TERM REWRITING IN ASSOCIATIVE
COMMUTATIVE THEORIES WITH
IDENTITIES

A THESIS PRESENTED

BY

Martin Joachim Henz

TO

THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE IN

COMPUTER SCIENCE

STATE UNIVERSITY OF NEW YORK

AT STONY BROOK

December 1991
State University of New York  
at Stony Brook  
The Graduate School

Martin Joachim Henz

We, the thesis committee for the above candidate for  
the degree of Master of Science,  
hereby recommend acceptance of this thesis.

__________________________
Professor Leo Bachmair, Advisor  
Computer Science

__________________________
Professor Jieh Hsiang,  
Computer Science

__________________________
Professor I.V. Ramakrishnan,  
Computer Science

This thesis is accepted by the Graduate School.

__________________________
Graduate School
Abstract of the Thesis

Term Rewriting in Associative Commutative Theories with Identities

by

Martin Joachim Henz

Master of Science
in
Computer Science
State University of New York
at Stony Brook
1991

Versions of constraint rewriting for completion of rewrite systems in the presence of associative commutative operators with identities have been proposed, in which constraints are used to limit the applicability of rewrite rules. We extend these approaches such that the initially given equations can contain constraints, and such that a suitable version of unification modulo associativity, commutativity and identity can be interleaved with the process of completion.
To

my parents Albert and Klara Henz

and

my wife Kelly Reedy.
Contents

Abstract iii

Acknowledgements ix

1 Introduction 1

2 Preliminaries 4
  2.1 Terms ........................................ 4
  2.2 Relations .................................... 8
  2.3 The Associative Commutative Theory
      with Identities ............................... 11
  2.4 Constraints ................................ 15

3 Unification 19
  3.1 Normal Forms ................................ 20
  3.2 AC-Unification ............................... 23
  3.3 AC1-Unification ............................... 36
  3.4 AC1-Unification with Zero-Disequations ......... 45

4 Completion 51
4.1 Constrained AC1-rewriting ............................. 51
4.2 Some Operations on Constraints ......................... 57
4.3 Auxiliary Transformations .............................. 57
4.4 The Transformation Rules
   AC1-Complete ........................................ 58
4.5 Application Strategies ................................. 63

5 Implementation ........................................... 65

A AC1-matching ............................................ 73

B Examples .................................................. 76
   B.1 Rewriting ........................................... 76
   B.2 Constraint Solving ................................ 79
List of Tables

1. The transformation rules **AC-Unify** .......................... 25
2. The transformation rules **AC1-Unify** .......................... 39
3. The transformation rules **Disequation-AC1-Unify**  ...... 47
4. The transformation rules **Aux** ................................. 59
5. The transformation rules **AC1-Complete** ...................... 61
6. The transformation rule **Equation Split** ...................... 63
7. The transformation rules **AC1-Match** ......................... 74
# List of Figures

1. Illustration to Proposition 4.1 .................................. 54
2. Illustration to Proposition 4.2 (Termination) .................. 55
3. Illustration to Proposition 4.2 (Church-Rosser) ............ 55
4. The structure of TRACONE .................................. 66
Acknowledgements

I take this opportunity to express my gratitude to my thesis advisor, Professor Leo Bachmair, for his continuous support and invaluable guidance he offered me in many fruitful discussions.

I thank the members of my thesis committee, Professor Jieh Hsiang and Professor I.V. Ramakrishnan, for having read and discussed this thesis.

I wish to extend my thanks to Professor Jacques Loeckx and Stefan Uhrig who supported and encouraged me throughout my studies in Saarbrücken.

I am indebted to my friends in Stony Brook, especially my office mates, for their hospitality and help, and for having turned my stay in Stony Brook into the most enjoyable experience.

I thank the Fulbright Commission for providing a scholarship that made my stay in Stony Brook possible and the Siemens AG for financial support throughout my studies.

Finally, I wish to give my most special thanks to my wife, Kelly Reedy, for “being with me in all my struggles”. I also thank her for proof-reading this thesis.
Chapter 1

Introduction

In mathematics and the sciences, terms and equations are extensively used. Computers provide tools to mechanize the handling of terms and equations. In many computer applications, including symbolic algebraic computation, automated theorem proving, program specification and verification, automated reasoning about terms and equations plays an important role.

Rewrite systems are sets of directed equations used to compute by replacing subterms in a given term until a simplest form possible, called a normal form, is obtained. A rewrite system is called canonical, if normal forms are unique and always exist. Knuth and Bendix [KB 70] proposed a procedure, called standard completion procedure, to build canonical systems by systematically generating and orienting valid equational consequences from a given set of equations.

An extension of the framework of rewrite systems is rewriting in equational theories where the replacement of subterms is generalized using equations that express properties of the used operators. Extended completion leads to the
construction of canonical systems in this framework.

In many applications, operators with the properties of associativity and commutativity occur (examples are conjunction in logic and addition of integers). A procedure for completion in the presence of associative commutative operators has been presented in [PS 81]. Here, equational consequences from equations are computed using \textit{associative-commutative unification}, a process that is in many cases particularly time-consuming.

Often, associative commutative operators are associated with an identity (for example \texttt{true} for conjunction and 0 for addition). Unfortunately, associative commutative completion often fails in the presence of identities, since severe restrictions on the orientation of equations must be imposed.

In [BPW 89], this problem is overcome by limiting the applicability of directed equations via \textit{constraints}. This approach is presented in [JM 90] in the framework of transformation rules for completion. A general approach to rewriting using constraints is presented in [KKR 91].

In [BPW 89] and [JM 90], only the resulting rewrite systems, but not the initial set of equations may contain constraints, which limits the application of this approach in practice.

We shall make use of constraints in completion in the presence of associative commutative operations with identities for two purposes. Firstly, we shall extend the approach of [BPW 89] and [JM 90] to undirected equations, such that the given sets of equations may contain constraints. Secondly, we shall provide a framework to interleave associative commutative unification with the process of completion in order to gain the flexibility of postponing particularly hard unification problems, hoping that they can be simplified by later derived
directed equations.

In Chapter 2, we present fundamental concepts, including our notion of constraints. The subject of Chapter 3 is the solving of constraints, which can be viewed as an extension of unification in the presence of associative commutative operators and identities. In Chapter 4, we present a transformation system for completion, incorporating the constraint solving methods of the previous chapter. A computer implementation of various components of completion procedures is described in Chapter 5.
Chapter 2

Preliminaries

2.1 Terms

Let $S$ be a set. When $X$ is a sequence of elements of $S$ of length $|X| = n$, we denote its components by $X^1, X^2, \ldots, X^n$. The empty sequence is denoted by $\lambda$. An element $s$ of $S$ occurs in $X$, denoted $s \in X$, if $s = X^i$ for some $i \in [1..n]$. ($[1..n]$ denotes the set $\{i \in \mathbb{N} \mid 1 \leq i \leq n\}$.) A sequence $Y$ of length $m$ is called a subsequence of a sequence $X$ of length $n$, denoted $Y \subseteq X$, if there exists a function $f$ from $[1..m]$ to $[1..n]$, such that $i > j$ implies $f(i) > f(j)$ for all $i, j \in [1..m]$ and $Y^i = X^{f(i)}$ for all $i \in [1..m]$. The number of occurrences of $s$ in $X$, denoted $|X|_s$, is the number $|\{i \mid X^i = s\}|$.

The difference of two sequences $X$ and $Y$, denoted $X - Y$, is the subsequence $Z$ of $X$ such that for all $s \in S$ the number $|Z|_s$ is $|X|_s - |Y|_s$ if $|X|_s - |Y|_s \geq 0$ and 0 otherwise. A sequence $Y$ is a permutation of a sequence $X$, if for all $s \in S$ holds $|Y|_s = |X|_s$.

Let $F$ be a set of function symbols with $F = F_{\text{free}} \uplus F_{\text{AC}} \uplus F_{\text{zero}}$. We
call the elements of $F_{\text{free}}$ free function symbols, the elements of $F_{\text{AC}}$ associative commutative function symbols, short AC-symbols, and the elements of $F_{\text{zero}}$ identities. We assume that there exists a partial function zero from $F_{\text{AC}}$ to $F_{\text{zero}}$. The subset of $F_{\text{AC}}$ that contains all associative commutative function symbols $f$ for which zero$(f)$ is defined is denoted by $F_{\text{AC1}}$; its elements are called associative commutative function symbols with identity, short AC1-symbols.

With each function symbol $f$, we associate a non-empty set $A(f) \subseteq \mathbb{N}$ that indicates the number of arguments $f$ may take. If $f \in F_{\text{zero}}$, then $A(f) = \{0\}$; if $f \in F_{\text{AC}}$, then $A(f) = \mathbb{N} - \{0, 1\}$. We assume that $A(f)$ contains only one element if $f \in F_{\text{free}}$.

Let $V$ be a countable set disjoint from $F$, whose elements we call variables. A term $t$ is either a variable or an expression $f(X)$, where $X$ is a sequence of terms and $|X| \in A(f)$. In the latter case, $f$ is called the root of $t$, denoted root$(t)$, and $X$ is called sequence of arguments of $t$, denoted args$(t)$. When root$(t) \in F_{\text{AC}}$, $t$ is called AC-term; when root$(t) \in F_{\text{AC1}}$, $t$ is called AC1-term. The set of all terms built from function symbols in $F$ and variables in $V$ is denoted by $T(F,V)$.

To enhance readability, we often use infix notation for symbols in $F_{\text{AC}}$, omit the empty sequence $\lambda$ and the parentheses around sequences of arguments of length 0 and 1.
Example 2.1 For this and all following examples, let

\[ F = F_{\text{free}} \uplus F_{\text{AC}} \uplus F_{\text{zero}} \text{ where} \]

\[ F_{\text{free}} = \{ a, b, c, d, e, - f, g \}, \quad F_{\text{AC}} = \{ \#, +, * \}, \quad F_{\text{zero}} = \{ 0, 1 \}, \]

\[ \text{zero}(+) = 0, \quad \text{zero}(*) = 1 \quad (+ \text{ and } * \text{ are AC1-symbols}), \]

\[ A(a) = A(b) = A(c) = A(d) = A(e) = 0, \quad A(\text{\neg}) = 1, \]

\[ A(f) = 1, A(g) = 2, \quad \text{and } v, w, x, y, z \in V. \]

The expressions \( r, s, t \) with

\[ r = g(\lambda, (f((g(0(\lambda), 1(\lambda)))))), \]

\[ s = + (x, y, *(a(\lambda), z)), \]

\[ t = \#(1(\lambda), f(0(\lambda))), g(+ (f(x), *(\text{\neg}(v), 0(\lambda))), w)) \]

are terms in \( T(F, V) \). The terms \( s, t \) are AC1-terms. The term \( s \) is additionally an AC1-term. In infix notation, the terms are written:

\[ r = g(1, f((g(0, 1)))), \]

\[ s = x + y + (a \ast z), \]

\[ t = 1 \# f(0) \# g(f(x) + ((-(v)) \ast 0), w). \]

The function \( \text{Var} \) maps each term to the set of variables that occur in it:

\( \text{Var}(x) = x \) if \( x \) is a variable and \( \text{Var}(f(X)) = \bigcup_{\lambda \in X} \text{Var}(X) \).

