

for $1 \leq i \leq n$.

Now, let P be the proof consisting of the proof step $\dfrac{Q_1 \ldots Q_n}{s \to_C^{p,\sigma} t}$. The complexity of P is

$$c_{\mathcal{CC}}^p(P) = \{(\{s\}, v, \{w\} \cup \mathcal{M}(\Delta))\} \cup c_{\mathcal{CC}}^p(Q_1) \cup \ldots \cup c_{\mathcal{CC}}^p(Q_n).$$

Using $(*)$ we know that $(\{s\}, v, \{w\} \cup \mathcal{M}(\Delta))$ is maximal in $c_{\mathcal{CC}}^p(P)$. Hence we have

$$\{(\{s, \bot\}, \bot, \{\})\} >_{mul}^{ccc} c_{\mathcal{CC}}^p(P)$$

and if $s \rhd v$

$$\{(\{s\}, s, \{\})\} >_{mul}^{ccc} c_{\mathcal{CC}}^p(P).$$

Ad (ii): In order to show the second part of the Lemma we apply (i) to each rewrite step in $s \to_R^* u \leftarrow_R^* t$. Since all rules in R are simplifying (and therefore terms get smaller by rewriting) there exist proofs P_1 and P_2 in R of $s \approx u$ and $u \approx t$ such that

$$\{(\{s, \bot\}, \bot, \{\})\} >_{mul}^{ccc} c_{\mathcal{CC}}^p(P_1)$$
$$\{(\{t, \bot\}, \bot, \{\})\} >_{mul}^{ccc} c_{\mathcal{CC}}^p(P_2).$$

The proof $P = (P_1, P_2)$ is a proof in R of $s \approx t$ with

$$\{(\{s, t\}, \bot, \{\})\} >_{mul}^{ccc} c_{\mathcal{CC}}^p(P).$$

□

LEMMA 3.18. *The ordering $>_{\mathcal{CC}}$ is a proof reduction ordering and compatible with the inference system $\mathcal{CC}$.*

PROOF. Using Lemma 3.17 the proof is similar to the proof of Lemma 3.10. □

4.2. Correctness of $\mathcal{CC}$. There also exists a Critical Pair Lemma for conditional equations.

LEMMA 3.19 (**Contextual Critical Pair Lemma**). *If P is a peak $s \leftarrow_R u \to_R t$, then either there exists a simpler proof Q of $s \approx t$ or, for some contextual critical pair $\Delta \Rightarrow v \approx w$ between rules in R a proof of the form $s \leftrightarrow_{v \approx w}^{p,\sigma} t$. In the case of unconditional rewriting of u the proof Q is a rewrite proof.*

PROOF. For a proof we refer to [**Ganzinger, 1991**]. (Although Ganzinger uses a different proof ordering, the proof can easily be adapted to $>_{\mathcal{CC}}$.) □

We can now give a definition of a fair derivation in conditional standard completion (which coincides with the definition of a fair derivation in equational standard completion).

DEFINITION 3.20. *A non-failing derivation $(E_0, R_0) \vdash_{\mathcal{EC}} (E_1, R_1) \vdash_{\mathcal{EC}} \cdots$ in $\mathcal{CC}$ is called* fair *if the set of all contextual critical pairs between rules in R_∞ is a subset of $\bigcup_j E_j$.*

LEMMA 3.21. *Let $(E_0, R_0) \vdash_{\mathcal{CC}} (E_1, R_1) \vdash_{\mathcal{CC}} \cdots$ be a non-failing, fair derivation and let P be a proof in (E_i, R_i) for $s \approx t$, for some i. If P is not a rewrite proof, then there exists, for some $j \geq i$, a proof Q in (E_j, R_j) for $s \approx t$ such that $P >_{\mathcal{CC}} Q$.*

PROOF. [**Ganzinger, 1991**] □

THEOREM 3.22. *If a non-failing derivation in basic equational completion is fair, then the limit rewrite system R_∞ is convergent.*

PROOF. The proof is a direct consequence of Lemma 3.21 and similar to the one given for unconditional equations (see Theorem 3.14). □

COROLLARY 3.23. *A terminating rewrite system R is Church–Rosser if all critical pairs deduced from R are joinable.*

Unlike in the equational case, the other direction of the implication in Corollary 3.23 does not hold. An equivalent characterization of convergence by critical pairs is given in Proposition 3.24.

PROPOSITION 3.24. *Given a set R of simplifying conditional rules. The rewrite system R is convergent if and only if for all critical pairs $v_1 \approx w_1, \ldots, v_n \approx w_n \Rightarrow v \approx w$ between rules in R and for all substitutions σ it holds that*

$$v\sigma \downarrow_R^* w\sigma, \ \textit{whenver } v_i\sigma \downarrow_R^* w_i\sigma, \ \textit{for all } 1 \leq i \leq n.$$

PROOF. see [**Kaplan, 1987**] or [**Jouannaud and Waldmann, 1987**]. □

4.3. A Conditional Completion Procedure. The completion procedure in [**Jouannaud and Waldmann, 1987**] is an example of a completion procedure for $\mathcal{CC}$ which produces fair derivations. Note, that this procedure does not use the inference rule "Simplification of condition in equation", but tries to check whether a conditional rule is solvable or not. A rule $s_1 \approx t_1, \ldots, s_n \approx t_n \Rightarrow s \rightarrow t$ is called solvable if there exists a substitution σ such that $s_i\sigma \downarrow_R^* t_i\sigma$, for all $1 \leq i \leq n$. Hence, this procedure uses Proposition 3.24 for checking if the rewrite system R is convergent.

given: A set E of conditional equations.
find: A convergent set R of conditional rules such that $\leftrightarrow_E^* = \leftrightarrow_R^*$.

begin
  $input(E)$;
  $R := Completion(\emptyset, E, \emptyset, 0)$;
  $output(R)$;
end

function $Completion(E_1, E_2, R, n)$

if $E_1 \neq \emptyset$ **then**
  select a conditional equation $\Gamma \Rightarrow s \approx t$ in E_1.
  $s' := Normalize(s)$;
  $t' := Normalize(t)$;
  if $s' = t'$ **then**
    $Completion(E_1 \setminus \{\Gamma \Rightarrow s \approx t\}, E_2, R, n)$;
  else
    $\Gamma \Rightarrow s'' \to t'' := Orient(\Gamma \Rightarrow s' \approx t')$;
    $R := R \cup \{n : \Gamma \Rightarrow s'' \to t''\}$;
    $K := \{k : \Delta \Rightarrow v \to w \mid k : \Delta \Rightarrow v \to w \in R, v$ is reducable w.r.t. $R \setminus \{k : \Delta \Rightarrow v \to w\}\}$;
    $E_1 := E_1 \setminus \{\Gamma \Rightarrow s \approx t\} \cup \{\Delta \Rightarrow v' \approx w \mid k : \Delta \Rightarrow v \to w \in K, v' := Normalize(v)\}$;
    $R := \{k : \Delta' \Rightarrow v \to w' \mid k : \Delta \Rightarrow v \to w \in R \setminus K, \Delta' := Normalize(\Delta), w' := Normalize(w)\}$;
    $Completion(E_1, E_2, R, n + 1)$;
  fi
elseif $\exists\, j : \Gamma \Rightarrow s \to t \in R$ unmarked **then**
  mark $j : \Gamma \Rightarrow s \to t$.
  $E_2 := E_2 \cup$ set of all contextual critical pairs of $j : \Gamma \Rightarrow s \to t$ and $k : \Lambda \Rightarrow u \to v \in R$, with $k \leq j$.
  $Completion(E_1, E_2, R, n)$;
elseif $(E_2 \neq \emptyset)$ **then**
  while $E_2 \neq \emptyset$ and $E_1 = \emptyset$ **do**
    $bound := bound + 1$;
    for all $\Delta \Rightarrow v \approx w$ **in** E_2 **do**
      if $Solvable(\Delta, R, bound)$ **then**
        $E_2 := E_2 \setminus \{\Delta \Rightarrow v \approx w\}$;
        $E_1 := E_1 \cup \{\Delta \Rightarrow v \approx w\}$;
      fi
    od
  od
  $Completion(E_1, E_2, R, n)$;
else
  return(R) %success
fi

function $Solvable(\Pi, R, bound)$

$F := [\,]$;
$push(\Pi, F)$;
for $i := 1$ **to** *bound* **do**
  $\Lambda := pop(F)$;
  if $\exists\sigma$ ($\Lambda\sigma$ is true) **then**
    return(true);
  else
    for all $\Gamma \Rightarrow s \approx t$ **in** R **do**
      for all p **in** $\{q \mid \exists\sigma\ (\Lambda|_q\sigma = s\sigma)\}$ **do**
        $push((\Lambda[t]_p \wedge \Gamma)\sigma, F)$;
      od
    od
    if $(F = [\,])$ **then return**(failure1); **fi**
  fi
od
return(failure2);

The sets E_1, E_2 and R have the following meaning:

- E_1 the set of solvable equations. Equations which are solvable are moved from E_2 to E_1.
- E_2 the set of yet unsolved equations; we do not know it if they are solvable or not. Critical pairs are added to E_2.
- R the set of rules.

The completion procedure leads to one of the following situations:

- It terminates with a convergent rewrite system R which is equivalent to the given set E of equations.
- It fails inside the orientation procedure *Orient*.
- It generates infinitely many rules.
- It runs forever with unsolvable contextual critical pairs in E_2 and an empty E_1. In this case the finite set of rules R is convergent (but we do not know it).

Practical experience shows that the completion procedure fails unfortunately almost always because a non-simplifying equation is generated.

A completion procedure which can, to some extent, also handle non-simplifying rules has been proven correct by Ganzinger in [**Ganzinger, 1991**].

CHAPTER 4

Ordered Completion

Consider the following examples of Standard Completion:
Suppose we want to transfer the set of equations

$$
\begin{aligned}
a &\approx b \\
a &\approx c \\
f(b) &\approx b \\
f(a) &\approx d
\end{aligned}
$$

into a convergent set of rules by the inference system $\mathcal{EC}$.

Let $\succ$ be the lexicographic path ordering corresponding to the precedence $f \succ a \succ b \succ d$ and $f \succ a \succ c \succ d$. A possible derivation is

E_i	R_i
$\{a \approx b, a \approx c, f(b) \approx b, f(a) \approx d\}$	$\{\}$
$\{b \approx c, f(b) \approx b, f(b) \approx d\}$	$\{a \to b\}$
$\{b \approx c, b \approx d\}$	$\{a \to b, f(b) \to b\}$
$\{d \approx c, f(d) \approx d\}$	$\{a \to d, b \to d\}$
$\{f(d) \approx d\}$	$\{a \to d, b \to d, c \to d\}$
$\{\}$	$\{a \to d, b \to d, c \to d, f(d) \to d\}$.

Another derivation (at the beginning another equation is picked for Orientation) is

E_i	R_i
$\{a \approx b, a \approx c, f(b) \approx b, f(a) \approx d\}$	$\{\}$
$\{c \approx b, f(b) \approx b, f(c) \approx d\}$	$\{a \to c\}$
$\{c \approx b, f(c) \approx d\}$	$\{c \to b, f(b) \to b\}$
$\{c \approx b\}$	$\{a \to c, f(c) \to d, f(b) \to b\}$.

Now, $\mathcal{EC}$ fails because $c \approx b$ can neither be simplified nor oriented. Thus in Standard Completion failure or success can depend on the order how equations are picked for orientation, superposition or rewriting.
(Dershowitz, Marcus and Tarlecki 1988)

Another example, where Standard Completion fails, regardless of which reduction ordering is chosen or in which way equations are picked, is:

Let E be the set of equations

$$\begin{aligned} 1 * ((-x) + x) &\approx 0 \\ 1 * (x + (-x)) &\approx x + (-x) \\ (-x) + x &\approx y + (-y) \end{aligned}$$

There is no other choice than to orient the first and second equation into the following rules (no other orientation is possible):

$$\begin{aligned} 1 * ((-x) + x) &\to 0 \\ 1 * (x + (-x)) &\to x + (-x) \end{aligned}$$

The third equation is irreducible and unorientable, hence Standard Completion fails.

Nonetheless there exists a convergent rewrite system for E:

$$\begin{aligned} (-x) + x &\to 0 \\ x + (-x) &\to 0 \\ 1 * 0 &\to 0 \end{aligned}$$

(Bachmair 1991).

Unfortunately in the conditional case it is much more likely that Standard Completion fails because of encountering a conditional equation which cannot be oriented into a conditional simplifying rule than in the unconditional case.

A specification of the 'less or equal' - predicate for integers may look like:

$$\begin{array}{ll} e_1 & s(p(x)) \approx x \\ e_2 & p(s(x)) \approx x \\ e_3 & le(0,0) \approx t \\ e_4 & le(0,p(0)) \approx f \\ e_5 & le(0,x) \approx t \Rightarrow le(0,s(x)) \approx t \\ e_6 & le(0,x) \approx f \Rightarrow le(0,p(x)) \approx f \\ e_7 & le(s(x),y) \approx le(x,p(y)) \\ e_8 & le(p(x),y) \approx le(x,s(y)) \end{array}$$

The reduction ordering is the lexicographic recursive path ordering based on the precedence $le \succ p \succ s \succ 0 \succ t \succ f$. All equations $e_1 - e_8$ can be oriented into rules and there are two critical pairs:

- Between e_1 and e_5 (with overlapped term $le(0, s(p(x)))$)

$$e_9 \quad le(0,p(x)) \approx t \Rightarrow \ le(0,x) \approx t \qquad [e_1, e_5].$$

- Between e_2 and e_6 (with overlapped term $le(0, p(s(x)))$)

$$e_{10} \quad le(0,s(x)) \approx f \Rightarrow \ le(0,x) \approx f \qquad [e_2, e_6].$$

These critical pairs can neither be simplified nor oriented into simplifying rules because the conditions are greater than the succedent. Therefore, Standard Completion will fail, although the rewrite system consisting of the oriented equations $e_1 - e_8$ is convergent.

We will show in this chapter how all these problems can be solved by the technique of Ordered Completion. In Ordered Completion we do no longer distinguish between simplifying rules and equations. Instead of orienting a simplifying equation into a rule and using this rule for superposition and rewriting, certain ordering constraints are used. Hence, Ordered Completion cannot fail because of encountering a non-simplifying equation.

In general, the limit set of a fair Ordered Completion derivation is no longer convergent, but only *ground convergent*, i.e. convergent on the set of all ground terms.

Another advantage of Ordered Completion is that it can also deal with Horn clauses and not only with conditional equations. We will prove that Ordered Completion is refutationally complete, so we can use it for *refutational theorem proving*.

Let N be a set of Horn clauses and s and t be terms. Suppose we want to check whether $N \models s \approx t$ or not. For this purpose we take the Skolemized terms $\hat{s}$ and $\hat{t}$ of s and t and a complete reduction ordering $\succ$ on $\mathcal{T}(\mathcal{F} \cup \mathcal{C}, \mathcal{V})$, where $\mathcal{C}$ is the set of Skolem constants occuring in $\hat{s}$ and $\hat{t}$. Let N_∞ be the limit of a fair Ordered Completion derivation starting from N. If $N \models s \approx t$ then also $N \models \hat{s} \approx \hat{t}$ and therefore it exists a rewrite proof in N_∞ of $\hat{s} \approx \hat{t}$, since N_∞ is ground convergent. On the other hand, if there exists a rewrite proof in N_∞ of $\hat{s} \approx \hat{t}$, then $N \models s \approx t$. So we have, $N \models s \approx t$ if and only if the normal forms of $\hat{s}$ and $\hat{t}$ are the same.

To summarize, the differences of Standard Completion and Ordered Completion are

- Ordered Completion can also handle non-simplifying equations; therefore it is unfailing.
- Ordered Completion works also for Horn clauses (and not just for conditional equations) and can be used for refutational theorem proving.
- The limit set of Ordered Completion is in general not convergent but only ground convergent. In Section 6 we will show under which circumstances Ordered Completion will even generate convergent sets.

1. Equation and Clause orderings

In Standard Completion the correctness proofs were based on the notion of proof-orderings. In ordered completion we will deal with clause-orderings.

Let $\succ$ be an ordering on $\mathcal{T}(\mathcal{F},\mathcal{V})$. This ordering can be extended to an ordering $\succ^c$ on clauses over $\mathcal{T}(\mathcal{F},\mathcal{V})$ as follows:

We have defined a clause $\Gamma \Rightarrow \Phi$ as a pair of multisets of equations. Alternatively, we can represent the clause as a multiset of *occurrences of equations*. An occurrence of an equation $s \approx t$ in the antecedent Γ is identified with the multiset $\{\{s,t\}\}$, whereas the occurrence of $s \approx t$ in the succedent Φ is identified with $\{\{s\},\{t\}\}$.

The twofold multiset ordering $\succ^o = (\succ_{mul})_{mul}$ is an ordering on occurrences of equations. An ordering $\succ^c$ on clauses can be obtained by building the multiset ordering of $\succ^o$ (which is the threefold ordering $((\succ_{mul})_{mul})_{mul}$ of $\succ$). As mentioned in Section 2 of Chapter 2, the ordering $\succ^c$ is complete and/or well-founded if $\succ$ is a complete and/or well-founded ordering.

To illustrate the definitions above let s,t,u,v,w be terms and assume $s \succ t \succ u$, $v \succ w \succ u$. We have the following correspondence:

$$\begin{aligned}
v \approx w \Rightarrow s \approx t \quad &\triangleq \quad \{\{\{v,w\}\},\{\{s\},\{t\}\}\},\\
v \approx w \Rightarrow s \approx u \quad &\triangleq \quad \{\{\{v,w\}\},\{\{s\},\{u\}\}\},\\
v \approx u \Rightarrow s \approx t \quad &\triangleq \quad \{\{\{v,u\}\},\{\{s\},\{t\}\}\},\\
v \approx w, s \approx u \Rightarrow \quad &\triangleq \quad \{\{\{v,w\}\},\{\{s,u\}\}\}.
\end{aligned}$$

Therefore the clauses are ordered in the following way:

$$\begin{aligned}
v \approx w \Rightarrow s \approx t \quad &\succ^c \quad v \approx w \Rightarrow s \approx u,\\
v \approx w \Rightarrow s \approx t \quad &\succ^c \quad v \approx u \Rightarrow s \approx t,\\
v \approx w, s \approx u \Rightarrow \quad &\succ^c \quad v \approx w \Rightarrow s \approx t.
\end{aligned}$$

DEFINITION 4.1. *Let $\succ$ be a reduction ordering. A Horn clause $C = \Gamma \Rightarrow s \approx t$ is called* reductive *if $t \not\succeq s$ and $s \approx t$ is a strictly maximal occurrence of an equation in C.*

Note, if $\succ$ is complete the definition of "simplifying" and "reductive" coincide for ground Horn clauses.

An equivalent definition of "reductive" is:
A Horn clause $C = \Gamma \Rightarrow s \approx t$ is reductive if

$$(s \succ v \wedge s \succ w) \vee (t \succ v \wedge t \succ w) \qquad \text{, for all } v \approx w \in \Gamma.$$

If $C = \Gamma \Rightarrow s \approx t$ is reductive, the term s must not occur in Γ. It is easy to see that reductive Horn clauses for which in addition $s \succ t$ holds are exactly simplifying conditional equations.

2. The Inference System $\mathcal{OC}_S$ for Ordered Completion

We will define the inference system for Ordered Completion in a slightly different way than the inference systems for Standard Completion. In Standard Completion we had deduction, simplification and deletion rules. In Ordered Completion, in a first step, we are only interested in refutational completeness and we will therefore only consider deduction rules.

DEFINITION 4.2. *An* inference rule π (for Ordered Completion) *has the form*

$$\frac{C_1 \cdots C_n}{C}$$

where C_1*,...,* C_n*, the premise, and* C*, the conclusion, are clauses. A set of inference rules for Ordered Completion is called an* ordered inference system*.*

For notational reasons we will write an inference rule $\dfrac{C_1 \cdots C_n}{C}$ also as $(C_1, \dots, C_n : C)$.

Let $\succ$ be a complete reduction ordering. The inference system $\mathcal{OC}$ for *Ordered Completion for Horn Clauses* consists of the following inference rules:

Superposition, right: $$\frac{\Gamma \Rightarrow s \approx t \quad \Delta \Rightarrow v[s'] \approx w}{\Gamma\sigma, \Delta\sigma \Rightarrow v[t]\sigma \approx w\sigma}$$

if

(i) $\sigma = \mathrm{mgu}(s, s')$,
(ii) $\Gamma\sigma \Rightarrow s\sigma \approx t\sigma$ is reductive,
(iii) $\Delta\sigma \Rightarrow v\sigma \approx w\sigma$ is reductive, and
(iv) s' is not a variable.

Superposition, left: $$\frac{\Gamma \Rightarrow s \approx t \quad v[s'] \approx w, \Delta \Rightarrow \Psi}{\Gamma\sigma, v[t]\sigma \approx w\sigma, \Delta\sigma \Rightarrow \Psi\sigma}$$

if

(i) $\sigma = \mathrm{mgu}(s, s')$,
(ii) $\Gamma\sigma \Rightarrow s\sigma \approx t\sigma$ is reductive,
(iii) $w\sigma \not\succeq v\sigma$ and $v\sigma \approx w\sigma$ is a maximal occurrence in $v\sigma \approx w\sigma, \Delta\sigma \Rightarrow \Psi\sigma$, and
(iv) s' is not a variable.

Equality Resolution: $$\frac{v \approx w, \Gamma \Rightarrow \Phi}{\Gamma\sigma \Rightarrow \Phi\sigma}$$

if

(i) $\sigma = \mathrm{mgu}(v, w)$ and
(ii) $v\sigma \approx w\sigma$ is a maximal occurrence in $v\sigma \approx w\sigma, \Gamma\sigma \Rightarrow \Phi\sigma$.

When applying a superposition rule it is assumed that the premises have no variables in common; if necessary the variables of one clause have to be renamed.

An essential property of the inference system $\mathcal{OC}$ is that the conclusion of a ground inference is always simpler than the maximal premise.

In order to reduce the number of clauses which can be deduced even further we introduce, besides the ordering constraints used above, a *selection function.*

DEFINITION 4.3. *A* selection function S *is a function from the set of clauses to the set of multisets of equations such that* $S(\Gamma \Rightarrow \Phi) \subseteq \Gamma$, *for all clauses* $\Gamma \Rightarrow \Phi$. *We call an equation* $s \approx t \in \Gamma$ selected *if* $s \approx t \in S(\Gamma \Rightarrow \Phi)$.

The following two inference rules are defined with respect to a selection function S.

Selective Superposition, left: $$\frac{\Gamma \Rightarrow s \approx t \quad v[s'] \approx w, \Delta \Rightarrow \Psi}{v[t]\sigma \approx w\sigma, \Gamma\sigma, \Delta\sigma \Rightarrow \Psi\sigma}$$

if

(i) $\sigma = \mathrm{mgu}(s, s')$,
(ii) $\Gamma\sigma \Rightarrow s\sigma \approx t\sigma$ is reductive,
(iii) $\Gamma \Rightarrow s \approx t$ contains no selected equation,
(iv) $w\sigma \not\succeq v\sigma$ and $v \approx w$ is a selected equation, and
(v) s' is not a variable.

Selective Resolution: $$\frac{\Gamma, v \approx w \Rightarrow \Phi}{\Gamma\sigma \Rightarrow \Phi\sigma}$$

if

(i) $\sigma = \mathrm{mgu}(v, w)$ and
(ii) $v \approx w$ is a selected equation.

By $\mathcal{OC}_S$ we denote the inference system consisting of all rules of $\mathcal{OC}$ plus Selective Superposition and Selective Resolution with the additional restriction that rules of $\mathcal{OC}$ must not be applied to Horn clauses with selected equations.
We will show in Section 3 refutational completeness of $\mathcal{OC}_S$ for an arbitrarily defined selection function S.

Inference systems resulting from special kinds of selection functions are the following ones:

- In the case where $S(C) = \{\}$, for all clauses C, the inference systems $\mathcal{OC}_S$ and $\mathcal{OC}$ are the same. An inference system for first-order clauses with equality of this kind has been proven refutationally complete in [**Bachmair and Ganzinger, 1990b**].
- If $S(C)$ consist of at most one equation, for all clauses C, the selection function is called *unitary*.
- A selection function S for which $S(C)$ is non-empty whenever the antecedent of a clause C is non-empty yields a so called *positive* selection function. In positive strategies only equations are superposed into other clauses. Dershowitz in [**Dershowitz and Okada, 1990**] described such an inference system, which he called "Maximal-Literal Unit Strategy". In this inference system the complete antecedent is selected and ordering constraints are imposed on selected equations.
- An inference system based on a combination of the above selection functions, a so called *positive unitary* selection function, is described in [**Nieuwenhuis and Orejas, 1992**].

3. Refutational Completeness of $\mathcal{OC}_S$

In this section refutational completeness of $\mathcal{OC}_S$ is proven in the same way as described in [**Bachmair and Ganzinger, 1994**]. In this article Bachmair and Ganzinger proved refutational completeness of an inference system (which also includes one of the inference rules "Equality Factoring" or "Merging Paramodulation") for first-order clauses with equality. This means that refutational completeness of $\mathcal{OC}_S$ is a consequence of refutational completeness of their inference system. However, by restricting ourselves to Horn logic, our proofs for the inference system $\mathcal{OC}_S$ become shorter and simpler.

The missing proofs for Lemma 4.6, Lemma 4.16 and Lemma 4.18 can be found in [**Bachmair and Ganzinger, 1994**].

DEFINITION 4.4. *Let $\mathcal{I}$ be an ordered inference system and $\pi = (C_1, \ldots, C_n : C)$ be an inference in $\mathcal{I}$, where the clauses C_1,..., C_n have no variables in common (if necessary some variables have to be renamed). By an* instance *of π we mean any inference in $\mathcal{I}$ with premises $C_1\sigma$,..., $C_n\sigma$ and conclusion $C\sigma$, where σ is some substitution.*

Let N be a set of Horn clauses. For Equality Resolution and Selected Resolution any inference from ground instances of N is a ground instance of an inference from clauses of N. This need not be true for arbitrary superposition inferences but only for instances from reduced clauses.

DEFINITION 4.5. *An instance $C\sigma$ of a clause C is* reduced *with respect to a rewrite system R if $x\sigma$ is irreducible by R, for all x occuring in C.*

LEMMA 4.6. *Let C and D be clauses with no variables in common. Let σ be a substitution such that $C\sigma = \Gamma \Rightarrow s \approx t$, $D\sigma$ are ground instances, $D\sigma \succ^c C\sigma$, $C\sigma$ is reductive and $D\sigma$ is reduced with respect to $\{s \rightarrow t\}$. Then any superposition inference with premises $C\sigma$ and $D\sigma$ is a ground instance of a similar inference from C and D.*

3.1. Equality Interpretation. Let N be a set of Horn clauses and $\succ$ be a complete reduction ordering. In order to prove (or disprove) the satisfiability of N we define an interpretation $\mathfrak{I}$ with the help of a convergent ground rewrite system R (see Section 5 of Chapter 2). We will use induction on the clause ordering $\succ^c$ to define sets of equations E_C, sets of rules R_C and interpretations $\mathfrak{I}_{R_C}$, for all ground instances C of clauses of N.

Let $C = \Gamma \Rightarrow s \approx t$ be a ground instance of a clause of N and suppose that $R_{C'}$ and $E_{C'}$ have been defined for all ground instances C' of clauses of N, for which $C \succ^c C'$. We define

$$R_C = \bigcup_{C \succ^c C'} E_{C'}$$

$$E_C = \begin{cases} \{s \rightarrow t\} & \text{, if } C \text{ is productive;} \\ \emptyset & \text{, otherwise.} \end{cases}$$

Where a ground clause C is *productive* if and only if

(i) C is reductive (or simplifying, since $\succ$ is complete),
(ii) s is irreducible by R_C and

(iii) $\mathfrak{I}_{R_C} \models \Gamma$.

Further, let $R = \bigcup_C E_C$.

Note, that the sets R_C are constructed in such a way that they only contain left-reduced rules (i.e. for every rule $s \to t \in R_C$ the term s is irreducible by $R_C \setminus \{s \to t\}$). Since ground rewrite systems which are left-reduced and terminating are convergent, the interpretations $\mathfrak{I}_{R_C}$ are well-defined.

A basic property of productive clauses is that they are false in $\mathfrak{I}_{R_C}$ but true in $\mathfrak{I}_{(R_C \cup E_C)}$.

LEMMA 4.7. *Given a set N of Horn clauses.*

(i) *Let $C = \Gamma \Rightarrow s \approx t$ be a reductive clause and D be another clause containing s. If $C \succ^c D$ and s is irreducible by R_C, then $R_C = R_D$.*
(ii) *Let $C = \Gamma \Rightarrow s \approx t$ and D be ground instances of clauses of N with $D \succeq^c C$. If $\mathfrak{I}_{R_C} \models s \approx t$ then $\mathfrak{I}_{R_D} \models s \approx t$ and $\mathfrak{I}_R \models s \approx t$.*
(iii) *Let $C = \Gamma', s \approx t \Rightarrow \Phi$ and D be ground instances of clauses of N with $D \succeq^c C$. Then, $\mathfrak{I}_{R_C} \models s \approx t$ if and only if $\mathfrak{I}_{R_D} \models s \approx t$ if and only if $\mathfrak{I}_R \models s \approx t$.*
(iv) *Let C and D be ground instances of clauses of N with $D \succeq^c C$. If C is true in $\mathfrak{I}_{R_C}$ then it is also true in $\mathfrak{I}_{R_D}$ and $\mathfrak{I}_R$.*

PROOF. Ad (i): Let C' be a clause with $C \succ^c C' \succeq^c D$. $E_{C'} = \emptyset$ because otherwise s would be reducible by R_C. Hence, $R_C = R_D \cup \bigcup_{C \succ^c C' \succeq^c D} E_{C'} = R_D$.

Ad (ii): If $\mathfrak{I}_C \models s \approx t$ then $s \downarrow^!_{R_C} t$. Since $R_C \subseteq R_D \subseteq R$ it is clear that also $s \downarrow^!_{R_D} t$ and $s \downarrow^!_R t$, which is equivalent to $\mathfrak{I}_{R_D} \models s \approx t$ and $\mathfrak{I}_R \models s \approx t$.

Ad (iii): First, assume that $s \approx t$ is true in $\mathfrak{I}_{R_C}$. By the same argumentation as above it follows that $s \approx t$ is also true in $\mathfrak{I}_{R_D}$ and $\mathfrak{I}_R$.