A term \( s \) is said to be a subterm of a term \( t = f(X) \) if either \( s = f(Y) \) for some subsequence \( Y \) of \( X \), or else \( s \) is a subterm of some term \( X^i \in X \). When we want to emphasize that a term \( t \) contains a term \( s \) as a subterm, we denote \( t \) by \( t[s] \). A term \( s \) is a proper subterm of a term \( t \) if \( s \) is a subterm of \( t \) and \( s \neq t \). The set of all subterms of a term \( t \) is denoted by \( \text{subt}(t) \). A subterm \( s \)
of $t$ is called a non-variable subterm of $t$, if $s \notin V$. The set of all non-variable subterms of a term $t$ is denoted by $F\text{sub}(t)$.

We use a pair $\pi = \langle p, P \rangle$, where $p$ is a sequence of of positive numbers and $P$ is a set of positive numbers, called a position, to refer in a term $t$ to a specific subterm in $t$, denoted by $t_{\pi}$. The pair $\langle \lambda, \emptyset \rangle$ is a position in any term and referring to the term itself; the pair $\langle (i, p), \emptyset \rangle$ is a position in a term $t = f(X)$ and referring to the subterm $s$ of $t$, if $\langle p, \emptyset \rangle$ is a position in $X^i$ referring to $s$; and $\langle p, P \rangle$ is a position in a term $t$ and referring to the subterm $s = g(Y)$ of $t$, if $\langle p, \emptyset \rangle$ is a position in $t$ referring to a term $g(X)$, $g(Y)$ is a subterm of $g(X)$, $2 \leq |P| < |X|$ and $i \in P$ whenever $X^i \in Y$. For instance, the term $x + z$ is the subterm of the term $a * g(x + y + z, 1))$ at position $\langle (2, 1), \{1, 3\} \rangle$. Note, that this notion of subterms and positions is an extension of the notion of “flattened positions” in [Mar 91], where as second components of positions only sets of subsequent numbers are allowed.

A position $\pi$ in $t$ is called a non-variable position in $t$ if $t|_{\pi} \notin V$. The set of all positions in $t$ is denoted by $pos(t)$ and the set of all non-variable positions in $t$ by $Fpos(t)$.

The result of replacing the subterm at position $\langle p, P \rangle$ in a term $t$ by a term $s$ is denoted by $t[s]_{\langle p, P \rangle}$ and defined to be the term $s$ if $p = \lambda$ and $P = \emptyset$; the term $f(X - Y, s)$ if $t = f(X)$, $p = \lambda$, $P \neq \emptyset$, and $f(Y)$ is the subterm of $t$ at position $\langle p, P \rangle$; and the term $f(X, t'[s]_{\langle p', P' \rangle}, Y)$ if $t = f(X, t', Y)$, $p = (i, p')$, and $i = |X| + 1$. For instance, $(g(x + y + z))[f(a) + b]_{\langle (1), \{1, 3\} \rangle} = g(y + f(a) + b)$.

A substitution $\sigma$ is a mapping from variables to terms. The value of a substitution $\sigma$ for a variable $x$ is denoted by $x\sigma$. We call the set of variables for
which $x\sigma \not= x$ the domain of $\sigma$, denoted $\text{Dom}(\sigma)$, and consider only substitutions with finite domain. We denote a substitution $\sigma$ by $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, when $\text{Dom}(\sigma) = \{x_1, \ldots, x_n\}$ and $x_i\sigma = t_i$ for $i \in [1..n]$. The set $\bigcup_{x \in \text{Dom}(\sigma)} \text{Var}(x\sigma)$ is called the range of $\sigma$, denoted $\text{Ran}(\sigma)$. If $W \subseteq V$ then $\Sigma(W)$ denotes the set of all substitutions $\sigma$ for which $\text{Dom}(\sigma) \subseteq W$. If $\sigma$ is a substitution and $W \subseteq V$, then $\sigma|_W$ denotes the restriction of $\sigma$ on $W$: $x\sigma|_W = x\sigma$ if $x \in W$ and $x\sigma|_W = x$ otherwise.

A substitution $\sigma$ can be uniquely extended to a mapping on terms in such a way that $(f(X))\sigma = f(X^1\sigma, \ldots, X^n\sigma)$, for all terms $f(X)$ where $|X| = n$. The composition of two substitutions $\sigma$ and $\tau$ is denoted by the juxtaposition $\sigma\tau$. That is, $t\sigma\tau = (t\sigma)\tau$ for all terms $t$.

A term $s$ is an instance of (or matches) another term $t$ if there exists a substitution $\sigma$, called a matching substitution, such that $s = t\sigma$. Two terms $s$ and $t$ are said to be unifiable if there exists a substitution $\sigma$, called a unifier, such that $s\sigma = t\sigma$.

Note: The notion of terms can easily be extended to many sorts (see Chapter 5), but considering only one sort simplifies the notation and imposes no restrictions on the validity of our results.

### 2.2 Relations

If $\rightarrow$ is a binary relation on a set $S$, we denote by $\leftarrow$ its inverse; by $\leftrightarrow$ its symmetric closure $\rightarrow \cup \leftarrow$; by $\rightarrow^+$ its transitive closure; by $\rightarrow^*$ its transitive-reflexive closure; and by $\leftrightarrow^*$ its symmetric-transitive-reflexive closure. The
composition of two relations $\rightarrow$ and $\leftrightarrow$ is denoted by $\rightarrow \circ \leftrightarrow$. The relation $\rightarrow$ is said to be terminating if there is no infinite sequence $t_1 \rightarrow t_2 \rightarrow t_3 \rightarrow \cdots$. A binary relation $\rightarrow$ is said to be Church-Rosser if for any two elements $r$ and $s$ with $r \leftrightarrow^* s$, there exists an element $v$ such that $r \rightarrow^* v \leftrightarrow^* s$. Terminating Church-Rosser relations are called convergent.

An equivalence relation is a reflexive, symmetric, transitive binary relation. Let $\rightarrow$ be a relation and $\diamond$ an equivalence. The relation $\rightarrow$ is called Church-Rosser modulo $\diamond$ if for all elements $r$ and $s$ with $r (\diamond \cup \leftrightarrow)^* s$, there exist terms $v$ and $w$, such that $r \rightarrow^* v \diamond w \leftrightarrow^* s$. We say that $\rightarrow$ is convergent modulo $\diamond$ if $\diamond \circ \rightarrow \circ \diamond$ is terminating and Church-Rosser modulo $\diamond$. The relation $\rightarrow$ is said to be locally confluent modulo $\diamond$, if for all elements $r, r', s, s'$ with $r' \leftarrow r \diamond s \rightarrow s'$, there exist elements $r'', s''$ such that $r' \leftarrow^* r'' \diamond s'' \rightarrow^* s'$.

An element $s \in S$ is said to be in normal form with respect to the relation $\rightarrow$ if there exists no element $s' \in S$ such that $s \rightarrow s'$. A function that assigns to every element in $S$ a normal form with respect to $R$ is denoted by $\rightarrow!$.

An ordering is an irreflexive, transitive binary relation and a quasi-ordering is a reflexive, transitive binary relation.

If $\succ$ is a strict ordering, then its reflexive closure $\succeq$ is a quasi-ordering. On the other hand, if $\succeq$ is a quasi-ordering, the corresponding equivalence $\sim$ and ordering $\succ$ are defined as follows: $s \sim t$ if $s \succeq t$ and $t \succeq s$; and $s \succ t$ if $s \succeq t$ and $t \not\succeq s$. An ordering is called well-founded if it is terminating.

**Definition 2.1** If $\succ_i$ are orderings on $S_i$ for all $i \in [1..n]$, then their lexicographic combination $(\succ_1, \succ_2, \ldots, \succ_n)$ on n-tuples in $S_1 \times S_2 \times \cdots \times S_n$ is defined
as follows:

\[(a_1, a_2, \ldots, a_n) (>_{1}, >_{2}, \ldots, >_{n}) (b_1, b_2, \ldots, b_n)\]

if there exists \(i \in [1..n]\) such that for all \(j \in [1..i - 1]\) holds \(a_j \geq_j b_j\) and \(a_i >_i b_i\), where \(\geq_j\) denotes the reflexive closure of \(>_j\).

**Proposition 2.1** If the orderings \(>_i\) are well-founded for all \(i \in [1..n]\), then their lexicographic combination \((>_{1}, >_{2}, \ldots, >_{n})\) is well-founded.

**Definition 2.2** If \(\succ\) is an ordering on \(S\), then its set extension \(\succ_{\text{set}}\) is defined as follows: \(M \succ_{\text{set}} N\), if \(M \neq N\) and for all \(s \in N - M\) there exists \(s' \in M - N\) such that \(s' \succ s\).

**Proposition 2.2** If \(\succ\) is well-founded, then its set extension \(\succ_{\text{set}}\) is well-founded on finite sets.

A **multiset** over a set \(S\) is a mapping \(M\) from \(S\) to the natural numbers. We say that \(x\) is an **element** of \(M\), denoted by \(x \in_{\text{mul}} M\), if \(M(x) > 0\). A multiset \(M\) is finite, if \(\{x \mid x \in_{\text{mul}} M\}\) is finite. We often denote a finite multiset \(M\) by \(\{A\}_s\), where \(A\) is a sequence of elements in \(S\) and \(|A|_s = M(s)\) for all \(s \in S\).

The union, intersection, and difference of multisets are defined by

\[
M_1 \cup M_2 (x) = M_1(x) + M_2(x),
\]
\[
M_1 \cap M_2 (x) = \min(M_1(x), M_2(x)),
\]
\[
M_1 - M_2 (x) = \begin{cases} M_1(x) - M_2(x) & \text{if } M_1(x) - M_2(x) \geq 0, \\ 0 & \text{otherwise}. \end{cases}
\]

**Definition 2.3** If \(\succ\) is an ordering on a multiset \(S\), then its multiset extension \(\succ_{\text{mul}}\) is defined as follows: \(M \succ_{\text{mul}} N\), if \(M \neq N\) and for all \(s \in N - M\) there exists \(s' \in M - N\) such that \(s' \succ s\).
Proposition 2.3 If $\succ$ is well-founded, then its multiset extension $\succ_{\text{mul}}$ is well-founded on finite multisets.

2.3 The Associative Commutative Theory with Identities

In order to define the associative commutative theory with identities, we shall briefly present the usual framework of equational theories.