In the other hand, if $s \approx t$ is false in $\mathfrak{I}_{R_C}$ the normal forms $s' = \mathrm{nf}_{\to_{R_C}}(s)$ and $t' = \mathrm{nf}_{\to_{R_C}}(t)$ have to be distinct. For all rules $v \to w$ in $R \setminus R_C$ it holds that $v \succ s \succ s'$ and $v \succ t \succ t'$, since clauses which produce rules with terms not greater than s or t are smaller than C. Hence, s' and t' are not only normal forms of s and t with respect to R_C but are also normal forms with respect to R, which implies that $s \approx t$ is false in $\mathfrak{I}_R$ and $\mathfrak{I}_{R_D}$.

Ad (iv): Follows immediately from (ii) and (iii). □

3.2. Redundancy and Saturation. We will prove that the interpretation $\mathfrak{I}$ constructed in the previous section is a model of N, provided that N is satisfiable and saturated.

DEFINITION 4.8. *A ground instance C of N is called* redundant *if it is true in $\mathfrak{I}_{R_C}$. A clause is called redundant if all its ground instances are redundant.*

One can think of redundant clauses as clauses which do not contribute to the model $\mathfrak{I}$ of N, i.e. non-productive clauses. The definition of redundancy can be extended to inferences as follows.

DEFINITION 4.9. *An inference* $\pi = (C_1, \ldots, C_n : C)$, *where* $C_1, \ldots, C_n$ *are ground instances of* N *is said to be* redundant *if either some* C_i *is redundant or* C *is true in* $\Im_{R_D}$, *where* D *is the maximum of* $C_1, \ldots, C_n$. *Again, an inference is redundant if all its ground instances are redundant.*

DEFINITION 4.10. *A set* N *of Horn clauses is said to be* saturated on N' *if every ground instance of an inference from* N, *the premises of which are in* N', *is redundant. We call* N *simply saturated if* N *is saturated on* N.

In the following we will use N_C to denote the set of all ground instances C' of clauses of N for which $C \succ^c C'$ holds. Lemma 4.11 characterizes productive clauses of saturated sets.

LEMMA 4.11. *Let* N *and* N' *be sets of Horn clauses and let* N *be saturated on* N' *(with respect to some inference system* $\mathcal{OC}_S$*). Further, let* C *be a productive ground instance of some clause of* N *with* $(N_C \cup \{C\}) \subseteq N'$. *Then the following holds*

(i) C *is non-redundant,*
(ii) C *is a reduced ground instance, and*
(iii) C *contains no selected equations.*

PROOF. We will prove this lemma by induction on the clause ordering $\succ^c$. Suppose N is saturated on N' and let $C = \Gamma \Rightarrow s \approx t$ be a productive ground instance of N, such that $(N_C \cup \{C\}) \subseteq N'$. From the definition of "productive" it follows that C is simplifying, s is irreducible by R_C, $\Im_{R_C} \models \Gamma$ and $\Im_{R_C} \not\models s \approx t$. Therefore C is non-redundant.

In order to prove that $C = D\sigma$ is a reduced ground instance, we assume that there exists a variable x occuring in D such that $x\sigma$ is reducible to u by R_C. We define the substitution τ as:

$$y\tau = \begin{cases} u & \text{, if } y = x; \\ y\sigma & \text{, otherwise.} \end{cases}$$

Let $C' = D\tau$. Since s is irreducible C' must be of the form $\Gamma' \Rightarrow s \approx t'$. Because $C \succ^c C'$ and $\Im_{R_C} \models \Gamma'$ we can infer by Lemma 4.7 that $R_{C'} = R_C$, which implies that $\Im_{R_C} = \Im_{R_{C'}}$. Putting all together, we have shown that C' is simplifying, s is irreducible by $R_{C'}$ and $\Im_{R_{C'}} \models \Gamma'$. Therefore C' is a productive ground instance and produces $s \to t'$. But this is a contradiction to the irreducibility of s by R_C, since $\{s \to t'\} = E_{C'} \subseteq R_C$.

From now on let us assume that C is a reduced ground instance (with respect to R_C).

To show that C contains no selected equations we assume that C is of the form $C = \Gamma', s' \approx t' \Rightarrow s \approx t$, where $s' \approx t'$ is a selected equation, and we try to derive a contradiction to $\Im_{R_C} \not\models s \approx t$. We distinguish the following three cases:

(i) If $s' = t'$, by selective resolution inference

$$\frac{\Gamma', s' \approx t' \Rightarrow s \approx t}{\Gamma' \Rightarrow s \approx t}.$$

we can obtain the clause $C' = \Gamma' \Rightarrow s \approx t$. From our assumption that N is saturated on N' and the fact that $C \succ^c C'$ it follows that C' has to be redundant. Thus C' has to be true in $\Im_{R_C}$. Because $\Im_{R_C} \models \Gamma$ (and therefore $\Im_{R_C} \models \Gamma'$) we get the contradiction that $\Im_{R_C} \models s \approx t$.

(ii) If $s' \succ t'$, we know that s' is reducible by R_C (s' and t' must have the same normal form because of $\Im_{R_C} \models s' \approx t'$). Let $v \to w$ be a rule which reduces s' and let $C' = \Delta \Rightarrow v \approx w$ be the corresponding clause which produces $v \to w$. Since $C \succ^c C'$ we can apply the induction hypothesis and infer that C' contains no selected equations. Consider the selective superposition inference

$$\frac{\Delta \Rightarrow v \approx w \quad \Gamma', s'[v] \approx t' \Rightarrow s \approx t}{\Delta, \Gamma', s'[w] \approx t' \Rightarrow s \approx t}.$$

Because C is a reduced ground instance, we know by Lemma 4.6, that this inference is a ground instance of an inference from N. From the assumption that N is saturated on $N_C \cup \{C\}$ we can conclude that $\Delta, \Gamma, s'[w] \approx t' \Rightarrow s \approx t$ has to be redundant. Because C' is productive we have $\Im_{R_{C'}} \models \Delta$ which implies $\Im_{R_C} \models \Delta$. From $\Im_{R_C} \models \Gamma$ it follows that $\Im_{R_C} \models \Gamma', s'[w] \approx t'$. Together, we derive the contradiction that $\Im_{R_C} \models s \approx t$.

(iii) The case where $t' \succ s'$ can be proven equally to the case where $s' \succ t'$.

□

LEMMA 4.12. *Let N and N' be a sets of Horn clauses. If N is saturated on N' and does not contain the empty clause then all ground instances C of clauses of N with $(N_C \cup \{C\}) \subseteq N'$ are true in $\Im_{(R_C \cup E_C)}$.*

PROOF. Suppose N is saturated on N' and does not contain the empty clause. Let $C = \Gamma \Rightarrow \Phi$ be a ground instance of some clause of N with $(N_C \cup \{C\}) \subseteq N'$.

If C is redundant then C is true in $\Im_{R_C}$ and hence also true in $\Im_{(R_C \cup E_C)}$.

If C is productive then C is false in $\Im_{R_C}$ but true in $\Im_{(R_C \cup E_C)}$.

Therefore, let us assume that C is neither redundant nor productive, i.e. $\Im_{R_C} \models \Gamma$, $\Im_{R_C} \not\models \Phi$ and $E_C = \emptyset$.

We can restrict ourselves to the case where C is a reduced ground instance of a clause of N (with respect to R_C). Because if $C = D\sigma$ would not be a reduced ground instance, i.e. $x\sigma$ is reducible to u, for some variable x in D, C is true in R_C if and only if $D\tau$ is true in R_C, where the substitution τ is defined as: $x\tau = u$ and $y\tau = y\sigma$, for all $y \neq x$.

We distinguish the following cases and show that each one contradicts with $\Im_{R_C} \not\models \Phi$:

(i) If C is of the form $C = \Gamma', s' \approx s' \Rightarrow \Phi$, where $s' \approx s'$ is either a selected equation or the term s' is maximal in C, by Selective Resolution or Equality Resolution

$$\frac{\Gamma', s' \approx s' \Rightarrow \Phi}{\Gamma' \Rightarrow \Phi}.$$

we obtain the clause $C' = \Gamma' \Rightarrow \Phi$ which has to be true in $\Im_{R_C}$, since N is saturated on N' and $C \succ^c C'$. But this together with $\Im_{R_C} \models \Gamma'$ contradicts that $\Im_{R_C} \not\models \Phi$.

(ii) If C has the form $C = \Gamma', s' \approx t' \Rightarrow \Phi$ with $s' \succ t'$, where $s' \approx t'$ is either a selected equation or the term s' is maximal in C. Because $\mathfrak{I}_{R_C} \models \Gamma', s' \approx t'$ the term s has to reducible by R_C. Let $v \to w$ be a rule which reduces s and let $D = \Delta \Rightarrow v \approx w$ be the corresponding clause that produces $v \to w$. Using Lemma 4.11, since $C \succ^c D$, we know that D contains no selected equations and by selective or left superposition
$$\frac{\Delta \Rightarrow v \approx w \quad \Gamma', s'[v] \approx t' \Rightarrow \Phi}{\Delta, \Gamma', s'[w] \approx t' \Rightarrow \Phi}.$$
we get the clause $C' = \Delta, \Gamma', s'[w] \approx t' \Rightarrow \Phi$ which has to be true in $\mathfrak{I}_{R_C}$. From $\mathfrak{I}_{R_C} \models \Delta$, $\mathfrak{I}_{R_C} \models \Gamma'$ and $\mathfrak{I}_{R_C} \models s'[w]_p \approx t'$ we derive the contradiction that $\mathfrak{I}_{R_C} \models \Phi$.

(iii) If $C = \Gamma \Rightarrow s \approx s$, where s is maximal in C, nothing has to be show.

(iv) Suppose C is of the form $C = \Gamma \Rightarrow s \approx t$ with $s \succ t$, where s is maximal in C. In this case the term s has to reducible, because otherwise C would be productive. Let $v \to w$ be a rule which reduces s and let $D = \Delta \Rightarrow v \approx w$ be the corresponding clause which produces $v \to w$. Since $C \succ^c D$ we can apply Lemma 4.11 and may infer that D contains no selected equations. By right superposition
$$\frac{\Delta \Rightarrow v \approx w \quad \Gamma \Rightarrow s[v] \approx t}{\Delta, \Gamma \Rightarrow s[w] \approx t}.$$
we get the clause $C' = \Delta, \Gamma \Rightarrow s[w] \approx t$. Again, since N is saturated on N' the clause C' has to be true in $\mathfrak{I}_{R_C}$. From $\mathfrak{I}_{R_C} \models \Delta$ and $\mathfrak{I}_{R_C} \models \Gamma$ it follows that $\mathfrak{I}_{R_C} \models s[w]_p \approx t$ which implies the contradiction that $\mathfrak{I}_{R_C} \models s \approx t$.

$\square$

COROLLARY 4.13. *Let N be a saturated set of Horn clauses which does not contain the empty clause. Every non-redundant ground instance of N is productive.*

PROOF. Suppose a clause C is a non-redundant ground instance of N, i.e. $\mathfrak{I}_{R_C} \not\models C$. Using Lemma 4.12 we know that C is true in $\mathfrak{I}_{(R_C \cup E_C)}$, which implies that $E_C \neq \emptyset$. Therefore C is productive. $\square$

THEOREM 4.14. *A saturated set N of Horn clauses is satisfiable if and only if it does not contain the empty clause.*

PROOF. If N contains the empty clause, then N has no model. On the other hand, if N is saturated and does not contain the empty clause, by Lemma 4.12 every ground instance C of N is true in $\mathfrak{I}_{(R_C \cup E_C)}$ and hence true in $\mathfrak{I}_R$. This means that $\mathfrak{I}_R$ is a model of N. $\square$

3.3. Compositeness. The notion of saturation is based on redundancy. In practice it is not possible to check if a clause is redundant or not. So we have to look for sufficient conditions to detect a clause or an inference as a redundant one. In this section we define the compositeness of a clause or inference. This allows us to use almost all simplification and deletion techniques used in automated theorem proving (especially in rewriting) for redundancy-checking.

Definition 4.15. *Let N be a set of Horn clauses and C be a ground clause (which does not have to be a ground instance of N). A clause C is called* composite with respect to N *if there exist ground instances C_1,..., C_k of clauses of N with $C \succ^c C_i$, for all $1 \leq i \leq k$, such that $C_1, \ldots, C_k \models C$.*

A non-ground clause is called composite if all its ground instances are composite.

Lemma 4.16. *Let N be a set of Horn clauses and C be a composite ground instance of some clause of N. If N is saturated on N_C and does not contain the empty clause then C is redundant.*

Definition 4.17. *Let $\pi = (D_1, \ldots, D_n : D)$ be a ground inference with maximal premise D'. The inference π is called* composite with respect to N *if either some premise D_i, is composite, or else there exist ground instances C_1,..., C_k of clauses of N with $D' \succ^c C_i$, for all for all $1 \leq i \leq k$, such that $C_1, \ldots, C_k \models D$.*

A non-ground inference is called composite if all its ground instances are composite.

Lemma 4.18. *Let N be a set of Horn clauses and π be a composite ground instance of an inference from N with maximal premise C. If N is saturated on N_C and does not contain the empty clause then π is redundant.*

4. Ordered Completion Derivations

DEFINITION 4.19. *A sequence $N_0, N_1, \ldots,$ where N_i are sets of Horn clauses, for all $i \in \mathbb{N}$, is an* ordered completion derivation *if either:*

Deduction: $N_{i+1} = N_i \cup \{C\}$, *where* $N_i \models C$.

Deletion: $N_{i+1} = N_i \backslash \{C\}$, *where* C *is a clause in* N_i *which is composite with respect to* N_i.

The set $N_\infty = \bigcup_i \bigcap_{j \geq i} N_j$ *is called the* limit *of the derivation. Clauses in* N_∞ *are called* persisting.

The combination of a deduction-step which triggers a deletion-step is called *Simplification*.

The definition of an ordered completion derivation allows also to add clauses which cannot be obtained by an inference rule of $\mathcal{OC}_S$. The reason is that this may be necessary in order to make another clause composite.

Consider the following example:

Let $N = \{s \approx t, v \approx w\}$, where $s \succ t$, $v \succ w$ and there exists a position p in w such that $w|_p = s$. If we add the equation $v \approx w[t]_t$, which is obtained by rewriting, to N then the equation $v \approx w$ becomes composite and can be deleted.

DEFINITION 4.20. *A derivation $N_0, N_1, \ldots$ is called* fair with respect to $\mathcal{OC}_S$ *if all inferences of* $\mathcal{OC}_S$ *with premises from* N_∞ *are composite with respect to* $\bigcup_i N_i$.

THEOREM 4.21. *Let $N_0, N_1, \ldots$ be a fair derivation with respect to $\mathcal{OC}_S$. If $\bigcup_i N_i$ does not contain the empty clause, then N_∞ is saturated and N_0 is satisfiable.*

PROOF. A proof can be found in [**Bachmair and Ganzinger, 1994**]. □

5. Simplification and Deletion Techniques

In this subsection we discuss the techniques used in the program **HCC** to simplify clauses or to detect clauses as composite.