An equation is a pair of terms, written $s \approx t$. If $E$ is a set of equations, we write $s \rightarrow_E t$ to indicate that there exists a term $w$, a position $\pi$ in $w$, a substitution $\sigma$, and an equation $u \approx v$ in $E$, such that $s = w[u\sigma]_\pi$ and $t = w[v\sigma]_\pi$. The relation $\rightarrow_E$ is called the rewrite relation induced by $E$. We say that $s$ rewrites to $t$ by $E$ if $s \rightarrow_E t$. A normal form with respect to $\rightarrow_E$ is said to be irreducible by $E$.

The symmetric-transitive-reflexive closure $\leftrightarrow_E^*$ of $\rightarrow_E$ is called the equational theory induced by $E$. A set $E$ of equations will be called a rewrite system if the corresponding rewrite relation $\rightarrow_E$ is the primary object of study. The equations of a rewrite system are also called rewrite rules.

A rewrite system $R$ is called convergent if the corresponding rewrite relation $\rightarrow_R$ is convergent. A convergent rewrite relation $R$ defines a unique normal form for every term $t$, denoted by $t \downarrow_R$.

Definition 2.4 For a given set of function symbols $F = F_{\text{free}} \cup F_{\text{AC}} \cup F_{\text{zero}}$, we define the associative commutative theory with identities of $F$, denoted by $\equiv_{\text{AC}1}$, as the equational theory induced by the set $\text{AC1}$ of
• all equations

\[ f(X, f(Y), Z) \approx f(X, Y, Z), \]

called flattening equations (and expressing associativity), where \( f \in F_{AC} \),
\( X, Y, Z \) are sequences of terms in \( T(F, V) \), \( |X| + |Z| \geq 1 \), and \( |Y| \geq 2 \),

• all equations

\[ f(X) \approx f(\Pi(X)), \]

called permutative equations (and expressing commutativity), where
\( f \in F_{AC} \), \( X \) is a sequence of terms in \( T(F, V) \), \( |X| \geq 2 \), and \( \Pi(X) \) is a
permutation of \( X \),

• all equations

\[
\begin{align*}
  f(X, 0, Y) & \approx f(X, Y) \\
  f(Z, 0) & \approx Z \\
  f(0, Z) & \approx Z,
\end{align*}
\]

called zero-equations (and expressing identity), where \( f \in F_{AC1} \),
\( 0 = \text{zero}(f) \), \( X, Y, Z \) are sequences of terms in \( T(F, V) \), \( |X| + |Y| \geq 2 \),
and \( |Z| = 1 \).

Clearly, every associative commutative theory with identities, where
\( F_{AC} \neq \emptyset \), is not terminating, due to the permutative equations. The rewrite
system consisting of all flattening equations is denoted by \( Flatten \), the rewrite
system consisting of all permutative equations by \( Perm \), and the rewrite sys-
tem consisting of all zero-equations by \( Zero \).
Example 2.2

\[
x + ((1 \ast (y \ast z)) + (0 + 1)) \rightarrow_{\text{Flatten}} x + ((1 \ast y \ast z) + (0 + 1))
\rightarrow_{\text{Flatten}} x + ((1 \ast y \ast z) + 0 + 1)
\rightarrow_{\text{Flatten}} x + (1 \ast y \ast z) + 0 + 1
\]

\[
1 \ast (x + (0 + y + (0 + 0))) \rightarrow_{\text{Zero}} x + (0 + y + (0 + 0))
\rightarrow_{\text{Zero}} x + (0 + y + 0)
\rightarrow_{\text{Zero}} x + (0 + y)
\rightarrow_{\text{Zero}} x + y
\]

Proposition 2.4 The rewrite system Flatten is convergent.

Proof. The relation \( \rightarrow_{\text{Flatten}} \) is terminating, since every application of a rule to a term decreases the number of occurrences of AC symbols in it. The relation \( \rightarrow_{\text{Flatten}} \) is Church-Rosser, as can be proved by structural induction. \( \square \)

Proposition 2.5 The rewrite system Zero is convergent.

Proof. The relation \( \rightarrow_{\text{Zero}} \) is terminating, since every application of a rule in Zero to a term \( t \) decreases the number of occurrences of identities in \( t \). The relation \( \rightarrow_{\text{Zero}} \) is Church-Rosser, as can be proved by structural induction. \( \square \)

Proposition 2.6 The rewrite system Flatten \( \cup \) Zero is convergent.

Proof. The relation \( \rightarrow_{\text{Flatten} \cup \text{Zero}} \) is terminating, since every application of a rule in Flatten \( \cup \) Zero to a term \( t \) decreases the number of occurrences of
function symbols in $t$. The Church-Rosser property then follows from Propositions 2.4 and 2.5.
\[\square\]

Additionally, we can show that $\rightarrow_{\text{Flatten} \cup \text{Zero}}$ is equal to $\rightarrow_{\text{Zero}} \circ \rightarrow_{\text{Flatten}}$.

The following technical lemma is used in Chapter 4.

**Lemma 2.1** If $s$ is the subterm of $t$ at position $\pi$ and $t' =_{AC1} t$, then there exists a position $\pi'$ in $t' \downarrow_{\text{Flatten} \cup \text{Zero}}$ such that $t' \downarrow_{\text{Flatten} \cup \text{Zero}} |_{\pi'} =_{AC1} s$.

**Proof.** (1) If $u \rightarrow_{\text{Flatten}} v \alpha u \rightarrow_{\text{Zero}} v$, then for every subterm $u'$ in $u$, there exists a subterm $v'$ in $v$ such that $u' =_{AC1} v'$. (Flatten and Zero preserve subterms up to $=_{AC1}$.) (2) If $r =_{AC1} s$ then $r \downarrow_{\text{Flatten} \cup \text{Zero}} \leftrightarrow_{\text{Perm}}^* s \downarrow_{\text{Flatten} \cup \text{Zero}} \leftrightarrow_{\text{Perm}}^*$ (Identities are eliminated, for $AC$-case see Lemma 2.3 in [Mar 91].) (3) If $r \leftrightarrow_{\text{Perm}}^* s$ then for every subterm $r'$ in $r$ there exists a subterm $s'$ in $s$ such that $r' \leftrightarrow s'$. (Every subsequence of the arguments of an $AC$-term forms a subterm.) From (1) follows that there exists a subterm $s'$ of $t \downarrow_{\text{Flatten} \cup \text{Zero}}$ such that $s =_{AC1} s'$; from (2) that $t \downarrow_{\text{Flatten} \cup \text{Zero}} \leftrightarrow_{\text{Perm}}^* t' \downarrow_{\text{Flatten} \cup \text{Zero}}$; and from (3) that there exists a subterm $s''$ in $t' \downarrow_{\text{Flatten} \cup \text{Zero}}$ such that $s'' \leftrightarrow_{\text{Perm}} s'$. \[\square\]

In this work, we want to describe methods for constructing a rewrite system $R$ for a given set of equations $E$, such that $AC1 \cup E$ and $AC1 \cup R$ define the same equational theory and the relation $=_{AC1} \circ \rightarrow_R \circ =_{AC1}$ is convergent modulo $AC1$. This task is called completion modulo $AC1$.

As shown in [BPW 89], we need to limit the applicability of rewrite rules, because for many interesting rewrite systems $R$, the relation $=_{AC1} \circ \rightarrow_R \circ =_{AC1}$ is not terminating. The following example is given in [JM 90].
Example 2.3 If $R$ contains the rule

$$-(x + y) \approx (-x) + (-y)$$

then $R/AC1$ is not terminating, as shown by the following infinite sequence of rewritings:

$$
\begin{align*}
-0 & \overset{AC1}{=} -(0 + 0) \\
& \rightarrow_R (-0) + (-0) \\
& \overset{AC1}{=} -(0 + 0) + (-0) \\
& \rightarrow_R (-0) + (-0) + (-0) \\
\vdots & \quad \vdots
\end{align*}
$$

The left hand side of a rule with a variable as argument of an $AC1$-symbol + "collapses" if instantiated with zero$(+). For every equation $r \approx s$, we limit its applicability to a term $t$ by restricting the set of substitutions $\sigma$ by which a subterm of $t$ $AC1$-matches $r$. Every equation is associated with an expression, called a constraint, describing the set of allowed substitutions. In addition to the restrictions involving the identity symbols, we shall integrate unification problems in the constraints. In Chapter 4, we shall use this approach to interleave the processes of completion and unification.

2.4 Constraints

Definition 2.5 Let $F = F_{\text{free}} \cup F_{AC} \cup F_{\text{zero}}$ and $V$ a set of variables. A constraint is an expression of one of the following forms:

- A zero-disequation is an expression of the form $t \neq 0$, where $t \in T(F, V)$ and $0 \in F_{\text{zero}}$. 
CHAPTER 2. PRELIMINARIES

- An atomic AC1-unification problem is an expression of the form \( r \not\equiv s \), where \( r, s \in T(F, V) \). Zero-disequations and atomic AC1-unification problems are called atomic constraints.

- A conjunctive constraint is an expression \( c = \bigwedge_{i \in [1..n]} a_i \), where \( a_i \) are atomic constraints for \( i \in [1..n] \).

- An existential constraint is an expression \( e = \exists \vec{z}.(c, k) \), where \( \vec{z} \) is a sequence of pairwise distinct variables in \( V \), and \( c, k \) are conjunctive constraints.

  (We omit the quantification \( \exists \vec{z} \), if \( \vec{z} \) is empty. Variables in \( \vec{z} \) are called bound, other variables occurring in \( c_1 \) or \( c_2 \) are called free. The conjunctive constraint \( c \) is called the unsolved part and \( k \) is called the solved part of the existential constraint.)

- A disjunctive constraint is an expression \( d = \bigvee_{i \in [1..n]} e_i \), where \( e_i \) are existential constraints for \( i \in [1..n] \).

**Definition 2.6** For a constraint \( C \), the set of constrained variables of \( C \), denoted \( V(C) \), is defined as follows: When \( a = t \neq 0 \) is a zero-disequation, then \( V(a) = Var(t) \); when \( a = r \not\equiv s \) is an atomic AC1-unification problem, then \( V(a) = Var(r) \cup Var(s) \); when \( c = \bigwedge_{i \in [1..n]} a_i \) is a conjunctive constraint, then \( V(c) = \bigcup_{i \in [1..n]} Var(a_i) \); when \( e = \exists \vec{z}.(c, k) \) is an existential constraint, then \( V(e) = (V(c) \cup V(k)) - \vec{z} \); and when \( d = \bigvee_{i \in [1..n]} e_i \) is a disjunctive constraint, then \( V(d) = \bigcup_{i \in [1..n]} Var(e_i) \).

**Definition 2.7** For a constraint \( C \), the set of solutions of \( C \), denoted \( Sol(C) \), is defined as follows: When \( a = t \neq 0 \) is a zero-disequation, then \( Sol(a) = \)
\{ \sigma \in \Sigma(V(t)) \mid \text{to} \sigma \neq_{AC1} 0 \}; \text{ when } a = r \equiv s \text{ is an atomic } AC1\text{-unification problem, then } Sol(a) = \{ \sigma \in \Sigma(V(t)) \mid r \sigma =_{AC1} s \sigma \}; \text{ when } c = \bigwedge_{i \in [1..n]} a_i \text{ is a conjunctive constraint, then } Sol(c) = \bigcap_{i \in [1..n]} Sol(a_i); \text{ when } e = \exists \exists(c, k) \text{ is an existential constraint, then } Sol(e) = \{ \sigma_{|V(e)} \mid \sigma \in Sol(e) \text{ and } \sigma \in Sol(k) \}; \text{ when } d = \bigvee_{i \in [1..n]} e_i \text{ is a disjunctive constraint, then } Sol(d) = \bigcup_{i \in [1..n]} Sol(e_i).

Remarks:

- The symbol $^\equiv$ is used in a commutative way: $r \equiv s$ is considered to be equal to $s \equiv r$.

- The symbols $\wedge$ and $\vee$ are used in an associative commutative way: $r \equiv s \wedge c$ denotes that either $r \equiv s$ or $s \equiv r$ occurs in a conjunctive constraint and the remainder of this constraint is denoted by $c$; $e \vee d$ denotes that $e$ occurs in a disjunction and the remainder of this disjunction is denoted by $d$.

- The empty conjunctive constraint is denoted by $\top$, and the empty disjunctive constraint is denoted by $\bot$.

- We may denote conjunctions $a_1 \wedge \cdots \wedge a_n$ by $\bigwedge_{i \in [1..n]} a_i$ and disjunctions $e_1 \vee \cdots \vee e_n$ by $\bigvee_{i \in [1..n]} e_i$.

- Let $c = \bigwedge_{i \in [1..n]} r_i \equiv s_i$ be a conjunction of atomic unification problems. We denote the set $\bigcup_{i \in [1..n]} r_i \cup \bigcup_{i \in [1..n]} s_i$ by $\cup c$; the set $\bigcup_{e \in [1..\omega]} \text{subt}(r_i) \cup \text{subt}(s_i)$ by $\text{subt}(c)$; and the set $\bigcup_{e \in [1..\omega]} \text{Fsubt}(r_i) \cup \text{Fsubt}(s_i)$ by $\text{Fsubt}(c)$. 

• We may interpret a conjunction \( x_1 \vdash t_1 \land \cdots \land x_n \vdash t_n \), where \( x_i \neq x_j \) if \( i \neq j \), as a substitution \( \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\} \) and vice versa.

• We extend the function \( \text{Var} \) from terms to existential constraints:

\[
\text{Var}(\exists z'(\wedge_{i \in [1..k]} t_i \neq 0_i \land \wedge_{i \in [1..m]} r_i \vdash s_i \land \wedge_{i \in [1..m]} t'_i \vdash 0_i' \land \wedge_{i \in [1..m]} r'_i \vdash s'_i)) = z' \cup \bigcup_{i \in [1..k]} \text{Var}(t_i) \cup \bigcup_{i \in [1..m]} (\text{Var}(r_i) \cup \text{Var}(s_i)) \cup \bigcup_{i \in [1..m]} (\text{Var}(t'_i) \cup \text{Var}(r'_i) \cup \text{Var}(s'_i)),
\]

and to disjunctive constraints:

\[
\text{Var}(\bigvee_{h \in [1..k]} e_h) = \bigcup_{h \in [1..k]} \text{Var}(e_h).
\]

For a given set of function symbols \( F \) and a set of variables \( V \), we denote the set of all constraints by \( C_{AC1} \) and call its elements \( AC1\)-unification problems with zero-disequations. When a constraint in \( C_{AC1} \) contains no zero-disequations, we call it an \( AC1\)-unification problem. When \( F = F_{free} \uplus F_{AC} \), i.e., \( F_{zero} = \emptyset \), we denote the set of all constraints by \( C_{AC} \), and call its elements \( AC\)-unification problems. Note that the sets \( C_{AC1} \) and \( C_{AC} \) are constraint languages in the sense of [Smo 89].
Chapter 3

Unification

Before we use constraints to define constrained rewriting, we shall outline in this chapter how to solve them. We approach the problem of finding the set of all solutions for an $AC1$-unification problem with zero-disequations. Prior to describing methods of solving an $AC1$-unification problem with zero-disequations in the general case in Section 3.4, we shall present the solving of an $AC$-unification problem in Section 3.2 and the solving of an $AC1$-unification problem in Section 3.3. A computer implementation of the procedures presented in this chapter is described in Chapter 5 and examples are given in Appendix B.

$AC$- and $AC1$-unification problems are inherently complex. Following the initial work in [LS 75] and [Sti 81] in $AC$-unification, considerable research has been done. The reader may consult [JK 91] for an extensive list of references.

The approach presented herein is not intended to optimize the running time of the solution of a given $AC1$-unification problem but rather to modularize the very process of solving such problems. Our goal is to have the capability of
interrupting the process at as many stages as possible and “shelve” partially solved problems for future resolution. Meanwhile, the problem may have become less complex. This approach can be particularly useful in the process of completion of term rewriting systems (see Chapter 4), where unification problems are computed and continuously transformed by simplification rules. The main difficulty in modularizing the solving of AC1-unification problems is to prove the termination of the resulting procedures. A considerable part of this chapter shall be devoted to termination proofs.

In this chapter, we shall obey the following normalization convention: All terms reduced to normal forms with respect to the rewrite system Flatten. Any operation \( op \) resulting in terms is implicitly seen as the composition \( op \downarrow_{\text{Flatten}} \). This will guarantee that all occurring terms are normal forms with respect to Flatten.

**Example 3.1** The application of the substitution \( \sigma = x \mapsto z + w \) to the term \( x + y \) results in \( z + w + y \), not in \( (z + w) + y \).

### 3.1 Normal Forms

The set of all solutions for a unification problem is infinite in most cases and we are confronted with the problem of finding a finite representation for it. The following notation is taken from [JK 91] and adapted for our purpose.

When a term \( t \) AC1-matches a term \( s \), we write \( s \sqsubseteq t \). The relation \( \sqsubseteq \) is a quasi-ordering on terms, called *AC1-subsumption*, whose equivalence \( \equiv \) and strict ordering \( < \) are respectively called *literal AC1-similarity* and *strict AC1-subsumption*.
CHAPTER 3. UNIFICATION

We lift subsumption from terms to substitutions. Two substitutions \( \sigma \) and \( \tau \) are \( AC1\)-equal on the set of variables \( W \subseteq V \), denoted \( \sigma =^W_{AC1} \tau \), if \( x\sigma =^C x\tau \) for all variables \( x \) in \( W \). We write \( \sigma \preceq^W \tau \), if there exists a substitution \( \rho \) such that \( \sigma \rho =^W_{AC1} \tau \). We call the quasi-ordering \( \preceq^W \) on substitutions \( AC1\)-subsumption, its equivalence \( \preceq^W \) literal \( AC1\)-similarity and its strict ordering \( \prec^W \) strict \( AC1\)-subsumption.

**Definition 3.1** Given an \( AC1\)-unification problem with zero-disequations \( C \), every set \( CSU(C) \) that fulfills

- \( CSU(C) \subseteq Sol(C) \), \hspace{1cm} \text{(correctness)}
- for all substitutions \( \theta \in Sol(C) \) there exists a substitution \( \sigma \in CSU(C) \) such that \( \sigma \preceq_{\text{Var}(C)} \theta \), \hspace{1cm} \text{(completeness)}
- \( \text{Ran}(\sigma) \cap \text{Dom}(\sigma) = \emptyset \) for all substitutions \( \sigma \in CSU(C) \), \hspace{1cm} \text{(idempotency)}

is called a complete set of \( AC1\)-unifiers for \( C \) and denoted by \( CSU(C) \).

A set \( CSU(C) \) is called a complete set of most general \( AC1\)-unifiers of \( C \), denoted \( CSMGU(C) \), if the substitutions are pairwise incomparable in the quasi-ordering \( \preceq_{\text{Var}(C)} \).

The goal in solving an \( AC1\)-unification problem with zero-disequations is the transformation of \( C \) into a form from which a finite \( CSU(C) \) can easily be derived.
Definition 3.2 An AC1-unification problem is said to be in solved form if it is of the form:

\[ \exists z_1. \ (\exists i \in [1..m_1] \ x_{1,i} \neq 0_{1,i} \land \exists \in [1..m_1] \ y_{1,j} \neq t_{1,j}) \ \lor \]
\[ \vdots \ \\
\exists z_k. \ (\exists i \in [1..m_k] \ x_{k,i} \neq 0_{k,i} \land \exists \in [1..m_k] \ y_{k,j} \neq t_{k,j}) \]

such that for all \( h \in [1..k] \), for all \( i, i' \in [1..n_h] \) and \( j, j' \in [1..m_h] \) holds:

- \( x_{h,i}, y_{h,j} \in V \), \hspace{1cm} (solved)
- if \( t_{h,j} = 0 \), where \( 0 \in F_{zero} \), then there exists no \( i'' \in [1..n_h] \) such that \( x_{h,i''} = y_{h,j} \) and \( 0_{h,i''} = 0 \), \hspace{1cm} (disequations not violated)
- \( y_{h,j} \neq y_{h,j'} \) if \( j \neq j' \), \hspace{1cm} (unique)
- \( y_{h,j} \not\in \text{Var}(t_{h,j'}) \). \hspace{1cm} (idempotent)

Proposition 3.1 Let

\[ C = \exists z_1. \ (\exists i \in [1..m_1] \ x_{1,i} \neq 0_{1,i} \land \exists \in [1..m_1] \ y_{1,j} \neq t_{1,j}) \ \lor \]
\[ \vdots \ \\
\exists z_k. \ (\exists i \in [1..m_k] \ x_{k,i} \neq 0_{k,i} \land \exists \in [1..m_k] \ y_{k,j} \neq t_{k,j}) \]

be an AC1-unification problem in solved form. The set of substitutions that are obtained for every \( h \in [1..k] \) by limiting the substitution \( \exists \in [1..m_h] \ y_{h,j} \neq t_{h,j} \) to the variables that do not occur in \( z_h \) is a complete set of AC1-unifiers for \( C \).

An atomic unification problem \( x \neq t \) in the solved part of an existential constraint is ignored when \( x \in \bar{z} \), because bound variables do not contribute to substitutions in \( \text{Sol}(C) \).
Note that we solve an AC1-unification problem $C$ to obtain a $CSU(C)$, not a $CSMGU(C)$. In cases, where a $CSMGU(C)$ is required, we must detect redundant unifiers in an additional pass over the obtained $CSU(C)$. This operation can be very time-consuming. For this reason, we operate with $CSU(C)$, accepting a certain overhead produced by redundant unifiers.