5.1. Ordered conditional rewriting.

DEFINITION 4.22. *Let N be a set of Horn clauses. The* ordered rewrite relation $\rightarrow_{N\succ}$ generated by a set N of Horn clauses *is defined as:*

$$
\begin{aligned}
s \rightarrow_{N\succ} t \textit{ if there exists } & \Delta \Rightarrow v \approx w \in N \\
& \textit{a position } p \textit{ in } s, \textit{ and} \\
& \textit{a substitution } \sigma, \\
& \textit{such that } \Delta\sigma \Rightarrow v\sigma \approx w\sigma \textit{ is simplifying,} \\
& v'\sigma \downarrow^*_{N\succ} w'\sigma, \textit{ for all } v' \approx w' \in \Delta, \textit{ and} \\
& s|_p = v\sigma,\ t = s[w\sigma]_p.
\end{aligned}
$$

Note, that by definition $\rightarrow_{N\succ}$ is terminating.

The clause $C = \Gamma \Rightarrow s \approx t$ (or $C = s \approx t, \Gamma \Rightarrow \Phi$) can be replaced by the clause $\Gamma \Rightarrow s' \approx t$ (or $s' \approx t, \Gamma \Rightarrow \Phi$) if $s \rightarrow_{N\succ} s'$ and $C \succ^c D\sigma$, where D is the clause and σ is the substitution used for rewriting s.

5.2. Contextual rewriting. In contextual rewriting we use the equations in the antecedent of a clause as additional rewrite rules when rewriting a term in the succedent by ordered conditional rewriting. [1]

DEFINITION 4.23. *The* constant congruence $\sim_\Gamma$ generated by a set Γ of equations *is the smallest congruence such that $u[s]_p \sim_\Gamma u[t]_p$, for all terms u, positions p in u and equations $s \approx t \in \Gamma$.*

DEFINITION 4.24. *Let N be a set of Horn clauses and Γ be a set of equations. The* contextual rewrite relation $\rightarrow_{N,\Gamma}$ generated by N and Γ *is defined by:*

$$
\begin{aligned}
s \rightarrow_{N,\Gamma} t \textit{ if either } & s \sim_\Gamma t \\
\textit{or there exists } & \Delta \Rightarrow v \approx w \in N \\
& \textit{a position } p \textit{ in } s, \textit{ and} \\
& \textit{a substitution } \sigma, \\
& \textit{such that } \Delta\sigma \Rightarrow v\sigma \approx w\sigma \textit{ is simplifying,} \\
& v'\sigma \downarrow^*_{N,\Gamma} w'\sigma, \textit{ for all } v' \approx w' \in \Delta, \textit{ and} \\
& s|_p = v\sigma,\ t = s[w\sigma]_p.
\end{aligned}
$$

The clause $C = \Gamma \Rightarrow s \approx t$ can be replaced by the clause $\Gamma \Rightarrow s' \approx t$ if $s \rightarrow_{N,\Gamma} s'$ and $C \succ^c D\sigma$, where D is the clause and σ is the substitution used for rewriting s.

The contextual rewrite relation $\rightarrow_{N,\Gamma}$ is implemented in **HCC** as follows:

[1] Contextual rewriting is implemented in **HCC** as described in [**Zhang, 1992**].

(i) We take the Skolemized version $\hat{\Gamma} \Rightarrow \hat{s} \approx \hat{t}$ of C and transfer the antecedent $\hat{\Gamma}$ into a convergent rewrite system $\hat{\Gamma}_0$. The ordering $\succ$ is extended to $\succ_0$ in such a way that the Skolem constants are less than the function symbols. Note, that all equations in $\hat{\Gamma}_0$ are simplifying with respect to $\succ_0$, since $\succ_0$ is total on ground terms. (With the help of $\hat{\Gamma}_0$ we can now decide $\sim_\Gamma$.)

(ii) The equations in $\hat{\Gamma}_0$ are used as additional rewrite rules in ordered conditional rewriting. Let $\hat{s'}$ be such that $\hat{s} \rightarrow_{(N \cup \hat{\Gamma}_0)^{\succ}} \hat{s'}$ and $C \succ^c D\sigma$, where D is the clause and σ is the substitution used for rewriting s.

(iii) Let s' be the result of replacing the Skolem constants of $\hat{s'}$ with the corresponding variables.

Example 2 in Chapter 5 shows how contextual rewriting is used to simplify clauses.

5.3. Context enrichment. Let $C = \Gamma \Rightarrow s \approx t$. In context enrichment we add equations $v \approx w$ to the context for which $(N \cup \Gamma) \models v \approx w$ and $C \succ^c v \approx w$ holds.

Context enrichment is implemented in **HCC** as follows:

(i) Compute $\hat{\Gamma}_0$ as in contextual rewriting.

(ii) Add $v\sigma \approx w\sigma$ to the context if there exists a clause $\Delta \Rightarrow v \approx w \in N$ and a substitution σ such that $D = \Delta\sigma \subseteq \hat{\Gamma}_0$ and $C \succ^c D\sigma$.

(iii) Repeat step (ii) at most `*context-enrich-max*` times.

5.4. Application of non-reductive equations. In ordered conditional rewriting we also allow the application of non-simplifying equations. To avoid loops we limit the number of such applications. The bound is `*ante-rewrite-depth*` and `*suc-rewrite-depth*` for rewriting terms in the antecedent and succedent, respectively.

5.5. Subsumption.

DEFINITION 4.25. *We say that a clause* $C = \Gamma \Rightarrow \Phi$ *subsumes a clause* $D = \Delta \Rightarrow \Psi$ *if there exists a substitution* σ *such that* $\Gamma\sigma \subseteq \Delta$ *and* $\Phi\sigma \subseteq \Psi$*.*

If C subsumes D, the clause D can be deleted.

5.6. Case analysis on the orderings on variables. When we are interested in ground-convergent sets only we may use the additional information that $\succ$ is complete on ground terms. Hence, variables when substituted with ground terms will be comparable. For that we extend the ordering $\succ$ to an ordering $\succ^{var}$ which allows us to compare variables also. In order to prove that an equation $s \approx t$ is composite (and hence can be deleted) we have to show that $s \downarrow^*_{N^{\succ^{var}}} t$, for all possible orderings $\succ^{var}$ on variables occuring in s and t.

For more details about this method see [**Martin and Nipkow, 1990**].

This technique is used in Example 1.2 in Chapter 5 to obtain a ground convergent rewrite system for idempotent commutative semigroups.

6. Convergence

Given a set N_0 of Horn clauses. Suppose a fair derivation $N_0, N_1, \ldots$ (with respect to $\mathcal{OC}_S$) terminates after finitely many steps without having produced the empty clause, so that $N_k = N_\infty$ is a finite set of clauses, for some $k \in \mathbb{N}$. We will show that the set N_k is ground convergent.

DEFINITION 4.26. *By $N^\succ$ we denote the set of all simplifying instances of clauses of N, i.e. $N^\succ = \{\Gamma\sigma \Rightarrow s\sigma \approx t\sigma \mid \Gamma \Rightarrow s \approx t \in N$ and $\Gamma\sigma \Rightarrow s\sigma \approx t\sigma$ is simplifying$\}$.*

Viewing the set $N^\succ$ as a set of conditional rules we can define a rewrite relation $\rightarrow_{N^\succ}$. As one can easily see, this definition coincides with the definition of ordered conditional rewriting given in the previous Section.

DEFINITION 4.27. *Let N be a set of Horn clauses. The rewrite relation $\rightarrow_{N^\succ}$ is said called* ground convergent *if for all ground terms s and t: $s \leftrightarrow^*_{N^\succ} t$ implies $s \downarrow^*_{N^\succ} t$.*

First we consider the case where N_0 consists only of equations.

6.1. Ordered Completion for Equations. Let E be a set of equations. We have shown that the interpretation $\mathfrak{I}_R$ of a fair derivation with respect to $\mathcal{OC}_S$ is a model of E, where R is a ground convergent rewrite system consisting of certain simplifying ground instances of E. Thus, for all ground terms s and t: $s \leftrightarrow^*_{E^\succ} t$ if and only if $s \downarrow^*_R t$ if and only if $s \downarrow^*_{E^\succ} t$. So we have proven the following theorem.

THEOREM 4.28. *Let $\succ$ be a complete reduction ordering and E be a set of equations. If E_∞ is the limit of a fair derivation with respect to $\mathcal{OC}_S$ starting from E, then $\rightarrow_{E^\succ_\infty}$ is ground convergent.*

That means, if for a set E of equations all "Superposition, right" inferences (no other inference rule of $\mathcal{OC}_S$ is applicable) are composite, then E is ground convergent. Because of this similarity to Standard Completion, conclusions of the "Superposition, right"–inference are also called *ordered critical pairs.*

If all equations in E_∞ are simplifying, the rewrite relation $\rightarrow_{E^\succ_\infty}$ is not just ground convergent but convergent on all terms. Let $\mathcal{F}$ and $\mathcal{V}$ be the given sets of function symbols and variables, respectively. Let $\mathcal{C}$ be a set of new constant symbols such that there exists a bijection $\iota : \mathcal{V} \rightarrow \mathcal{C}$ and let $\kappa : \mathcal{C} \rightarrow \mathcal{F}$ be the function that maps each constant in $\mathcal{C}$ to the same minimal (with respect to $\succ$) constant in $\mathcal{F}$. We extend the reduction ordering $\succ$ on $\mathcal{T}(\mathcal{F}, \mathcal{V})$ to a reduction ordering $\succ_\kappa$ on $\mathcal{T}(\mathcal{F} \cup \mathcal{C}, \mathcal{V})$ in the following way [**Bachmair, 1991**]:

$$s \succ_\kappa t \text{ if and only if either } \kappa(s) \succ \kappa(t)$$
$$\text{or } \kappa(s) = \kappa(t) \text{ and } s \succ_{\mathcal{F},\mathcal{C}} t,$$

where $\succ_{\mathcal{F},\mathcal{C}}$ denotes the lexicographic recursive path ordering based on a well-founded, total precedence on $\mathcal{F} \cup \mathcal{C}$.

LEMMA 4.29. *Let $\succ$ be a complete reduction ordering and let E be a set of equations such that $v \succ w$, for all $v \approx w \in E$. Then $\iota(s) \rightarrow_{E^{\succ_\kappa}} \iota(t)$ implies $s \rightarrow_{E^\succ} t$, for all $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$.*

PROOF. **[Bachmair et al., 1993]**
Suppose that s and t are of the form $s[v\sigma]_p$ and $s[w\sigma]_p$, for some equation $v \approx w \in E$ and position p in s, and $\iota(s) \succ_\kappa \iota(t)$. Since either $v \succ w$ or $w \succ v$ we know that $\kappa(\iota(s)) \neq \kappa(\iota(t))$. Hence, we know $\kappa(\iota(s)) \succ \kappa(\iota(t))$ which implies that $v \succ w$ and $s \succ t$. □

THEOREM 4.30. *Let $\succ$ be a complete reduction ordering and E be a set of equations. If E_∞ is the limit of a fair derivation with respect to $\mathcal{OC}_S$ starting from E and $v \succ w$, for all $v \approx w \in E_\infty$, then $\rightarrow_{E_\infty^\succ}$ is convergent.*

PROOF. **[Bachmair et al., 1993]**
Every fair derivation with respect to the ordering $\succ$ is also a fair derivation with respect to the ordering $\succ_\kappa$. Therefore E_∞ is also ground convergent on $\mathcal{T}(\mathcal{F} \cup \mathcal{C}, \mathcal{V})$.

Let s and t be two terms in $\mathcal{T}(\mathcal{F}, \mathcal{V})$, such that $s \leftrightarrow^*_{E_\infty^\succ} t$. Then $\iota(s) \leftrightarrow^*_{E_\infty^{\succ_\kappa}} \iota(t)$ and, since E_∞ is ground convergent on $\mathcal{T}(\mathcal{F} \cup \mathcal{C}, \mathcal{V})$, $\iota(s) \downarrow^*_{E_\infty^{\succ_\kappa}} \iota(t)$. Using Lemma 4.29 we get $s \downarrow^*_{E_\infty^\succ} t$. □

The next theorem shows that Ordered Completion can be used as an unfailing extension of Standard Completion.

THEOREM 4.31. *Let $\succ$ be a complete reduction ordering, E a set of equations and R a reduced convergent rewrite system (with respect to $\succ$) for E. Further, let E_∞ be the limit of a fair derivation with respect to $\mathcal{OC}_S$ starting from E where all equations are reduced (with respect to $\rightarrow_{E_\infty^\succ}$). Then all equations in E_∞ are simplifying and E_∞ and R are literally similar (identical up to renaming of variables).*

PROOF. A proof can be found in **[Bachmair, 1991]** □

That is, in Ordered Completion it cannot happen that one derivation succeeds in computing a convergent set of rules while another derivation does not. Moreover, if there exists a convergent rewrite system (for a given reduction ordering), we will find it by Ordered Completion.

6.2. Ordered Completion for Horn clauses. In the case of Horn clauses we can state the following Theorem.

THEOREM 4.32. *Let $N_u \cup N_s$ be a saturated set of Horn clauses, where no clause in N_u contains a selected equation and each clause in N_s contains at least one selected equation. Then the initial algebras of $N_u^\succ$, N_u, and $N_u \cup N_s$ are the same and $N_u^\succ$ is ground convergent.*

PROOF. **[Bachmair and Ganzinger, 1994]** □

If $N_u \cup N_s$ is also complete with respect to an extension of the reduction ordering by infinitely many new constants then $N_u^\succ$ is not only ground convergent but convergent on all terms (see proof of Theorem 4.30).

Let N be a set of Horn clauses and S be a unitary selection function such that a clause C contains a selected equation if and only if C is non-reductive. Let $N_u \cup N_s$ be the limit of a fair derivation with respect to $\mathcal{OC}_S$ starting from N, where N_u and N_s are defined as above. Further assume that the completion procedure uses only the simplification and deletion techniques described in Section 5 except “Case analysis on the orderings between variables”. Then the completion procedure is basically an unfailing extension of the completion procedure described by Ganzinger in [**Ganzinger, 1991**] and the rewrite relation $\rightarrow_{N_u^{\succ}}$ is convergent.

CHAPTER 5

Examples

1. Two Equational Examples

In this section we will illustrate how **HCC** can be used to construct a (ground) convergent rewrite systems from given sets of equations.

1.1. Let E be the set of equations from page 34

$$\begin{aligned} 1 \cdot ((-x) + x) &\approx 0 \\ 1 \cdot (x + (-x)) &\approx x + (-x) \\ (-x) + x &\approx y + (-y). \end{aligned}$$

The input file `bac91` for the above system is:[1]

```
;; **************************************
;; * Example from Bachmair 1991         *
;; **************************************

;; options (must end with ".")
(setf *simpl-cond* NIL)
(setf *use-context* NIL)
(setf *trace-level* 10)
.

;; precedence ">"
* i + 1 0

;; axioms
(1 * (i(x0) + x0)) = 0
(1 * (x0 + i(x0))) = (x0 + i(x0))
(i(x0) + x0) = (x1 + i(x1))
```

Running the Ordered Completion procedure yields:

```
*******************************************
*  This is HCC (common-lisp version 1.4c)  *
*******************************************

hcc 1: i
Input filename: bac91
hcc 2: d

Precedence:
  [ * i + 1 0 ]

Raw Clauses:
   1. (1*(i(x0)+x0))=0   [user input]
   2. (1*(x0+i(x0)))=(x0+i(x0))   [user input]
```

[1] See Appendix A for a detailed description of the options available in **HCC** .

```
   3. (i(x0)+x0)=(x1+i(x1))   [user input]

Selected Clauses:

Non-selected Clauses:

hcc 3: c
add clause:
   1. (1*(i(x0)+x0))=0   [user input]
add clause:
   2. (1*(x0+i(x0)))=(x0+i(x0))   [user input]
add clause:
   3. (i(x0)+x0)=(x1+i(x1))   [user input]
inferences with:
   1. (1*(i(x0)+x0))=0   [user input]
inferences with:
   2. (1*(x0+i(x0)))=(x0+i(x0))   [user input]
inferences with:
   3. (i(x0)+x0)=(x1+i(x1))   [user input]
add clause:
   4. (x0+i(x0))=0   [3 into 1 (r)]
clause has become redundant:
   2. (1*(x0+i(x0)))=(x0+i(x0))   [user input]
add clause:
   5. (1*0)=0   [e-normalized 2]
clause has become redundant:
   3. (i(x0)+x0)=(x1+i(x1))   [user input]
add clause:
   6. (i(x0)+x0)=0   [e-normalized 3]
inferences with:
   5. (1*0)=0   [e-normalized 2]
inferences with:
   6. (i(x0)+x0)=0   [e-normalized 3]
inferences with:
   4. (x0+i(x0))=0   [3 into 1 (r)]

  Set of clauses is COMPLETE.

hcc 4: d

Precedence:
  [ * i + 1 0 ]

Raw Clauses:

Selected Clauses:

Non-selected Clauses:
   5. (1*0)=0   [e-normalized 2]
   6. (i(x0)+x0)=0   [e-normalized 3]
   4. (x0+i(x0))=0   [3 into 1 (r)]

hcc 5: quit
HCC - end
```

The key step in the above derivation is the deduction of the equation (ordered critical pair)

$$1 \cdot (y + (-y)) \approx 0$$

by "Superposition, right" between the non-simplifying equation 3 and equation 1. This equation is then simplified and renamed into

$$(x + (-x)) \approx 0.$$

Since all equations in the limit system are simplifying we know by Theorem 4.30 that they form a convergent rewrite system for E.

Therefore we have the convergent system

$$1 \cdot 0 \to 0$$
$$(-x) + x \to 0$$
$$x + (-x) \to 0.$$

1.2. Idempotent Commutative Semigroups. An idempotent commutative semigroup is a semigroup where besides

$$(x \cdot y) \cdot z \approx x \cdot (y \cdot z)$$

also

$$x \cdot y \approx y \cdot x$$
$$x \cdot x \approx x$$

hold.

The input file `ics`:

```
;; ******************************************
;; * idempotent commutative semigroup       *
;; ******************************************

(setf *simpl-cond* NIL)
(setf *use-context* NIL)
(setf *del-case* T)
(setf *trace-level* 10)
.

;; precedence
*

;; axioms
(x0 * x1) = (x1 * x0)
((x0 * x1) * x2) = (x0 * (x1 * x2))
(x0 * x0) = x0
```

Applying the completion procedure we get:

```
*******************************************
*  This is HCC (common-lisp version 1.4c)  *
*******************************************

hcc 1: i
Input filename: ics
```

```
hcc 2: d

Precedence:
  [ * ]

Raw Clauses:
   3. (x0*x0)=x0   [user input]
   1. (x0*x1)=(x1*x0)   [user input]
   2. ((x0*x1)*x2)=(x0*(x1*x2))   [user input]

Selected Clauses:

Non-selected Clauses:

hcc 3: c
add clause:
   3. (x0*x0)=x0   [user input]
add clause:
   1. (x0*x1)=(x1*x0)   [user input]
add clause:
   2. ((x0*x1)*x2)=(x0*(x1*x2))   [user input]
inferences with:
   3. (x0*x0)=x0   [user input]
inferences with:
   1. (x0*x1)=(x1*x0)   [user input]
inferences with:
   2. ((x0*x1)*x2)=(x0*(x1*x2))   [user input]
add clause:
   4. (x0*(x1*(x0*x1)))=(x0*x1)   [2 into 3 (r)]
add clause:
   5. (x0*(x0*x1))=(x0*x1)   [3 into 2 (r)]
add clause:
   6. (x0*(x1*x2))=(x1*(x2*x0))   [2 into 1 (r)]
add clause:
   7. (x0*(x1*x2))=(x1*(x0*x2))   [1 into 2 (r)]
inferences with:
   5. (x0*(x0*x1))=(x0*x1)   [3 into 2 (r)]
inferences with:
   7. (x0*(x1*x2))=(x1*(x0*x2))   [1 into 2 (r)]

  Set of clauses is COMPLETE.

hcc 4: d

Precedence:
  [ * ]

Raw Clauses:

Selected Clauses:

Non-selected Clauses:
   3. (x0*x0)=x0   [user input]
   5. (x0*(x0*x1))=(x0*x1)   [3 into 2 (r)]
   1. (x0*x1)=(x1*x0)   [user input]
   2. ((x0*x1)*x2)=(x0*(x1*x2))   [user input]
   7. (x0*(x1*x2))=(x1*(x0*x2))   [1 into 2 (r)]
```

```
hcc 5: quit
HCC - end
```

The procedure stops with the ground convergent system (simplifying equations are written as rules):

$$x \cdot x \to x$$
$$x \cdot (x \cdot y) \to x \cdot y$$
$$x \cdot y \approx y \cdot x$$
$$(x \cdot y) \cdot z \to x \cdot (y \cdot z)$$
$$x \cdot (y \cdot z) \approx y \cdot (x \cdot z).$$

Note, that the deletion technique "Case analysis on the orderings between variables" is essential in this example to make the completion derivation terminate. E.g. superposing equation 1 on equation 5 yields

$$(x \cdot y) \cdot x \approx x \cdot y$$

which can be reduced to

$$(*) \qquad x \cdot (y \cdot x) \approx x \cdot y.$$

Without any additional information on the ordering of the variables x and y this equation is irreducible and hence would be added as a superposition-deduction. But assuming $x \succ y$ allows us to simplify the equation $(*)$ into (by equations 7, 3 and 1)

$$y \cdot x \approx y \cdot x$$

which is trivial. The equation $(*)$ reduces into a trivial one, for all other possible orderings on the variables x and y and therefore can be deleted.

2. The Natural Numbers with $\leq$

This example is clausal specification of the $\leq$-predicate for the natural numbers with the totality and transitivity axiom.

$$\begin{aligned} & le(0,x) \approx true \\ & le(s(x),0) \approx false \\ & le(x,x) \approx true \\ & le(s(x),s(y)) \approx le(x,y) \\ le(x,y) \approx true \Rightarrow\ & le(x,s(y)) \approx true \\ le(x,y) \approx false \Rightarrow\ & le(y,x) \approx true \\ le(x,y) \approx true, le(y,z) \approx true \Rightarrow\ & le(x,z) \approx true \end{aligned}$$

The clause

$$true \approx false \Rightarrow$$

has to be added (as an inductive theorem) in order to make the completion procedure terminate.

The input file `nat-le`:

```
;; *********************************************
;; * natural number with less-equal predicate   *
;; *********************************************

;; options (must end with ".")
(setf *simpl-cond* T)
(setf *use-context* T)
(setf *enrich-context* T)
(setf *del-case* NIL)
(setf *trace-level* 15)
.

;; precedence ">"
le s 0 true false

;; axioms
true = false -->

le(0, x0) = true
le(s(x0), 0) = false
le(x0, x0) = true
le(s(x0), s(x1)) = le(x0, x1)
le(x0, x1) = true --> le(x0, s(x1)) = true

le(x0, x1) = false --> le(x1, x0) = true

le(x0, x1) = true & le(x1, x2) = true --> le(x0, x2) = true
```

HCC produces to following output while completing the system:

```
******************************************
*  This is HCC (common-lisp version 1.4c)  *
******************************************

hcc 1: i
Input filename: nat-le
hcc 2: d

Precedence:
  [ le s 0 true false ]

Raw Clauses:
   1. true=false -->    [user input]
   2. le(0,x0)=true   [user input]
   4. le(x0,x0)=true   [user input]
   3. le(s(x0),0)=false   [user input]
   6. le(x0,x1)=true --> le(x0,s(x1))=true   [user input]
   5. le(s(x0),s(x1))=le(x0,x1)   [user input]
   8. le(x0,x2)=true & le(x2,x1)=true --> le(x0,x1)=true   [user input]
   7. le(x1,x0)=false --> le(x0,x1)=true   [user input]

Selected Clauses:

Non-selected Clauses:

hcc 3: c
process clause (i):
   1. true=false -->    [user input]
process clause (o):
   1. true=false -->    [user input]
add clause:
   1. true=false -->    [user input]
process clause (i):
   2. le(0,x0)=true   [user input]
process clause (o):
   2. le(0,x0)=true   [user input]
add clause:
   2. le(0,x0)=true   [user input]
process clause (i):
   4. le(x0,x0)=true   [user input]
process clause (o):
   4. le(x0,x0)=true   [user input]
add clause:
   4. le(x0,x0)=true   [user input]
process clause (i):
   3. le(s(x0),0)=false   [user input]
process clause (o):
   3. le(s(x0),0)=false   [user input]
add clause:
   3. le(s(x0),0)=false   [user input]
process clause (i):
   6. le(x0,x1)=true --> le(x0,s(x1))=true   [user input]
process clause (o):
   6. le(x0,x1)=true --> le(x0,s(x1))=true   [user input]
add clause:
   6. le(x0,x1)=true --> le(x0,s(x1))=true   [user input]
```

```
process clause (i):
   5. le(s(x0),s(x1))=le(x0,x1)   [user input]
process clause (o):
   5. le(s(x0),s(x1))=le(x0,x1)   [user input]
add clause:
   5. le(s(x0),s(x1))=le(x0,x1)   [user input]
process clause (i):
   8. le(x0,x2)=true & le(x2,x1)=true --> le(x0,x1)=true   [user input]
process clause (o):
   8. le(x0,x2)=true & le(x2,x1)=true --> le(x0,x1)=true   [user input]
add clause:
   8. le(x0,x2)=true & le(x2,x1)=true --> le(x0,x1)=true   [user input]
process clause (i):
   7. le(x1,x0)=false --> le(x0,x1)=true   [user input]
process clause (o):
   7. le(x1,x0)=false --> le(x0,x1)=true   [user input]
add clause:
   7. le(x1,x0)=false --> le(x0,x1)=true   [user input]
add marked clause:
   1. true=false -->    [user input]
inferences with:
   1. true=false -->    [user input]
add marked clause:
   2. le(0,x0)=true   [user input]
inferences with:
   2. le(0,x0)=true   [user input]
add marked clause:
   3. le(s(x0),0)=false   [user input]
inferences with:
   3. le(s(x0),0)=false   [user input]
add marked clause:
   4. le(x0,x0)=true   [user input]
inferences with:
   4. le(x0,x0)=true   [user input]
add marked clause:
   6. le(x0,x1)=true --> le(x0,s(x1))=true   [user input]
inferences with:
   6. le(x0,x1)=true --> le(x0,s(x1))=true   [user input]
add marked clause:
   5. le(s(x0),s(x1))=le(x0,x1)   [user input]
inferences with:
   5. le(s(x0),s(x1))=le(x0,x1)   [user input]
process clause (i):
   9. le(s(x0),x1)=true --> le(x0,x1)=true   [5 into 6 (r)]
process clause (o):
   9. le(s(x0),x1)=true --> le(x0,x1)=true   [5 into 6 (r)]
add clause:
   9. le(s(x0),x1)=true --> le(x0,x1)=true   [5 into 6 (r)]
add marked clause:
   7. le(x1,x0)=false --> le(x0,x1)=true   [user input]
inferences with:
   7. le(x1,x0)=false --> le(x0,x1)=true   [user input]
process clause (i):
  10. true=false --> le(x0,0)=true   [2 into 7 (l1)]
process clause (o):
  11. true=false --> le(x0,0)=false   [subsumed by 1]
process clause (i):
```

```
  12. true=false --> le(x0,x0)=true   [4 into 7 (l1)]
process clause (o):
  12. true=false --> le(x0,x0)=true   [trivial by e-rewriting]
process clause (i):
  13. le(0,s(x0))=true   [3 into 7 *(l1)]
process clause (o):
  13. le(0,s(x0))=true   [trivial by e-rewriting]
process clause (i):
  14. le(x1,x0)=true & true=false --> le(s(x0),x1)=true   [6 into 7 (l1)]
process clause (o):
  15. le(x1,x0)=false & true=false --> le(s(x0),x1)=false   [subsumed by 1]
process clause (i):
  16. le(x1,x0)=false --> le(s(x0),s(x1))=true   [5 into 7 (l1)]
process clause (o):
  17. le(x1,x0)=false --> le(x0,x1)=true   [subsumed by 7]
add marked clause:
   8. le(x0,x2)=true & le(x2,x1)=true --> le(x0,x1)=true   [user input]
inferences with:
   8. le(x0,x2)=true & le(x2,x1)=true --> le(x0,x1)=true   [user input]
process clause (i):
  18. le(x1,x0)=true --> le(0,x0)=true   [2 into 8 *(l1)]
process clause (o):
  18. le(x1,x0)=true --> le(0,x0)=true   [trivial by e-rewriting]
process clause (i):
  19. le(x0,x1)=true --> le(x0,x1)=true   [4 into 8 *(l1)]
process clause (o):
  19. le(x0,x1)=true --> le(x0,x1)=true   [trivial by e-rewriting]
process clause (i):
  20. le(0,x1)=true & true=false --> le(s(x0),x1)=true   [3 into 8 (l1)]
process clause (o):
  21. true=false --> le(s(x0),x1)=false   [subsumed by 1]
process clause (i):
  22. le(s(x2),x1)=true & le(x0,x2)=true --> le(x0,x1)=true   [6 into 8 *(l1)]
process clause (o):
  22. le(s(x2),x1)=true & le(x0,x2)=true --> le(x0,x1)=true   [6 into 8 *(l1)]
add clause:
  22. le(s(x2),x1)=true & le(x0,x2)=true --> le(x0,x1)=true   [6 into 8 *(l1)]
process clause (i):
  23. le(s(x2),x1)=true & le(x0,x2)=true --> le(s(x0),x1)=true   [5 into 8 (l1)]
process clause (o):
  23. le(s(x2),x1)=true & le(x0,x2)=true --> le(s(x0),x1)=true   [5 into 8 (l1)]
add clause:
  23. le(s(x2),x1)=true & le(x0,x2)=true --> le(s(x0),x1)=true   [5 into 8 (l1)]
add marked clause:
  22. le(s(x2),x1)=true & le(x0,x2)=true --> le(x0,x1)=true   [6 into 8 *(l1)]
inferences with:
  22. le(s(x2),x1)=true & le(x0,x2)=true --> le(x0,x1)=true   [6 into 8 *(l1)]
process clause (i):
  24. le(x0,x1)=true --> le(x0,s(x1))=true   [4 into 22 *(l1)]
process clause (o):
  24. le(x0,x1)=true --> le(x0,s(x1))=true   [trivial by r-rewriting]
process clause (i):
  25. le(x0,x1)=true & true=false --> le(x0,0)=true   [3 into 22 (l1)]
process clause (o):
  26. le(x0,x1)=false & true=false --> le(x0,0)=false   [subsumed by 1]
process clause (i):
  27. le(s(x2),x1)=true & le(x0,x2)=true --> le(x0,s(x1))=true   [6 into 22 *(l1)]
```

```
process clause (o):
  27. le(s(x2),x1)=true & le(x0,x2)=true --> le(x0,s(x1))=true   [trivial by r-rewriting]
process clause (i):
  28. le(x2,x1)=true & le(x0,x2)=true --> le(x0,s(x1))=true   [5 into 22 (l1)]
process clause (o):
  28. le(x2,x1)=true & le(x0,x2)=true --> le(x0,s(x1))=true   [trivial by r-rewriting]
add marked clause:
   9. le(s(x0),x1)=true --> le(x0,x1)=true   [5 into 6 (r)]
inferences with:
   9. le(s(x0),x1)=true --> le(x0,x1)=true   [5 into 6 (r)]
process clause (i):
  29. le(x0,s(x0))=true   [4 into 9 *(l1)]
process clause (o):
  29. le(x0,s(x0))=true   [trivial by r-rewriting]
process clause (i):
  30. true=false --> le(x0,0)=true   [3 into 9 (l1)]
process clause (o):
  31. true=false --> le(x0,0)=false   [subsumed by 1]
process clause (i):
  32. le(s(x0),x1)=true --> le(x0,s(x1))=true   [6 into 9 *(l1)]
process clause (o):
  32. le(s(x0),x1)=true --> le(x0,s(x1))=true   [trivial by r-rewriting]
process clause (i):
  33. le(x0,x1)=true --> le(x0,s(x1))=true   [5 into 9 (l1)]
process clause (o):
  33. le(x0,x1)=true --> le(x0,s(x1))=true   [trivial by r-rewriting]
add marked clause:
  23. le(s(x2),x1)=true & le(x0,x2)=true --> le(s(x0),x1)=true   [5 into 8 (l1)]
inferences with:
  23. le(s(x2),x1)=true & le(x0,x2)=true --> le(s(x0),x1)=true   [5 into 8 (l1)]
process clause (i):
  34. le(x0,x1)=true --> le(s(x0),s(x1))=true   [4 into 23 *(l1)]
process clause (o):
  34. le(x0,x1)=true --> le(s(x0),s(x1))=true   [trivial by e-rewriting]
process clause (i):
  35. le(x0,x1)=true & true=false --> le(s(x0),0)=true   [3 into 23 (l1)]
process clause (o):
  35. le(x0,x1)=true & true=false --> le(s(x0),0)=true   [trivial by e-rewriting]
process clause (i):
  36. le(s(x2),x1)=true & le(x0,x2)=true --> le(s(x0),s(x1))=true   [6 into 23 *(l1)]
process clause (o):
  37. le(s(x2),x1)=true & le(x0,x2)=true --> le(x0,x1)=true   [subsumed by 22]
process clause (i):
  38. le(x2,x1)=true & le(x0,x2)=true --> le(s(x0),s(x1))=true   [5 into 23 (l1)]
process clause (o):
  39. le(x2,x1)=true & le(x0,x2)=true --> le(x0,x1)=true   [subsumed by 8]

  Set of clauses is COMPLETE.

hcc 4: d

Precedence:
  [ le s 0 true false ]

Raw Clauses:

Selected Clauses:
```

```
   1. true=false -->    [user input]
   7. le(x1,x0)=false --> le(x0,x1)=true   [user input]
   8. le(x0,x2)=true & le(x2,x1)=true --> le(x0,x1)=true   [user input]
  22. le(s(x2),x1)=true & le(x0,x2)=true --> le(x0,x1)=true   [6 into 8 *(l1)]
   9. le(s(x0),x1)=true --> le(x0,x1)=true   [5 into 6 (r)]
  23. le(s(x2),x1)=true & le(x0,x2)=true --> le(s(x0),x1)=true   [5 into 8 (l1)]

Non-selected Clauses:
   2. le(0,x0)=true   [user input]
   3. le(s(x0),0)=false   [user input]
   4. le(x0,x0)=true   [user input]
   6. le(x0,x1)=true --> le(x0,s(x1))=true   [user input]
   5. le(s(x0),s(x1))=le(x0,x1)   [user input]

hcc 5: quit
HCC - end
```