### 3.2 AC-Unification

In this section, we consider sets $F = F_{\text{free}} \uplus F_{\text{AC}}$ (with no identities declared). Our notations are compatible with AC1-unification problems with zero-disequations so that we can extend the methods that we describe here to Section 3.3 and Section 3.4.

In the transformation of an AC1-unification problem, atomic AC1-unification problems of various forms occur. The most difficult case consists of a problem with the same AC-symbol on both sides. The decomposition step for this case forms the heart of most AC-unification procedures. In [Fag 84], an algorithm to decompose these atomic AC1-unification problems is presented. Based on this algorithm, we define a *simplifier for AC-unification problems*.

**Definition 3.3** A *simplifier for AC-unification problems* is a partial function

$$dio_{\text{AC}}: C_{\text{AC}} \times 2^V \rightarrow 2^{C_{\text{AC}}}$$

such that for all $f \in F_{\text{AC}}$ and $W \subseteq V$,

$$dio_{\text{AC}}(\exists \bar{z}.(f(X) \equiv f(Y), T), W) = \begin{cases}
\exists z_1. (\land_{i=1}^{n_1} r_{1,i} \equiv s_{1,i}, T), \\
\exists z_2. (\land_{i=1}^{n_2} r_{2,i} \equiv s_{2,i}, T), \\
\vdots \\
\exists z_k. (\land_{i=1}^{n_k} r_{k,i} \equiv s_{k,i}, T)
\end{cases}$$
where
\[
\text{Sol}(\exists \exists (f(X) = f(Y), T)) = \bigcup_{h \in [1..k]} \text{Sol}(\exists z_h \forall \left( \bigwedge_{i \in [1..m_h]} r_{h,i} = s_{h,i}, T \right))
\]
and for all \( h \in [1..k] \), for all \( i, i' \in [1..m_h] \)
\begin{enumerate}
\item \( z \subseteq z_h \),
\item \( r_{h,i} \in X \cup Y \) (see the algorithm \texttt{trans} in [Fag 84]),
\item \( r_{h,i} \neq r_{h,i'} \) (see the algorithm \texttt{elimcom} in [Fag 84]),
\item if \( r_{h,i} \in V \) then \( s_{h,i} \in X \cup Y \) or \( s_{h,i} = f(Z), Z \subseteq (V - W) \cup X \cup Y \) (see the algorithm \texttt{trans} in [Fag 84]),
\item if \( r_{h,i} \notin V \), then \( r_{h,i} \) and \( s_{h,i} \in X \cup Y \) (see Proposition 1 in [Fag 84]).
\end{enumerate}

Table 1 shows the set \( \textbf{AC-Unify} \) of transformation rules whose goal is to transform any \( AC \)-unification problem \( C \) into an equivalent \( AC \)-unification problem in solved form. Each rule describes a binary relation on constraints. The union of these relations is denoted by \( \Rightarrow_{\text{AC-Unify}} \).

**Definition 3.4** An \( AC \)-unification procedure is a subset \( U \) of the relation \( \Rightarrow_{\text{AC-Unify}} \) such that every normal form with respect to \( U \) is a normal form with respect to \( \Rightarrow_{\text{AC-Unify}} \).

An \( AC \)-unification procedure applies the rules in \( \textbf{AC-Unify} \) to an \( AC \)-unification problem until no further rules are applicable.

**Lemma 3.1 (Soundness)** Every \( AC \)-unification procedure preserves the unifiers: If \( C_1 \Rightarrow^*_{\text{AC-Unify}} C_2 \) then \( \text{Sol}(C_1) = \text{Sol}(C_2) \).
 Delete:
\[ \exists z.(r \equiv s \land c, k) \lor d \Rightarrow \exists z.(c, k) \lor d \]
if \( r =_{AC1} s \)

 Fail:
\[ \exists z.(f(X) \equiv g(Y) \land c, k) \lor d \Rightarrow d \]
if \( f \neq g \)

 Merge:
\[ \exists z.(x \equiv r \land x \equiv s \land c, k) \lor d \Rightarrow \exists z.(x \equiv r \land r \equiv s \land c, k) \lor d \]
if \( \{ x \in V \}
\quad \{ r \notin V \}

 Check:
\[ \exists z.(x \equiv f(X) \land c, k) \lor d \Rightarrow d \]
if \( \{ x \in V \}
\quad \{ x \in Var(f(X)) \}

 Eliminate:
\[ \exists z.(x \equiv t \land c, k) \lor d \Rightarrow \exists z.(c \sigma, x \equiv t \land k \sigma) \lor d \]
where \( \sigma = \{ x \mapsto t \} \)
if \( \{ x \in V \}
\quad \{ x \notin Var(t) \}

 Free Decompose:
\[ \exists z.(f(X) \equiv f(Y) \land c, k) \lor d \Rightarrow \exists z.(\land_{i \in [1..|\mathcal{X}|]} X^i \equiv Y^i, k) \lor d \]
if \( f \in F_{free} \)
\[ \{ f(X) \neq_{AC1} f(Y) \}
\quad \{ f(X) \notin \text{sub}(c) \text{ or } f(Y) \notin \text{sub}(c) \}

 AC Decompose:
\[ \exists z.(f(X) \equiv f(Y) \land c, k) \lor d \Rightarrow \]
\[ \lor_{(\exists z.(c', \tau)) \in \text{dio}_{AC}(\exists z.(f(X) \equiv f(Y) \land \tau), W)} \exists z.(c' \land c, k) \lor d \]
where \( W = \text{Var}(\exists z.(f(X) \equiv f(Y) \land c, k)) \)
if \( f \in F_{AC} \)
\[ \{ f(X) \neq_{AC1} f(Y) \}
\quad \{ f(X) \notin \text{sub}(c) \text{ or } f(Y) \notin \text{sub}(c) \}

 Table 1: The transformation rules AC-Unify
**Proof.** For every transformation rule $R$ in AC-Unify, we prove that if $C_1 \Rightarrow_R C_2$ then $\text{Sol}(C_1) = \text{Sol}(C_2)$. The lemma follows by induction.

- **Delete.** If $r =_{AC1} s$ then any substitution is in $\text{Sol}(r \leftarrow s)$, so that for $C_1 = \exists \Sigma_r(x_r \leftarrow s \land c, k) \lor d$ and $C_2 = \exists \Sigma_s(c, k) \lor d$ holds $\text{Sol}(C_1) = \text{Sol}(C_2)$.

- **Fail.** If $C_1 = \exists \Sigma_r(f(X) \leftarrow g(Y) \land c, k) \lor d$ and $f \neq g$, then $\text{Sol}(f(X) \leftarrow g(Y)) = \emptyset$, because, since $F_\varnothing = \emptyset$, AC1 does not change the root symbol of any term. Thus, for $C_2 = d$, we have $\text{Sol}(C_1) = \text{Sol}(C_2)$.

- **Merge.** If $C_1 = \exists \Sigma_r(x_r \leftarrow r \land x \leftarrow r \land c, k) \lor d$ and $C_2 = \exists \Sigma_s(x \leftarrow r \land r \leftarrow s \land c, k) \lor d$, then $\sigma \in \text{Sol}(x_r \leftarrow r \land x \leftarrow r) \iff x \sigma =_{AC1} r \sigma$ and $x \sigma =_{AC1} s \sigma \iff x \sigma =_{AC1} r \sigma$ and $r \sigma =_{AC1} s \sigma$ ($=_{AC1}$ is transitive) $\iff \sigma \in \text{Sol}(x \leftarrow r \land r \leftarrow s)$. Therefore $\text{Sol}(x \leftarrow r \land x \leftarrow s) = \text{Sol}(x \leftarrow r \land r \leftarrow s)$.

- **Check.** Let $C_1 = \exists \Sigma_r(x \leftarrow f(X) \land c, k) \lor d$ and $x \in \text{Var}(f(X))$. Assume $\sigma \in \text{Sol}(x \leftarrow f(X))$. Since $F_\varnothing = \emptyset$, AC1 is non-collapsing. On the other hand $x \sigma$ is a proper subterm of $f(X)\sigma$, which is a contradiction. Thus, $\text{Sol}(x \leftarrow t) = \emptyset$, and for $C_2 = d$, we obtain $\text{Sol}(C_1) = \text{Sol}(C_2)$.

- **Eliminate.** Let $C_1 = \exists \Sigma_r(x \leftarrow t \land c, k) \lor d$ and $C_2 = \exists \Sigma_r(c, x \leftarrow t \land k \sigma) \lor d$, where $\sigma = \{ x \mapsto t \}$. Let $\rho \in \text{Sol}(x \leftarrow t \land c, k)$. First, we state $x \rho =_{AC1} t \rho$. Let $r \leftarrow s$ be any atomic AC1-unification problem in $c$ or $k$. Then $\rho \in \text{Sol}(r \leftarrow s)$, and thus $r \rho =_{AC1} s \rho$. We have $\sigma \rho =_{AC1} \rho$, i.e. $y \sigma \rho =_{AC1} y \rho$ for all $y \in V$. (If $y \neq x$, then $\sigma$ does not change $y$; if $y = x$, then $y \sigma = t$ and $t \rho =_{AC1} x \rho = y \rho$) $r \rho =_{AC1} s \rho \iff r \sigma \rho =_{AC1} s \sigma \rho$, and therefore $\rho \in \text{Sol}(r \leftarrow s) \iff \rho \in \text{Sol}(r \leftarrow s \sigma)$. Thus, $\text{Sol}(x \leftarrow t \land c, k) = \text{Sol}(c, x \leftarrow t \land k \sigma)$ and $\text{Sol}(C_1) = \text{Sol}(C_2)$.
• **Free Decompose.** Let $C_1 = \exists \exists^*(f(X)^{\geq} f(Y) \land c, k) \lor d, f \in F_{\text{free}},$ and $C_2 = \exists \exists^*(\forall i \in [1..n] X_i^{\geq} Y^i \land c, k) \lor d$. The theory $=_{AC1}$ does not involve the symbol $f$. Thus, for a substitution $\theta$ to fulfill $f(X^1, \ldots, X^n)\theta =_{AC1} f(Y^1, \ldots, Y^n)\theta$, we must have $X_i^i \theta =_{AC1} Y^i \theta$, for $i \in [1..n]$. Therefore $\text{Sol}(\exists \exists^*(f(X)^{\geq} f(Y) \land c, k)) = \text{Sol}(\exists \exists^*(\forall i \in [1..n] X_i^{\geq} Y^i \land c, k))$ and $\text{Sol}(C_1) = \text{Sol}(C_2)$.