According to Theorem 4.32 the set of clauses

$$le(0,x) \approx true$$
$$le(s(x),0) \approx false$$
$$le(x,x) \approx true$$
$$le(s(x),s(y)) \approx le(x,y)$$
$$le(x,y) \approx true \Rightarrow le(x,s(y)) \approx true$$

is ground convergent.

In the following we will show how the simplification and deletion techniques of Section 5 in Chapter 4 are used to simplify clauses or to detect them as redundant.

To detect clause 19

```
  19. le(x0,x1)=true --> le(x0,x1)=true   [4 into 8 *(l1)]
```

as redundant contextual rewriting has to be used. Since we can rewrite $le(x,y)$ into $true$ with respect to $\rightarrow_{N,\Gamma}$, where $\Gamma = \{le(x,y) \approx true\}$, the succedent of the clause becomes trivial and therefore the clause can be deleted.

The clause

```
  27. le(s(x2),x1)=true & le(x0,x2)=true --> le(x0,s(x1))=true   [6 into 22 *(l1)]
```

can also be deleted by contextual recursive rewriting together with context enrichment. The context is set to $\Gamma = \{le(x,y) \approx true, le(s(z),y) \approx true, le(x,z) \approx true\}$, where the equation $le(x,y) \approx true$ is added during context enrichment using clause 22.

```
  22. le(s(x2),x1)=true & le(x0,x2)=true --> le(x0,x1)=true   [6 into 8 *(l1)]
```

The term $le(x,s(y))$ of the succedent of clause 21 can be rewritten into $true$ using clause 6

```
   6. le(x0,x1)=true --> le(x0,s(x1))=true   [user input]
```

provided that $le(x,y)$ can be rewritten into $true$, which is the case since $le(x,y) \approx true \in \Gamma$.

3. The Maximum Function

This example is taken from [**Ganzinger, 1991**] and defines a maximum function based on a transitive quasi-ordering $\leq$.

$$
\begin{aligned}
& le(x,x) \approx true \\
le(x,y) \approx true, le(y,z) \approx true \Rightarrow\ & le(x,z) \approx true \\
le(x,y) \approx true \Rightarrow\ & max(x,y) = y \\
le(x,y) \approx false \Rightarrow\ & max(x,y) = x \\
& le(x, max(x,y)) \approx true \\
& le(y, max(x,y)) \approx true
\end{aligned}
$$

Again, the clause

$$true \approx false \Rightarrow$$

has to be added to make the completion procedure terminate.

The input file `maximum`:

```
;; *************************************************
;; * maximum function based on transitive ordering *
;; *************************************************

;; options (must end with ".")
(setf *simpl-cond* T)
(setf *use-context* T)
(setf *enrich-context* T)
(setf *del-case* NIL)
(setf *trace-level* 10)
.

;; precedence ">"
max le true false


;; axioms
true = false -->

le(x0,x0) = true
le(x0,x1) = true & le(x1,x2) = true --> le(x0,x2) = true

le(x0,x1) = true --> max(x0,x1) = x1
le(x0,x1) = false --> max(x0,x1) = x0
le(x0,max(x0,x1)) = true
le(x1,max(x0,x1)) = true
```

HCC produces the following output:

```
*******************************************
*  This is HCC (common-lisp version 1.4c)  *
*******************************************

hcc 1: i
Input filename: maximum
hcc 2: d
```

```
Precedence:
  [ max le true false ]

Raw Clauses:
   1. true=false -->    [user input]
   5. le(x0,x1)=false --> max(x0,x1)=x0   [user input]
   4. le(x0,x1)=true --> max(x0,x1)=x1   [user input]
   2. le(x0,x0)=true   [user input]
   7. le(x0,max(x1,x0))=true   [user input]
   6. le(x0,max(x0,x1))=true   [user input]
   3. le(x0,x2)=true & le(x2,x1)=true --> le(x0,x1)=true   [user input]

Selected Clauses:

Non-selected Clauses:

hcc 3: c
add clause:
   1. true=false -->    [user input]
add clause:
   5. le(x0,x1)=false --> max(x0,x1)=x0   [user input]
add clause:
   4. le(x0,x1)=true --> max(x0,x1)=x1   [user input]
add clause:
   2. le(x0,x0)=true   [user input]
add clause:
   7. le(x0,max(x1,x0))=true   [user input]
add clause:
   6. le(x0,max(x0,x1))=true   [user input]
add clause:
   3. le(x0,x2)=true & le(x2,x1)=true --> le(x0,x1)=true   [user input]
inferences with:
   1. true=false -->    [user input]
inferences with:
   2. le(x0,x0)=true   [user input]
inferences with:
   4. le(x0,x1)=true --> max(x0,x1)=x1   [user input]
inferences with:
   5. le(x0,x1)=false --> max(x0,x1)=x0   [user input]
inferences with:
   6. le(x0,max(x0,x1))=true   [user input]
inferences with:
   7. le(x0,max(x1,x0))=true   [user input]
add clause:
  11. le(x1,x0)=false --> le(x0,x1)=true   [5 into 7 (r)]
inferences with:
  11. le(x1,x0)=false --> le(x0,x1)=true   [5 into 7 (r)]
inferences with:
   3. le(x0,x2)=true & le(x2,x1)=true --> le(x0,x1)=true   [user input]
add clause:
  18. le(max(x2,x0),x1)=true --> le(x0,x1)=true   [7 into 3 *(l1)]
add clause:
  19. le(max(x0,x2),x1)=true --> le(x0,x1)=true   [6 into 3 *(l1)]
inferences with:
  19. le(max(x0,x2),x1)=true --> le(x0,x1)=true   [6 into 3 *(l1)]
add clause:
  23. le(x0,max(x1,max(x0,x2)))=true   [7 into 19 *(l1)]
```

```
add clause:
  24. le(x0,max(max(x0,x1),x2))=true   [6 into 19 *(l1)]
inferences with:
  24. le(x0,max(max(x0,x1),x2))=true   [6 into 19 *(l1)]
add clause:
  26. le(x0,x1)=true --> le(x0,max(x1,x2))=true   [4 into 24 (r)]
inferences with:
  26. le(x0,x1)=true --> le(x0,max(x1,x2))=true   [4 into 24 (r)]
add clause:
  29. le(max(x2,x3),x1)=true & le(x0,x2)=true --> le(x0,x1)=true   [26 into 3 *(l2)]
inferences with:
  23. le(x0,max(x1,max(x0,x2)))=true   [7 into 19 *(l1)]
add clause:
  33. le(x1,max(x0,x2))=false --> le(x0,x1)=true   [5 into 23 (r)]
add clause:
  35. le(x0,x2)=true --> le(x0,max(x1,x2))=true   [4 into 23 (r)]
inferences with:
  35. le(x0,x2)=true --> le(x0,max(x1,x2))=true   [4 into 23 (r)]
add clause:
  36. le(x1,x2)=false & le(x0,x2)=true --> le(x0,x1)=true   [5 into 35 (r)]
add clause:
  38. le(max(x2,x3),x1)=true & le(x0,x3)=true --> le(x0,x1)=true   [35 into 3 *(l2)]
inferences with:
  36. le(x1,x2)=false & le(x0,x2)=true --> le(x0,x1)=true   [5 into 35 (r)]
inferences with:
  38. le(max(x2,x3),x1)=true & le(x0,x3)=true --> le(x0,x1)=true   [35 into 3 *(l2)]
add clause:
  47. le(x2,x1)=true & le(x0,x3)=true & le(x2,x3)=false --> le(x0,x1)=true   [5 into 38 (l1)]
inferences with:
  47. le(x2,x1)=true & le(x0,x3)=true & le(x2,x3)=false --> le(x0,x1)=true   [5 into 38 (l1)]
inferences with:
  33. le(x1,max(x0,x2))=false --> le(x0,x1)=true   [5 into 23 (r)]
inferences with:
  29. le(max(x2,x3),x1)=true & le(x0,x2)=true --> le(x0,x1)=true   [26 into 3 *(l2)]
add clause:
  64. le(x2,x1)=true & le(x0,x3)=true & le(x3,x2)=true --> le(x0,x1)=true   [4 into 29 (l1)]
inferences with:
  64. le(x2,x1)=true & le(x0,x3)=true & le(x3,x2)=true --> le(x0,x1)=true   [4 into 29 (l1)]
inferences with:
  18. le(max(x2,x0),x1)=true --> le(x0,x1)=true   [7 into 3 *(l1)]
add clause:
  71. le(x2,x1)=true & le(x2,x0)=false --> le(x0,x1)=true   [5 into 18 (l1)]
inferences with:
  71. le(x2,x1)=true & le(x2,x0)=false --> le(x0,x1)=true   [5 into 18 (l1)]

  Set of clauses is COMPLETE.

hcc 4: d

Precedence:
  [ max le true false ]

Raw Clauses:

Selected Clauses:
   1. true=false -->    [user input]
  71. le(x2,x1)=true & le(x2,x0)=false --> le(x0,x1)=true   [5 into 18 (l1)]
```

```
  64. le(x2,x1)=true & le(x0,x3)=true & le(x3,x2)=true --> le(x0,x1)=true   [4 into 29 (l1)]
  47. le(x2,x1)=true & le(x0,x3)=true & le(x2,x3)=false --> le(x0,x1)=true   [5 into 38 (l1)]
  36. le(x1,x2)=false & le(x0,x2)=true --> le(x0,x1)=true   [5 into 35 (r)]
  11. le(x1,x0)=false --> le(x0,x1)=true   [5 into 7 (r)]
   3. le(x0,x2)=true & le(x2,x1)=true --> le(x0,x1)=true   [user input]
  38. le(max(x2,x3),x1)=true & le(x0,x3)=true --> le(x0,x1)=true   [35 into 3 *(l2)]
  33. le(x1,max(x0,x2))=false --> le(x0,x1)=true   [5 into 23 (r)]
  29. le(max(x2,x3),x1)=true & le(x0,x2)=true --> le(x0,x1)=true   [26 into 3 *(l2)]
  19. le(max(x0,x2),x1)=true --> le(x0,x1)=true   [6 into 3 *(l1)]
  18. le(max(x2,x0),x1)=true --> le(x0,x1)=true   [7 into 3 *(l1)]

Non-selected Clauses:
   2. le(x0,x0)=true   [user input]
   4. le(x0,x1)=true --> max(x0,x1)=x1   [user input]
   5. le(x0,x1)=false --> max(x0,x1)=x0   [user input]
  35. le(x0,x2)=true --> le(x0,max(x1,x2))=true   [4 into 23 (r)]
  26. le(x0,x1)=true --> le(x0,max(x1,x2))=true   [4 into 24 (r)]

hcc 5: quit
HCC - end
```

We have the following ground convergent set of clauses:

$$
\begin{aligned}
le(x,x) &\approx true \\
le(x,y) \approx true &\Rightarrow max(x,y) \approx y \\
le(x,y) \approx false &\Rightarrow max(x,y) \approx x \\
le(x,z) \approx true &\Rightarrow le(x,max(y,z)) \approx true \\
le(x,y) \approx true &\Rightarrow le(x,max(y,z)) \approx true.
\end{aligned}
$$

4. Lists

4.1. Given a total-transitive quasi-ordering $\leq$. We define the following functions and predicates for lists.

$first(L)$	Return the first element of the list L.
$rest(L)$	Return the sublist of L starting at the second element.
$append(L1, L2)$	Append two lists $L1$ and $L2$.
$length(L)$	Return the length of the list L.
$member(x, L)$	Check if x is a member of the list L.
$insert(x, L)$	Insert x in the list L such that all elements before x are less than x.
$max(L)$	return the maximal element of the list L (with respect to $\leq$).
$reverse(L)$	Reverse the list L.
$palindrome(L)$	Test if the list L is a palindrome.