• **AC Decompose.** Let $C_1 = \exists \exists^*(f(X)^{\geq} f(Y) \land c, k) \lor d, \text{root}(r) = \text{root}(s) \in F_{AC}$, and $C_2 = \bigvee (\exists \exists^*(c', \top) \in \text{dio}_{AC}(\exists \exists^*(f(X)^{\geq} f(Y), \top), W) \exists \exists^*(c' \land c, k) \lor d, \text{where} \ W = \text{Var}(\exists \exists^*(f(X)^{\geq} f(Y) \land c, k))$. By definition of $\text{dio}_{AC}$ (see Definition 3.3), we obtain that $\text{Sol}(\exists \exists^*(f(X)^{\geq} f(Y), \top) = \bigcup \text{Sol}(e)$. By distributivity, we obtain that $\text{Sol}(\exists \exists^*(f(X)^{\geq} f(Y) \land c, k)) = \text{Sol}(\bigvee (\exists \exists^*(c', \top) \in \text{dio}_{AC}(\exists \exists^*(f(X)^{\geq} f(Y), \top), W) \exists \exists^*(c' \land c, k))$. Here, we use that the variables introduced by **AC Decompose** do not occur in $C_1$.

\[\square\]

**Lemma 3.2 (Completeness)** Every normal form with respect to AC-Unify is an AC-unification problem in solved form.

**Proof.** Clearly, every normal form is of the form

\[C = \bigvee_{h \in [1..k]} \exists \exists^*(c_h, k_h)\]

where $c_h$ and $k_h$ are conjunctive constraints for all $h \in [1..k]$. 
Assume \( c_i \) contains an atomic AC1-unification problem \( r \approx s \). Then \( r \) and \( s \) cannot be both non-variable terms, for otherwise Delete, Fail, Free Decompose, or AC Decompose would apply. Note, that if all atomic AC1-unification problems in \( c_i \) are of the form \( f(X) \approx f(Y) \) then Free Decompose or AC Decompose applies to at least one of them. (In a set of terms, not all terms can be proper subterms of other terms in the set.)

So, assume that \( r = x \in V \). If \( s = x \), Delete applies. If \( s \neq x \), then Check applies if \( x \in \var{var}(s) \) and Eliminate applies otherwise. Since AC-unification problems do not contain zero-disequations, \( c_i = \top \), for all \( i \), and \[ C = \forall_{h \in [1..k]} \exists z_1^h. (\top ; \forall_{i \in [1..n_h]} x_{h,i} \approx t_{1,i}) \]. The only rule that changes the solved part of an existential constraint is Eliminate. The rule Eliminate removes an atomic AC1-unification problem of the form \( x \approx t \), where \( x \in V \), from the unsolved part. The variable \( x \) is eliminated from the unsolved and the solved part, and \( x \approx t \) is added to the solved part. The variable \( x \) will not occur in the unsolved part again, because AC Decompose uses only variables in the solved part and variables that do not occur in the existential constraint, and the other rules use only variables that occur in the solved part. If a variable \( x \) occurs on the left hand side of an atomic AC1-unification problem in an existential constraint \( e \) in \( C \), then \( x \) occurs nowhere else in \( e \) (idempotency and uniqueness).

We conclude that \( C \) is of the form \[ C = \forall_{h \in [1..k]} \exists z_1^h. (\top ; \forall_{i \in [1..n_h]} x_{h,i} \approx t_{1,i}) \] such that for all \( h \in [1..k] \) and \( i, i' \in [1..n_h], x_{h,i} \in V, x_{h,i} \neq x_{h,i'} \text{ if } i \neq i', \text{ and } x_{h,i} \notin \var{var}(t_{1,i'}) \). \( \Box \)
CHAPTER 3. UNIFICATION

Not all AC-unification procedures are terminating (for an example see [Fag 84]). One way to guarantee termination is to apply the decomposition step recursively in a sense that once an atomic AC1-unification problem $f(X)^2 \equiv f(Y)$ is simplified, all emerging subproblems are completely solved before another atomic problem is examined ([Sti 81], [Fag 84]). This method is not satisfactory for our approach to completion modulo AC1, where we want to be able to interrupt the solving of a constraint when a particularly hard atomic AC1-unification problem arises. Other approaches to AC-unification are described in [Kir 89] and [Bou 90].

We shall present a class of terminating AC-unification procedures that are based on [Fag 84] but can be interrupted between every application of AC Decompose. We require that, after the application of Free Decompose and AC Decompose, some rules are “eagerly” applied. To prove the termination of these AC-unification procedures, we develop an extension of the complexity used in the termination proof of [Fag 84].

In the following, let $@$ be a new symbol. The set of immediate operators of a variable $x$ in a term $t$ is the set

$$OP(x,t) = \begin{cases} \{\text{root}(t|_{p,0}) \mid (p,0) \in \text{pos}(t) \text{ and } \exists i \in \mathbb{N}, t|_{(p,i),0} = x\} & \text{if } x \neq t, \\ \{@\} & \text{if } x = t. \end{cases}$$

(We include the occurrence of a variable in the topmost position: If $x = t$, we say $x$ occurs in $t$ under $@$.)

The set of immediate operators of a variable $x$ in an existential constraint

$$e = \exists z. (\bigwedge_{i \in [1..n]} x_i \neq 0_i \land \bigwedge_{j \in [1..m]} r_{j}^{2} \equiv s_{j} \land k)$$

...
CHAPTER 3. UNIFICATION

is the set

\[ OP(x, e) = \bigcup_{j \in [1..m]} OP(x, r_j) \cup OP(x, s_j), \]

**Definition 3.5** The complexity \( \mu_e \) of an existential AC1 constraint \( e = \exists \mathcal{Z} (c \land g, k) \), where \( g = \bigwedge_{i \in [1..n]} x_i \neq 0 \) and \( c = \bigwedge_{j \in [1..n]} r_j \neq s_j \), is defined as

\[ \mu_e(e) = (\alpha(e), \beta(e), \gamma(e), \delta(e), \varepsilon(e), \zeta(e), \eta(e)), \]

where

- \( \alpha(e) = \{ \{i \mid r_i \in V \text{ and } s_i \in V \} \} \), \( \alpha(e) \) is the number of atomic AC1-unification problems between variables in the unsolved part;

- \( \beta(e) = \{ \{x \in V \mid |OP(x, e)| \geq 2\} \} \), \( \beta(e) \) is the number of distinct variables that occur in \( r_i \) and \( s_i \) immediately under at least two different function symbols;

- \( \gamma(e) = |F_{\text{subl}}(e) - \{0(\lambda) \in T(F, V) \mid 0 \in F_{\text{zero}}\}| \), \( \gamma(e) \) is the number of distinct non-variable subterms of \( r_i \) and \( s_i \);

- \( \delta(e) = \{ \{i \mid r_i \in V \text{ or } s_i \in V \} \} \), \( \delta(e) \) is the number of atomic AC1-unification problems in the unsolved part of which at least one side consists of a variable;

- \( \varepsilon(e) = \{ \{i \mid r_i \notin V, r_i \notin \text{subl}(e) - \bigcup c, r_i \notin F_{\text{zero}}\} \} + \{ \{i \mid s_i \notin V, s_i \notin \text{subl}(c) - \bigcup c, s_i \notin F_{\text{zero}}\} \} \), \( \varepsilon(e) \) is the number of occurrences of non-variable terms on one side of an atomic problem in \( c \) that are not identities and not proper subterms of terms in \( e \);

- \( \zeta(e) = \sum_{i \in [1..m]} \left( \{ \pi \in \text{pos}(r_i) \mid r_i \notin F_{\text{zero}}\} + \{ \pi \in \text{pos}(s_i) \mid s_i \notin F_{\text{zero}}\} \right) \), \( \zeta(e) \) is the number of all positions in \( e \) at which no identities occur; and
\* \( \eta(e) = n \), \( \eta(e) \) is the number of atomic AC-unification problems in \( e \).

The complexity \( \mu_d \) of a disjunctive AC-unification problem \( d = \bigvee_{i \in [1..n]} e_i \) is defined as the multiset \( \mu_d(d) = \{ \mu_e(e_i) \mid i \in [1..n] \}_{mul} \).

**Example 3.2** The constraint

\[
e = \exists v,w.(x + v + w = f(a) + g(f(a), x) \land v = x \land x * f(v) = 1 \land f(a), y = b)
\]

has the complexity \( \mu_e(e) = (1, 2, 6, 3, 3, 20, 4) \). As a disjunctive constraint, \( e \) has the complexity \( \mu_d(e) = \{(1, 2, 6, 3, 3, 20, 4)\}_{mul} \).

We use the seven-fold lexicographic combination of the “greater-than”-ordering \( > \) on natural numbers, denoted \( >_e \), to compare complexities of existential constraints, and the multiset extension of \( >_e \), denoted \( >_d \), to compare complexities of disjunctive constraints. Propositions 2.1 and 2.3 imply that \( >_e \) and \( >_d \) are well-founded.

**Lemma 3.3 (Termination)** Every AC-unification procedure in which, after every application of Free Decompose and AC Decompose, all newly introduced atomic problems with variables on at least one side are eliminated via Delete, Check and Eliminate terminates on every AC-unification problem.

**Proof.** We prove that if \( C_1 \Rightarrow_R C_2 \) for a rule \( R = \text{Delete, Fail, Merge, Check, or Eliminate} \) then \( \mu_d(C_1) >_d \mu_d(C_2) \); and if \( C_1 \Rightarrow_R C'_2 \) for a rule \( R = \text{Free Decompose, AC Decompose, and } C_2 = \Rightarrow^*_Q (C'_2) \), where \( Q = \text{Delete} \cup \text{Check} \cup \text{Eliminate} \), then \( \mu_d(C_1) >_d \mu_d(C_2) \). The termination follows by induction on the number of applications of rules in AC-Unify to the given AC-unification problem.
\section*{Chapter 3. Unification}

- **Delete.** If \( e_1 = \exists z.(r \equiv s \land c, k), e_2 = \exists z.(c, k), C_1 = e_1 \lor d, \) and \( C_2 = e_2 \lor d, \) then \( \Gamma(e_1) \geq \Gamma(e_2), \) where \( \Gamma \in \{\alpha, \beta, \gamma, \delta, \varepsilon, \zeta\}, \) while \( \eta(e_1) > \eta(e_2), \) and therefore \( \mu_d(C_1) >_d \mu_d(C_2). \)

- **Fail.** If \( C_1 = \exists z.(f(X) \equiv g(Y) \land c, k) \lor d, \) and \( C_2 = d, \) then \( \mu_d(C_1) = \mu_d(C_2) \cup \{\mu \exists z.(f(X) \equiv g(Y) \land c, k)\} \) \( \mu_d(C_2). \)

- **Merge.** Let \( e_1 = \exists z.(x \equiv r \land x \equiv s \land c, k), \) where \( r \not\in V, e_2 = \exists z.(x \equiv r \land r \equiv s \land c, k), C_1 = e_1 \lor d, \) and \( C_2 = e_2 \lor d. \) We have \( \beta(e_1) \geq \beta(e_2) \) and \( \gamma(e_2) \geq \gamma(e_2). \) If \( s \in V \) then \( \alpha(e_1) > \alpha(e_2), \) and otherwise \( \delta(e_1) > \delta(e_2). \) Thus we obtain \( \mu_d(C_1) >_d \mu_d(C_2). \)

- **Check.** Similar to **Fail.**

- **Eliminate.** Let \( e_1 = \exists z.(x \equiv t \land c, k), e_2 = \exists z.(\alpha \land c, k), \) where \( \alpha = \{x \mapsto t\}, C_1 = e_1 \lor d, \) and \( C_2 = e_2 \lor d. \) If \( t \in V \) then \( \alpha(e_1) > \alpha(e_2). \) Otherwise \( \alpha(e_1) \geq \alpha(e_2), \beta(e_1) \geq \beta(e_2) \) (Since \( t \not\in V, \) no variable is placed under a new symbol), \( \gamma(e_1) \geq \gamma(e_2), \) and \( \delta(e_1) > \delta(e_2), \) thus \( \mu_d(C_1) >_d \mu_d(C_2). \)

- **Free Decompose.** We prove that the application of **Free Decompose**, followed by the exhaustive application of **Delete**, **Check** and **Eliminate** on the newly introduced atomic problems, decreases the complexity of an AC-unification problem.

Let \( e_1 = \exists z.(f(X) \equiv f(Y) \land c, k), \) where \( f \in F_{\text{free}} \) and \( |X| = |Y|, \) \( e_2 = \exists z.(\land_{i \in [1..|X|]} X_i \equiv Y_i \land c, k), \) and \( e_2 = \exists z.(c' \land c \land k \land c), \) where \( c' \) and \( c \) are defined as follows:

Let \( (x_1 \equiv t_1 \land \cdots \land x_n \equiv t_n \land r_1 \equiv s_1 \land \cdots \land r_m \equiv s_m) = \land_{i \in [1..|X|]} X_i \equiv Y_i, \) where \( x_i \in V \) for all \( i \in [1..n], \) and \( r_i \) or \( s_i \not\in V \) for all \( i \in [1..m]. \)
(In the decomposition, we separate the atomic problems of which at least one side consists of a variable from the rest.)

Let $R$ be the relation $\text{Delete} \cup \text{Check} \cup \text{Eliminate}$. Then $\sigma = \Rightarrow^1_R (x_1 \overset{2}{\Leftarrow} \bar{t}_1 \land \cdots \land x_n \overset{2}{\Leftarrow} \bar{t}_n)$ and $c' = (r_1 \overset{2}{\Leftarrow} \bar{s}_1 \land \cdots \land r_m \overset{2}{\Leftarrow} \bar{s}_m)$.

Let $C_1 = e_1 \lor d$ and $C_2 = e_2 \lor d$.

We show: $\mu_e(e_1) >_e \mu_e(e_2)$.

We obtain $\alpha(e_1) \geq \alpha(e_2)$, since $\sigma$ eliminates the atomic problems with variables on both sides that are introduced by the decomposition.

We show: $\beta(e_1) \geq \beta(e_2)$.

Because $\sigma$ may substitute a variable by another variable, there may exist variables $x$ with $|\text{OP}(x, e_2)| \geq 2$ and $|\text{OP}(x, e_1)| \geq 2$. In this case, there exists $g \in F$ with $g \neq f$ such that $g \in \text{OP}(x, e_2)$ and $g \notin \text{OP}(x, e_1)$, and there exists $y \in V$ such that $(y \mapsto x) \in \sigma$ and $g \in \text{OP}(y, e_1)$. Furthermore, $f \in \text{OP}(y, e_1)$ (because $y \in X \cup Y$) and therefore $|\text{OP}(y, e_1)| \geq 2$. The variable $y$ does not occur in $e_2$, therefore $|\text{OP}(y, e_2)| \geq 2$.

In other words, whenever there is a variable $x$ such that $|\text{OP}(x, e_2)| \geq 2$ and $|\text{OP}(x, e_1)| \geq 2$, there is a corresponding variable $y$ such that $|\text{OP}(y, e_2)| \geq 2$ and $|\text{OP}(y, e_1)| \geq 2$. Thus $\beta(e_1) \geq \beta(e_2)$.

If at most one of the terms $f(X)\sigma$ and $f(Y)\sigma$ occurs in $e_2$, then $\gamma(e_1) > \gamma(e_2)$, and $\mu_e(e_1) >_e \mu_e(e_2)$.

Now, assume that both $f(X)\sigma$ and $f(Y)\sigma$ occur in $e_2$. None of the occurrences of terms in $c' \sigma$ count for $\varepsilon(e_2)$, since they are proper sub-terms of $f(X)\sigma$ or $f(Y)\sigma$. At least one of the occurrences of $f(X)$
and \( f(Y) \) counts for \( \varepsilon(e_1) \) (see condition to apply Free Eliminate).

Clearly, this occurrence has no corresponding occurrence in \( e_2 \). Therefore \( \varepsilon(e_1) > \varepsilon(e_2) \), and \( \mu_e(e_1) > \mu_e(e_2) \).

We conclude that \( \mu_d(C_1) > \mu_d(C_2) \).

- **AC Decompose.** We prove that the application of AC Decompose, followed by the exhaustive application of Delete, Check, and Eliminate on the newly introduced atomic problems, decreases the complexity of an AC-unification problem. To describe the eager application of Delete, Check, and Eliminate more precisely, we define the function \( \text{elim} \):

Let \( e = \exists \exists^*(f(X) \mapsto f(Y), \top) \), \( f \in F_{AC} \) and \( R \) be the relation Delete \( \cup \) Check \( \cup \) Eliminate. Let

\[
\exists^*_{e_1},(c_1, \sigma_1) \quad \forall \\
\vdots \quad \vdots = \Rightarrow^1_R (\bigvee_{i \in [1..\delta_{AC}(e)]} \text{dio}_{AC}(e)). \\
\exists^*_{e_n},(c_n, \sigma_n)
\]

Then \( \text{elim}(e) = \{(z_i, c_i, \sigma_i) \mid i \in [1..n]\} \).

Now, let \( C_1 = \exists \exists^*(f(X) \mapsto f(Y) \land c, k) \lor d \), where \( f \in F_{AC} \), and \( C_2 = \exists \exists^*(c \sigma_i \land c \sigma_j, \sigma_i \land k \sigma_j) \lor d \).

Let \( e_1 = \exists \exists^*(f(X) \mapsto f(Y) \land c, k) \), and \( e_2 = \exists \exists^*(c_j \sigma_j \land c_j \sigma_j \land k \sigma_j) \), where \( (z_j, c_j, \sigma_j) \in \text{elim}(e) \).

The conjunctive constraint \( c_j \) consists of proper non-variable subterms of \( f(X) \) and \( f(Y) \) (see Property 5 in Definition 3.3). The substitution \( \sigma_j \) represents the accumulated substitutions resulting from exhaustive
elimination using the atomic problems introduced by $dio_{AC}$. If $(x \mapsto t) \in \sigma_j$, then $x \in X \cup Y$ (see Property 4 in Definition 3.3).

We obtain $\alpha(e_1) \geq \alpha(e_2)$, since $\sigma_j$ eliminates the atomic problems with variables on both sides that are introduced by the decomposition, and $\beta(e_1) \geq \beta(e_2)$ (argument similar to Free Decompose).

If there exists $(x \mapsto t) \in \sigma_j$ with $t \notin V$, such that $|OP(x, e_1)| \geq 2$ then $\beta(e_1) > \beta(e_2)$, and $\mu_e(e_1) \geq \mu_e(e_2)$.

Now, we assume that $OP(x, e_1) = \{f\}$ for all $(x \mapsto t) \in \sigma_j$ where $t \notin V$. Let $e'_2 = \exists z_j. (f(X) \sigma_j \sim f(Y) \sigma_j \land e_j \sigma_j \land \alpha \sigma_j, \sigma_j \land k \sigma_j)$.

We can show by induction on the number of applications of Eliminate in the construction of $\sigma_j$ that $\gamma(e_1) \geq \gamma(e'_2)$. (Here, we use that Flatten applies, when we substitute a variable argument in a term with root $f$ by another term with root $f$. Because $OP(x, e_1) = \{f\}$ for all $(x \mapsto t) \in \sigma_j$ where $t \notin V$, every newly built term is flattened into another term.)

If at most one of the terms $f(X) \sigma_j$ and $f(Y) \sigma_j$ occurs in $e_2$, then $\gamma(e_1) > \gamma(e_2)$, and $\mu_e(e_1) > \mu_e(e_2)$.

Now, assume that both $f(X) \sigma_j$ and $f(Y) \sigma_j$ occur in $e_2$. We obtain $\delta(e_1) \geq \delta(e_2)$, since all newly introduced atomic AC1-unification problems are eliminated. None of the occurrences of terms in $c \sigma_j$ count for $\varepsilon(e_2)$, since they are proper subterms of $f(X) \sigma_j$ or $f(Y) \sigma_j$. The constraint $c \sigma$ has at most as many occurrences of terms that count for $\varepsilon(e_1)$ than has $c$ that count for $\varepsilon(e_1)$, (Here, we use that $\oplus \notin OP(x, c)$, for all $x$ with $(x \mapsto t) \in \sigma_j$ and $t \notin V$.)

Furthermore, at least one of the occurrences of $f(X)$ and $f(Y)$ in the
atomic problem $f(X) /\overset{2}{=} f(Y)$ counts for $\varepsilon(e_1)$ (see condition for application of AC Decompose). Clearly, this occurrence has no corresponding occurrence in $e_2$. Therefore $\varepsilon(e_1) > \varepsilon(e_2)$, and $\mu_\varepsilon(e_1) > \mu_\varepsilon(e_2)$.

We have shown that for any existential constraint

$$e_2 \subseteq \bigcup_{(e_{i', e_i}) \in \text{elim}(e)} \exists e_i \cdot e_{i'} \sigma_i \land e_i \sigma_i \land k \sigma_i$$

holds $\mu_\varepsilon(e_1) > \mu_\varepsilon(e_2)$.

To obtain $C_2$, the existential constraint $e_1$ is replaced in $C_1$ by a finite number of existential constraints such that $e_1$ is bigger than all of them under $>_\varepsilon$. By definition of the multiset extension, we obtain $\mu_\delta(C_1) > \mu_\delta(C_2)$.

\vspace{1em}

\section{3.3 AC1-Unification}

The equations $f(Z, 0) \approx Z$ and $f(0, Z) \approx Z$ are collapsing in the sense that one side consists of a term that occurs in the other side as a proper subterm. This makes solving of AC1-unification problems more complicated than solving AC-unification problems.

Similar to the previous section, we define a simplifier for AC1-unification problems to describe the decomposition of atomic AC1-unification problems with AC1-terms on both sides.