The input file `lists`:

```
;; ********************************
;; * lists                        *
;; ********************************

;; options (must end with ".")
.

;; precedence ">"
palindrome reverse reverse2 max insert member append length first rest cons e le s 0 true false

;; axioms
true = false -->

le(x0,x0) = true
le(x0,x1) = false --> le(x1,x0) = true
le(x0, x1) = true & le(x1, x2) = true --> le(x0, x2) = true

first(cons(x0, x10)) = x0
rest(cons(x0, x10)) = x10

append(e, x10) = x10
append(cons(x0, x10), x11) = cons(x0, append(x10, x11))

length(e) = 0
length(cons(x0, x10)) = s(length(x10))

member(x0, e) = false
member(x0, cons(x0, x10)) = true
member(x0, x10) = true --> member(x0, cons(x1, x10)) = true

insert(x0, e) = cons(x0, e)
le(x0, x1) = true --> insert(x0, cons(x1, x10)) = cons(x0, cons(x1, x10))
le(x0, x1) = false --> insert(x0, cons(x1, x10)) = cons(x1, insert(x0, x10))

max(cons(x0, e)) = x0
le(x0, x1) = true --> max(cons(x0, cons(x1, x10))) = max(cons(x1, x10))
le(x0, x1) = false --> max(cons(x0, cons(x1, x10))) = max(cons(x0, x10))

reverse(x10) = reverse2(x10, e)
```

```
reverse2(e, x10) = x10
reverse2(cons(x0, x10), x11) = reverse2(x10, cons(x0, x11))

reverse(x10) = x10 --> palindrome(x10) = true
```

The output of **HCC** looks like:

```
*******************************************
*  This is HCC (common-lisp version 1.4c)  *
*******************************************

hcc 1: i
Input filename: lists
hcc 2: c

  Set of clauses is COMPLETE.

hcc 3: d

Precedence:
  [ palindrome reverse reverse2 max insert member append length first rest cons e le
s 0 true false ]

Raw Clauses:

Selected Clauses:
   1. true=false -->    [user input]
   3. le(x1,x0)=false --> le(x0,x1)=true   [user input]
   4. le(x0,x2)=true & le(x2,x1)=true --> le(x0,x1)=true   [user input]

Non-selected Clauses:
   9. length(e)=0   [user input]
  24. reverse2(x0,e)=x0 --> palindrome(x0)=true   [e-normalized 23]
  11. member(x0,e)=false   [user input]
   2. le(x0,x0)=true   [user input]
   7. append(e,x0)=x0   [user input]
  21. reverse2(e,x0)=x0   [user input]
  17. max(cons(x0,e))=x0   [user input]
  20. reverse(x0)=reverse2(x0,e)   [user input]
   5. first(cons(x0,x1))=x0   [user input]
   6. rest(cons(x0,x1))=x1   [user input]
  14. insert(x0,e)=cons(x0,e)   [user input]
  12. member(x0,cons(x0,x1))=true   [user input]
  13. member(x0,x2)=true --> member(x0,cons(x1,x2))=true   [user input]
  10. length(cons(x0,x1))=s(length(x1))   [user input]
  18. le(x0,x1)=true --> max(cons(x0,cons(x1,x2)))=max(cons(x1,x2))   [user input]
  19. le(x0,x1)=false --> max(cons(x0,cons(x1,x2)))=max(cons(x0,x2))   [user input]
   8. append(cons(x0,x1),x2)=cons(x0,append(x1,x2))   [user input]
  15. le(x0,x1)=true --> insert(x0,cons(x1,x2))=cons(x0,cons(x1,x2))   [user input]
  16. le(x0,x1)=false --> insert(x0,cons(x1,x2))=cons(x1,insert(x0,x2))   [user input]
  22. reverse2(cons(x0,x1),x2)=reverse2(x1,cons(x0,x2))   [user input]

hcc 5: quit
HCC - end
```

4.2. Insertion Sort for Natural Numbers. The input file sort:

```
;; *************************************
;; * insertion sort for natural numbers *
```

```
;; *************************************

;; options (must end with ".")
(setf *simpl-cond* T)
(setf *use-context* T)
(setf *enrich-context* T)
(setf *trace-level* 15)
.

;; precedence ">"
sort insert c e le s 0 true false

;; axioms
true = false -->

; less equal
; ----------
le(0, x0) = true
le(s(x0), 0) = false
le(x0, x0) = true
le(s(x0), s(x1)) = le(x0, x1)

; insertion sort
; --------------
sort(e) = e
sort(c(x0, x10)) = insert(x0, sort(x10))

insert(x0, e) = c(x0, e)
le(x0, x1) = true --> insert(x0, c(x1, x10)) = c(x0, c(x1, x10))
le(x0, x1) = false --> insert(x0, c(x1, x10)) = c(x1, insert(x0, x10))
```

Running **HCC** we get as output:

```
*******************************************
*  This is HCC (common-lisp version 1.4c)  *
*******************************************

hcc 1: i
Input filename: sort
hcc 2: c

  Set of clauses is COMPLETE.

hcc 3: d

Precedence:
  [ sort insert c e le s 0 true false ]

Raw Clauses:

Selected Clauses:
   1. true=false -->    [user input]

Non-selected Clauses:
   6. sort(e)=e   [user input]
   2. le(0,x0)=true   [user input]
   3. le(s(x0),0)=false   [user input]
   4. le(x0,x0)=true   [user input]
```

```
  8. insert(x0,e)=c(x0,e)   [user input]
  5. le(s(x0),s(x1))=le(x0,x1)   [user input]
  7. sort(c(x0,x1))=insert(x0,sort(x1))   [user input]
  9. le(x0,x1)=true --> insert(x0,c(x1,x2))=c(x0,c(x1,x2))   [user input]
 10. le(x0,x1)=false --> insert(x0,c(x1,x2))=c(x1,insert(x0,x2))   [user input]

hcc 4:
```

Now, we can use the normal form function to sort lists of natural numbers. E.g. the normal form of the term $sort([4,2,0,1])$ is $sort([0,1,2,4])$.

```
hcc 4: t
     Input term: sort(c(s(s(s(s(0)))), c(s(s(0)), c(0, c(s(0), e)))))
Normalized term: c(0,c(s(0),c(s(s(0)),c(s(s(s(s(0)))),e))))
hcc 5: quit
HCC - end
```

5. Refutational Theorem Proving

5.1. Example 1. If in a group $(G,+)$ it holds that $x+x+x=0$, for all $x \in G$, then the commutator $[[x,y],y]=0$, for all $x,y \in G$, where $[x,y]=x+y+(-x)+(-y)$.

The input file `ref-example1`:

```
;; ******************************************************
;; * Problem from Overbeek                              *
;; ******************************************************

;; options (must end with ".")
(setf *simpl-cond* NIL)
(setf *use-context* NIL)
(setf *trace-level* 10)
.

;; precedence ">"
h  i  +  0  a  b

;; axioms
(0 + x0) = x0
(i(x0) + x0) = 0
((x0 + x1) + x2) = (x0 + (x1 + x2))
(x0 + (x0 + x0)) = 0
h(x0, x1) = (x0 + (x1 + (i(x0) + i(x1))))

h(h(a,b),b) = 0 -->
```

Running the completion procedure yields:

```
*******************************************
*  This is HCC (common-lisp version 1.4c)  *
*******************************************

hcc 1: i
Input filename: ref-example1
hcc 2: d

Precedence:
  [ h i + 0 a b ]

Raw Clauses:
   1. (0+x0)=x0   [user input]
   6. h(h(a,b),b)=0 -->    [user input]
   5. h(x0,x1)=(x0+(x1+(i(x0)+i(x1))))   [user input]
   2. (i(x0)+x0)=0   [user input]
   4. (x0+(x0+x0))=0   [user input]
   3. ((x0+x1)+x2)=(x0+(x1+x2))   [user input]

Selected Clauses:

Non-selected Clauses:

hcc 3: c
add clause:
   1. (0+x0)=x0   [user input]
add clause:
   6. h(h(a,b),b)=0 -->    [user input]
```

```
add clause:
   5. h(x0,x1)=(x0+(x1+(i(x0)+i(x1))))   [user input]
clause has become redundant:
   6. h(h(a,b),b)=0 -->   [user input]
add clause:
   7. ((a+(b+(i(a)+i(b))))+(b+(i((a+(b+(i(a)+i(b)))))+i(b))))=0 -->    [e-normalized 6]
add clause:
   2. (i(x0)+x0)=0   [user input]
add clause:
   4. (x0+(x0+x0))=0   [user input]
add clause:
   3. ((x0+x1)+x2)=(x0+(x1+x2))   [user input]
clause has become redundant:
   7. ((a+(b+(i(a)+i(b))))+(b+(i((a+(b+(i(a)+i(b)))))+i(b))))=0 -->    [e-normalized 6]
add clause:
   8. (a+(b+(i(a)+(i(b)+(b+(i((a+(b+(i(a)+i(b)))))+i(b)))))))=0 -->    [e-normalized 7]
inferences with:
   1. (0+x0)=x0   [user input]
inferences with:
   5. h(x0,x1)=(x0+(x1+(i(x0)+i(x1))))   [user input]
inferences with:
   2. (i(x0)+x0)=0   [user input]
inferences with:
   3. ((x0+x1)+x2)=(x0+(x1+x2))   [user input]
add clause:
   9. (i(x0)+(x0+x1))=x1   [2 into 3 (r)]
clause has become redundant:
   8. (a+(b+(i(a)+(i(b)+(b+(i((a+(b+(i(a)+i(b)))))+i(b)))))))=0 -->    [e-normalized 7]
add clause:
  10. (a+(b+(i(a)+(i((a+(b+(i(a)+i(b)))))+i(b)))))=0 -->    [e-normalized 8]
inferences with:
   4. (x0+(x0+x0))=0   [user input]
add clause:
  11. (x0+(x1+(x0+(x1+(x0+x1)))))=0   [4 into 3 (r)]
add clause:
  12. (x0+(x0+(x0+x1)))=x1   [4 into 3 (r)]
inferences with:
   9. (i(x0)+(x0+x1))=x1   [2 into 3 (r)]
add clause:
  13. (i(0)+x0)=x0   [1 into 9 (r)]
add clause:
  14. (i(i(x0))+0)=x0   [2 into 9 (r)]
add clause:
  15. (i(x0)+0)=(x0+x0)   [4 into 9 (r)]
clause has become redundant:
  14. (i(i(x0))+0)=x0   [2 into 9 (r)]
add clause:
  16. (i(x0)+i(x0))=x0   [e-normalized 14]
add clause:
  17. (i((x0+x1))+(x0+(x1+x2)))=x2   [3 into 9 (r)]
add clause:
  18. (i(i(x0))+x1)=(x0+x1)   [9 into 9 (r)]
inferences with:
  15. (i(x0)+0)=(x0+x0)   [4 into 9 (r)]
add clause:
  19. (i(x0)+x1)=(x0+(x0+x1))   [15 into 3 (r)]
clause has become redundant:
```

```
  15. (i(x0)+0)=(x0+x0)   [4 into 9 (r)]
add clause:
  20. (x0+(x0+0))=(x0+x0)   [e-normalized 15]
clause has become redundant:
   5. h(x0,x1)=(x0+(x1+(i(x0)+i(x1))))   [user input]
clause has become redundant:
  16. (i(x0)+i(x0))=x0   [e-normalized 14]
add clause:
  22. (x0+(x0+i(x0)))=x0   [e-normalized 16]
clause has become redundant:
  17. (i((x0+x1))+(x0+(x1+x2)))=x2   [3 into 9 (r)]
add clause:
  23. (x0+(x1+(x0+(x1+(x0+(x1+x2))))))=x2   [e-normalized 17]
clause has become redundant:
  10. (a+(b+(i(a)+(i((a+(b+(i(a)+i(b)))))+i(b)))))=0 -->    [e-normalized 8]
add clause:
  24. (a+(b+(b+(a+(a+(b+(b+(a+(b+(a+(a+b))))))))))=0 -->    [e-normalized 10]
inferences with:
  19. (i(x0)+x1)=(x0+(x0+x1))   [15 into 3 (r)]
add clause:
  25. (x0+i(x0))=0   [19 into 4 (r)]
clause has become redundant:
  22. (x0+(x0+i(x0)))=x0   [e-normalized 16]
add clause:
  26. (x0+0)=x0   [e-normalized 22]
inferences with:
  26. (x0+0)=x0   [e-normalized 22]
add clause:
  27. i(x0)=(x0+x0)   [26 into 19 (r)]
inferences with:
  27. i(x0)=(x0+x0)   [26 into 19 (r)]
inferences with:
  12. (x0+(x0+(x0+x1)))=x1   [4 into 3 (r)]
inferences with:
  11. (x0+(x1+(x0+(x1+(x0+x1)))))=0   [4 into 3 (r)]
add clause:
  28. (x0+(x1+(x2+(x0+(x1+(x2+(x0+(x1+x2))))))))=0   [11 into 3 (r)]
add clause:
  29. (x0+(x1+(x0+(x1+x0))))=(x1+x1)   [11 into 12 (r)]
inferences with:
  29. (x0+(x1+(x0+(x1+x0))))=(x1+x1)   [11 into 12 (r)]
add clause:
  30. (x0+(x1+(x2+(x0+(x1+(x2+(x0+x1)))))))=(x2+x2)   [29 into 3 (r)]
add clause:
  31. (x0+(x1+(x0+(x1+(x0+x2)))))=(x1+(x1+x2))   [29 into 3 (r)]
add clause:
  32. (x0+(x1+(x2+(x0+(x1+(x2+x0))))))=(x1+(x2+(x1+x2)))   [3 into 29 (r)]
add clause:
  33. (x0+(x0+(x1+x1)))=(x1+(x0+(x1+x0)))   [29 into 12 (r)]
inferences with:
  31. (x0+(x1+(x0+(x1+(x0+x2)))))=(x1+(x1+x2))   [29 into 3 (r)]
add clause:
  35. (x0+(x1+(x2+(x0+(x1+(x2+(x0+(x1+x3))))))))=(x2+(x2+x3))   [31 into 3 (r)]
add clause:
  36. (x0+(x1+(x2+(x0+(x1+(x2+(x0+x3)))))))=(x1+(x2+(x1+(x2+x3))))   [3 into 31 (r)]
add clause:
  37. (x0+(x0+(x1+(x1+x2))))=(x1+(x0+(x1+(x0+x2))))   [31 into 12 (r)]
```

```
clause has become redundant:
  24. (a+(b+(b+(a+(a+(b+(b+(a+(b+(a+(a+b)))))))))))=0 -->    [e-normalized 10]
add clause:
   0. []   [e-normalized 24]

  The EMPTY CLAUSE was produced.

hcc 4: d

Precedence:
  [ h i + 0 a b ]

Raw Clauses:

Selected Clauses:
  24. (a+(b+(b+(a+(a+(b+(b+(a+(b+(a+(a+b)))))))))))=0 -->    [e-normalized 10]

Non-selected Clauses:
   0. []   [e-normalized 24]
  27. i(x0)=(x0+x0)   [26 into 19 (r)]
  26. (x0+0)=x0   [e-normalized 22]
   1. (0+x0)=x0   [user input]
   3. ((x0+x1)+x2)=(x0+(x1+x2))   [user input]
   4. (x0+(x0+x0))=0   [user input]
  12. (x0+(x0+(x0+x1)))=x1   [4 into 3 (r)]
  29. (x0+(x1+(x0+(x1+x0))))=(x1+x1)   [11 into 12 (r)]
  31. (x0+(x1+(x0+(x1+(x0+x2)))))=(x1+(x1+x2))   [29 into 3 (r)]
  32. (x0+(x1+(x2+(x0+(x1+(x2+x0))))))=(x1+(x2+(x1+x2)))   [3 into 29 (r)]
  33. (x0+(x0+(x1+x1)))=(x1+(x0+(x1+x0)))   [29 into 12 (r)]
  36. (x0+(x1+(x2+(x0+(x1+(x2+(x0+x3)))))))=(x1+(x2+(x1+(x2+x3))))   [3 into 31 (r)]
  35. (x0+(x1+(x2+(x0+(x1+(x2+(x0+(x1+x3))))))))=(x2+(x2+x3))   [trivial by e-rewriting]
  37. (x0+(x0+(x1+(x1+x2))))=(x1+(x0+(x1+(x0+x2))))   [31 into 12 (r)]

hcc 5: quit
HCC - end
```

5.2. Example 2. A ring $R(+,\cdot)$ in which $x \cdot x = x$, for all $x \in R$, is commutative.

The input file `ref-example2`:

```
;;***************************************************
;;* A ring in which x*x = x, prove commutativity of * *
;;***************************************************

;; options (must end with ".")
(setf *simpl-cond* NIL)
(setf *use-context* NIL)
(setf *trace-level* 10)
.

;; precedence ">"
*  i +  0  a  b

;; axioms
((x0 + x1) + x2) = (x0 + (x1 + x2))
(0 + x0) = x0
(i(x0) + x0) = 0
(x0 + x1) = (x1 + x0)

((x0 * x1) * x2) = (x0 * (x1 * x2))
((x0 + x1) * x2) = ((x0 * x2) + (x1 * x2))
(x0 * (x1 + x2)) = ((x0 * x1) + (x0 * x2))
(x0 * x0) = x0

(a * b) = (b * a) -->
```

The output of **HCC** is:

```
*******************************************
*  This is HCC (common-lisp version 1.4c)  *
*******************************************

hcc 1: i
Input filename: ref-example2
hcc 2: d

Precedence:
  [ * i + 0 a b ]

Raw Clauses:
   9. (a*b)=(b*a) -->    [user input]
   2. (0+x0)=x0   [user input]
   8. (x0*x0)=x0   [user input]
   3. (i(x0)+x0)=0   [user input]
   7. (x0*(x1+x2))=((x0*x1)+(x0*x2))   [user input]
   6. ((x0+x1)*x2)=((x0*x2)+(x1*x2))   [user input]
   5. ((x0*x1)*x2)=(x0*(x1*x2))   [user input]
   1. ((x0+x1)+x2)=(x0+(x1+x2))   [user input]
   4. (x0+x1)=(x1+x0)   [user input]

Selected Clauses:

Non-selected Clauses:

hcc 3: c
```

```
...

inferences with:
 633. (x0+((x1*x0)+(x0*x1)))=x0   [e-normalized 581]
add clause:
 736. (x0*x1)=(x1*x0)   [706 into 633 (r)]
clause has become redundant:
   9. (a*b)=(b*a) -->    [user input]
add clause:
   0. []   [e-normalized 9]

  The EMPTY CLAUSE was produced.

hcc 4: quit
HCC - end
```

APPENDIX A

About HCC

HCC uses three lists for storing clauses while running the completion procedure.

`$raw-clauses`	These clauses are not reduced (with respect to the other clauses). If clauses are read from a file they are stored in this list.
`$clauses`	Here are the clauses stored which are reduced. During completion the first unmarked clause of `$clauses` is picked for computing the superpositions (with clauses from `$marked-clauses`) and resolutions. The clause is also marked and copied to `$marked-clauses`.
`$marked-clauses`	This list is a sublist of `$clauses` and contains the marked clauses, i.e. clauses for which all superpositions and resolutions have been computed. If this list is equal to `$clauses` then the completion procedure is finished.

1. Commands

HCC is written in Common Lisp [**Steele, 1990**] and was developed with "GCL (GNU Common Lisp) Version(2.1)" and CLISP under Linux 1.2.11. If GCL is used just type `gcl`, the `init.lsp` is loaded automatically. If you are working with CLISP you have to type `(load "init.lsp")` within the lisp environment. Now the files are loaded and if necessary compiled. After having typed `(hcc)` you should see the input shell of **HCC** .

HCC has the following shell commands.

`quit`	Quit **HCC** and quit the LISP environment.
`q`	Quit **HCC** .
`h`	Help.
`i`	Open an input file.
`d`	Display the lists of clauses.
`c`	Run the completion procedure.
`e`	Normalize equation.
`t`	Normalize term.
`l`	Evaluate LISP command.
`o`	Print options.
`r`	Reset **HCC** .

2. Options

The following variables can be used to change the default completion method of **HCC** . To change the value of some variable use the `l`-command of **HCC** , e.g. `(setf *add-raw-strategy* 1)`.

2.1. Add Strategies.

2.1.1. `*add-raw-strategy*`. This variable determines how clauses are added to `$raw-clauses`. Default value: `4`

1 Append the clause at the end of `$raw-clauses`.
2 Append the clause at the beginning of `$raw-clauses`.
3 Insert the clause according to its size in `$raw-clauses`. If there are already clauses of the same size, the clause is inserted at the end.
4 Insert the clause according to its size in `$raw-clauses`. If there are already clauses of the same size, the clause is inserted at the beginning.

2.1.2. `*add-strategy*`. This variable determines how clauses are added to `$clauses`. Default value: `4`

1 Append the clause at the end of `$clauses`.
2 Append the clause at the beginning of `$clauses`.
3 Insert the clause according to its size in `$clauses`. If there are already clauses of the same size, the clause is inserted at the end.
4 Insert the clause according to its size in `$clauses`. If there are already clauses of the same size, the clause is inserted at the beginning.

2.1.3. `*add-marked-strategy*`. This variable determines how clauses are added to `$marked-clauses`. Default value: `4`

1 Append the clause at the end of `$marked-clauses`.
2 Append the clause at the beginning of `$marked-clauses`.
3 Insert the clause according to its size in `$marked-clauses`. If there are already clauses of the same size, the clause is inserted at the end.
4 Insert the clause according to its size in `$marked-clauses`. If there are already clauses of the same size, the clause is inserted at the beginning.

2.2. Process Strategies.

2.2.1. `*proc-simpl-strategy*`. This variable controls how a simplified clause is treated. Default value: `2`

1 Add the clause to `$raw-clauses`.
2 Reduce the clause and check if it is redundant. If this is not the case add it to `$clauses`.

2.2.2. `*proc-deduc-strategy*`. This variable controls how a deduced (by superposition or resolution) clause is treated. Default value: `2`

1 Add the clause to `$raw-clauses`.
2 Reduce the clause and check if it is redundant. If this is not the case add it to `$clauses`.

2.3. Simplification / Deletion Techniques.

2.3.1. The following variables can either be `T` or `NIL`.

`*simpl-equ*`	`T`	Use ordered equational rewriting.
`*simpl-cond*`	`T`	Use ordered conditional rewriting.
`*use-context*`	`T`	Use contextual rewriting (in equational and/or conditional rewriting).
`*enrich-context*`	`T`	Use context enrichment.
`*del-subsume*`	`T`	Use subsumption.
`*del-case*`	`NIL`	Use case-analysis on the orderings between variables.
`*block-check*`	`NIL`	Use blocking in superposition.

2.3.2. The following variables have integer values.

`*context-enrich-max*`	`3`	Maximal number of clauses which are added during context enrichment.
`*ante-rewrite-depth*`	`0`	Maximal number of non-simplifying rewrite steps in antecedent.
`*suc-rewrite-depth*`	`2`	Maximal number of non-simplifying rewrite steps in succedent.

2.4. Selection function. There are two possible selection strategies implemented in **HCC** which are controlled by `*select-positive*`. Default value: `NIL`

`T`	Use a positive unitary strategy.
`NIL`	Select an equation in the antecedent of a clause only if the clause is non-simplifying.

2.5. Trace level. The variable `*trace-level*` controls the output of **HCC** on the. A higher value produces a more detailed output. Default value: `10`

APPENDIX B

Program Listing of HCC

The program **HCC** consists of the following files:

Main file and data definition:

`init.lisp`
`hcc.lisp`
`defs.lisp`

User input and output:

`utility.lisp`
`output.lisp`
`parse.lisp`
`shell.lisp`

Terms, equations and clauses:

`term.lisp`
`equation.lisp`
`clause.lisp`

Multisets, lexicographic recursive path orderings and substitutions:

`multiset.lisp`
`lrpo.lisp`
`substitution.lisp`

Completion loop and inference rules:

`completion.lisp`
`superposition.lisp`
`resolution.lisp`

Simplification and deletion techniques:

`subsumption.lisp`
`rewrite.lisp`
`rewrite-g.lisp`
`rewrite-r.lisp`
`rewrite-ca.lisp`

Bibliography

[Bachmair, 1991] Bachmair, L. (1991). *Canonical Equational Proofs*. Progress in theoretical computer science. Birkhäuser.

[Bachmair et al., 1986] Bachmair, L., Dershowitz, N., and Hsiang, J. (1986). Orderings for equational proofs. In *Symposium on Logic in Computer Science, Cambridge, Mass., June 16-18, 1986*, pages 346–357.

[Bachmair and Ganzinger, 1990a] Bachmair, L. and Ganzinger, H. (1990a). Completion of first-order clauses with equality by strict superposition (extended abstract). In Kaplan, S. and Okada, M., editors, *2nd International Workshop on Conditional Term Rewriting Systems, Montreal, Canada, June 11-14, 1990*, LNCS 516, pages 162–180. Springer.

[Bachmair and Ganzinger, 1990b] Bachmair, L. and Ganzinger, H. (1990b). On restrictions of ordered paramodulation with simplification. In Stickel, M., editor, *10th International Conference on Automated Deduction, Kaiserslauten, Germany, July 24-27, 1990*, LNCS 449, pages 427–441. Springer.

[Bachmair and Ganzinger, 1994] Bachmair, L. and Ganzinger, H. (1994). Rewrite-based equational theorem proving with selection and simplification. *J. of Logic and Computation*, 4:1–31.

[Bachmair et al., 1993] Bachmair, L., Ganzinger, H., Lynch, C., and Snyder, W. (1993). Basic paramodulation. Technical Report MPI-I-93-236, Max-Planck-Institut für Informatik, Im Stadtwald, Saarbrücken, Germany.

[Birkhoff, 1935] Birkhoff, G. (1935). On the structure of abstract algebras. *Cambridge Philosophical Society*, 31:433–454.

[Dershowitz, 1982] Dershowitz, N. (1982). Orderings for term-rewriting systems. *Theoret. Comput. Sci.*, 17:279–301.

[Dershowitz, 1987] Dershowitz, N. (1987). Termination of rewriting. *J. Symb. Comput.*, 3:69–116.

[Dershowitz and Jouannaud, 1990] Dershowitz, N. and Jouannaud, J.-P. (1990). Rewrite systems. In van Leeuwen, J., editor, *Handbook of Theoretical Computer Science, Volume B*, pages 244–320. Elsevier.

[Dershowitz and Manna, 1979] Dershowitz, N. and Manna, Z. (1979). Proving termination with multiset orderings. *Comm. ACM*, 22(8):465–476.

[Dershowitz and Okada, 1990] Dershowitz, N. and Okada, M. (1990). A rational for conditional equational programming. *Theoret. Comput. Sci.*, 75:111–138.

[Ganzinger, 1991] Ganzinger, H. (1991). A completion procedure for conditional equations. *J. Symb. Comput.*, 11:51–81.

[Goguen, 1987] Goguen, J. A. (1987). How to prove algebraic inductive hypotheses without induction, with applications to the correctness of data type implementation. In Bibel, W. and Kowalski, R., editors, *5th Conference on Automated Deduction, Les Arcs, France, July 1980*, LNCS 87, pages 356–373. Springer.

[Huet, 1976] Huet, G. (1976). *Résolution d'équations dans les langages d'ordre* $1, 2, \ldots, \omega$. PhD thesis, Univ. de Paris VII, Paris, France.

[Huet, 1981] Huet, G. (1981). A complete proof of correctness of the knuth–bendix completion algorithm. *J. Comput. System Sci.*, 23:11–21.

[Jouannaud and Waldmann, 1987] Jouannaud, J.-P. and Waldmann, B. (1987). Reductive conditional term rewriting systems. In Wirsing, M., editor, *Proceedings of the IFIP TC 2/WG 2.2 Working Confernce on Formal Description of Programming Concepts, Ebberup, Denmark, August 25-28, 1986*, pages 223–242. Elsevier Science Publishers B.V.

[Kaplan, 1983] Kaplan, S. (1983). *Un langage de spécifications de types abstraits algébriques.* Thèse de 3ème cycle, Univ. de Paris, Orsay, 1983.

[Kaplan, 1984] Kaplan, S. (1984). Conditional rewrite rules. *Theoret. Comput. Sci.*, 33:175–193.

[Kaplan, 1987] Kaplan, S. (1987). Simplifying conditional term rewriting systems: Unification, termination and confluence. *J. Symb. Comput.*, 4:294–334.

[Kapur and Zhang, 1989] Kapur, D. and Zhang, H. (1989). *RRL: Rewrite Rule Laboratory – User's manual.*

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

[Kounalis and Rusinowitch, 1991] Kounalis, E. and Rusinowitch, M. (1991). On word problems in horn theories. *J. Symb. Comput.*, 11:113–127.

[Martelli and Montanari, 1982] Martelli, A. and Montanari, U. (1982). An efficient unification algorithm. *ACM Trans. Programming Languages and Systems*, 4:258–282.

[Martin and Nipkow, 1990] Martin, U. and Nipkow, T. (1990). Ordered rewriting and confluence. In Stickel, M. E., editor, *10th International Conference on Automated Deduction, Kaiserslautern, Germany, July 24-27, 1990*, LNAI 449, pages 366–380. Springer.

[Nieuwenhuis and Orejas, 1992] Nieuwenhuis, R. and Orejas, F. (1992). Clausal rewriting. In Kapur, D., editor, *11th International Conference on Automated Deduction, Saratoga Springs, NY, USA, June 15-18, 1992*, LNCS 607, pages 246–258. Springer.

[Robinson, 1965] Robinson, G. A. (1965). A machine-oriented logic based on the resolution principle. *J. Assoc. Comput. Mach.*, 12:23–41.

[Steele, 1990] Steele, G. L. (1990). *Common Lisp the Language (2nd edition).* Digital Press.

[Zhang, 1992] Zhang, H. (1992). Implementing contextual rewriting. In Rusinowitch, M. and Rémy, J.-L., editors, *3rd International Workshop on Conditional Term Rewriting Systems, Pont-à-Mousson, France, July 8-10, 1992*, LNCS 656, pages 363–377. Springer.