\textbf{Definition 3.6} A simplifier for AC1-unification problems is a partial function
\[ dio_{AC} : C_{AC1} \times 2^V \rightarrow 2^{C_{AC1}} \text{ such that for all } f \in F_{AC1} \text{ and } W \subseteq V, \]

\[
dio_{AC}(\exists \exists (f(X) \equiv f(Y), T), W) = \left\{ \begin{array}{l}
\exists \exists_1. \ (\land_{i \in [1..n_1]} r_{1,i} \equiv s_{1,i}, T), \\
\exists \exists_2. \ (\land_{i \in [1..n_2]} r_{2,i} \equiv s_{2,i}, T), \\
\vdots \\
\exists \exists_k. \ (\land_{i \in [1..n_k]} r_{k,i} \equiv s_{k,i}, T), \\
\vdots
\end{array} \right\}
\]

where

\[
Sol(\exists \exists (f(X) \equiv f(Y), T)) = \bigcup_{h \in [1..k]} Sol(\exists \exists_h. (\land_{i \in [1..n_h]} r_{i,h} \equiv s_{i,h}, T))
\]

and for all \( h \in [1..k] \), for all \( i, i' \in [1..n_h] \)

1. \( \exists \subseteq \exists_h \),
2. \( r_{h,i} \in X \cup Y \),
3. \( r_{h,i} \neq r_{h,i'} \),
4. if \( r_{h,i} \in V \) or \( \text{root}(r_{h,i}) \in F_{AC1} \) then \( s_{h,i} \in X \cup Y \), or \( s_{h,i} = f(Z) \), where \( Z \subseteq (V - W) \cup X \cup Y \),
5. if \( r_{h,i} \notin V \) and \( \text{root}(r_{h,i}) \notin F_{AC1} \) then \( r_{h,i} \) and \( s_{h,i} \in X \cup Y \).

The simplification is often much easier with than without identities (for many examples see [BHK 88]).

Note that \( AC1 \)-terms in \( X \) and \( Y \) must be handled like variables, because they may collapse and unify with any other term. Therefore, we cannot maintain the properties 4 and 5 in Definition 3.3. However, if \( F_{AC} \) contains only
one element, then no AC1-term can occur in X and Y, since we consider flattened terms. In this case, the properties 4 and 5 in Definition 3.3 are valid (see Lemma 3.6).

The set AC1-Unify of transformation rules consists of the rules Delete, Merge, Eliminate, Free Decompose, and AC Decompose in Table 1 and the rules in Table 2. The union of the relations described by the rules in AC1-Unify is denoted by $\Rightarrow_{AC1-Unify}$.

**Definition 3.7** An AC1-unification procedure is a subset $U$ of the relation $\Rightarrow_{AC1-Unify}$ such that every normal form with respect to $U$ is a normal form with respect to $\Rightarrow_{AC1-Unify}$.

Similar to the previous section, we state the soundness, completeness, and termination of AC1-unification procedures. We give only the parts of the proofs that differ from the corresponding proof in the previous section.

**Lemma 3.4 (Soundness)** Every AC1-unification procedure preserves the unifiers: If $C_1 \Rightarrow^*_{AC1-Unify} C_2$, then $Sol(C_1) = Sol(C_2)$.

**Proof.**

- **Fail.** If $f, g \notin F_{AC1}$ and $f \neq g$ then $Sol(f(X) \overline{=} g(Y)) = \emptyset$, since AC1 can only change the root symbols of terms $t$ with $\text{root}(t) \in F_{AC1}$. Thus, for $C_1 = \exists \overline{x}. (f(X) \overline{=} g(Y) \land c, k) \lor d$ and $C_2 = d$, we obtain $Sol(C_1) = Sol(C_2)$.

- **Check.** Let $C_1 = \exists \overline{x}. (x \overline{=} f(X) \land c, k) \lor d$, $f \notin F_{AC1}$, and $x \in \text{Var}(f(X))$. Assume $\sigma \in Sol(x \overline{=} f(X))$. Since $x$ occurs in $f(X)$, every subterm of $x\sigma$ must occur in $f(X)\sigma$. Since $f \notin F_{AC1}$, $\text{root}(x\sigma)$
Fail:
\[ \exists z. (f(X)\equiv g(Y) \land c, k) \land d \Rightarrow d \]
\[
\text{if } \begin{cases} 
  f, g \notin F_{AC1} \\
  f \neq g 
\end{cases}
\]

Check:
\[ \exists z. (x \equiv f(X) \land c, k) \land d \Rightarrow d 
\]
\[
\text{if } \begin{cases} 
  x \in V \\
  f \notin F_{AC1} \\
  x \in \text{Var}(f(X)) 
\end{cases}
\]

Collapse 1:
\[ \exists z. (f(X)\equiv t \land c, k) \land d \Rightarrow \]
\[
\bigvee_{i \in [1..|X|]} \exists z. (\bigwedge_{j \in [1..|X|], j \neq i} X_i \equiv \text{zero}(f) \land X_i \equiv t \land c, k) \land d 
\]
\[
\text{if } \begin{cases} 
  \text{either } t \in V \text{ and } t \in \text{Var}(f(X)) \\
  \text{or root}(t) \notin F_{AC1} \\
  f \in F_{AC1} \\
  f(X) \notin \text{subt}(c) 
\end{cases}
\]

Collapse 2:
\[ \exists z. (f(X)\equiv g(Y) \land c, k) \land d \Rightarrow \]
\[
\bigvee_{i \in [1..|X|]} \exists z. (\bigwedge_{j \in [1..|X|], j \neq i} X_i \equiv \text{zero}(f) \land X_i \equiv g(Y) \land c, k) 
\]
\[
\bigvee_{i \in [1..|Y|]} \exists z. (\bigwedge_{j \in [1..|Y|], j \neq i} \text{zero}(f) \equiv Y_j \land f(X) \equiv Y_i \land c, k) \land d 
\]
\[
\text{if } \begin{cases} 
  f, g \in F_{AC1} \\
  f \neq g 
\end{cases}
\]

AC1 Decompose:
\[ \exists z. (f(X)\equiv f(Y) \land c, k) \land d \Rightarrow \]
\[
\bigvee_{(c', \tau) \in \text{dio}_{AC1}(\exists z. (f(X)\equiv f(Y) \land \tau), W)} \exists z. (c' \land c, k) \land d 
\]
\[
\text{where } W = \text{Var}(\exists z. (f(X)\equiv f(Y) \land c, k)) 
\]
\[
\text{if } \begin{cases} 
  f \in F_{AC1} \\
  f(X) \neq AC1 \ f(Y) \\
  f(X) \notin \text{subt}(c) \text{ or } f(Y) \notin \text{subt}(c) 
\end{cases}
\]

Table 2: The transformation rules AC1-Unify
root(\(f(X)\sigma\)). Either a proper subterm of \(f(X)\sigma\) contains \(x\sigma\) or Flatten has applied and \(|\text{args}(x\sigma)| < |\text{args}(f(X)\sigma)|\). Both lead to a contradiction. (If \(f(\bar{Y}) =_{\text{AC1}} f(\bar{Z})\), then \(|\bar{Y}| = |\bar{Z}|\) and there exists a permutation \(\pi\), such that \(Y^i =_{\text{AC1}} Z^{\pi(i)}\), for all \(i \in [1..|\bar{Y}|]\). Thus \(\text{Sol}(x^2 = f(X)) = \emptyset\); and for \(C_2 = d\), we obtain \(\text{Sol}(C_1) = \text{Sol}(C_2)\).

- **Collapse 1.** Let \(C'_1 = \exists \Xi. (f(X)\Xi t \land c, k)\), where \(f \in F_{\text{AC1}}\), and \(t \in V\) or \(\text{root}(t) \notin F_{\text{AC1}}\). \(C_1 = C'_1 \lor d\), \(C'_2 = \lor_{i \in [1..|X|]} \exists \Xi (\land_{j \in [1..|X|]} j \neq i X^j = \text{zero}(f) \land X^i = t \land c, k)\), and \(C_2 = C'_2 \lor d\).

We show that \(\text{Sol}(C'_1) = \text{Sol}(C'_2)\).

\(`\subseteq\)`: If \(\sigma \in \text{Sol}(f(X)\Xi t)\) then \(f(X)\sigma =_{\text{AC1}} t\sigma\). Either \(t\) is in \(V\) or \(\text{root}(t) \notin F_{\text{AC1}}\). Zero must apply to \(f(X)\sigma\) \(|X| - 1\) times at top-level, leaving an instance of a term \(X^i\) for some \(i \in [1..|X|]\): \(X^i\sigma =_{\text{AC1}} t\).

In this case, \(X^j\sigma =_{\text{AC1}} \text{zero}(f)\) for all \(j \in [1..|X|], j \neq i\) and thus \(\sigma \in \text{Sol}(\land_{j \in [1..|X|]} j \neq i X^j = \text{zero}(f) \land X^i = t)\). Since all \(i \in [1..|X|]\) are covered in \(C'_2\), we conclude \(\text{Sol}(C'_1) \subseteq \text{Sol}(C'_2)\).

\(`\supseteq\)`: If \(\sigma \in \text{Sol}(\land_{j \in [1..|X|]} j \neq i X^j = \text{zero}(f) \land X^i = t)\), for some \(i \in [1..|X|]\), then Zero applies to \(f(X)\sigma\) \(|X| - 1\) times at top-level since \(\sigma X^i =_{\text{AC1}} \text{zero}(f)\), for all \(j \in [1..|X|], j \neq i\), leaving an instance of \(X^i\).

- **Collapse 2.** similar to **Collapse 1**, but both the left and the right hand side can collapse.

- **AC1 Decompose.** similar to **AC Decompose.**
Lemma 3.5 (Completeness) Every normal form with respect to AC1-Unify is an AC1-unification problem in solved form.

Proof. Let $C = \bigvee_{h \in [1..k]} \exists x_h^r(c_h, k_h)$ be a normal form with respect to AC1-Unify. Assume that for some $h \in [1..k]$ $c_h = r \exists s \land c_h^r$, where $r = f(X)$ and $s = g(Y)$ are non-variable terms. If both $f \in F_{AC1}$ and $g \in F_{AC1}$, then Collapse 2 applies if $f \neq g$ and AC1 Decompose or Delete applies otherwise. If only one of the symbols $f$ and $g$ are in $F_{AC1}$, then Collapse 1 applies. If $f = g \notin F_{AC1}$ then AC Decompose, Free Decompose or Delete applies, and if $f, g \notin F_{AC1}$ and $f \neq g$, then Fail applies. Note that the restrictions in Collapse 1, Free Decompose, AC Decompose, and AC1 Decompose cause no deadlock. (In a set of terms, not all terms can be strictly contained in other terms.)

So, assume that $r = x \in V$. If $s = x$, Delete applies. If $s \neq x$, then Check applies if $x \in \text{Var}(s)$ and $\text{root}(s) \notin F_{AC1}$, Collapse 1 applies if $x \in \text{Var}(s)$ and $\text{root}(s) \in F_{AC1}$, and Eliminate applies otherwise.

The validation of the solved form properties is similar to the proof of Lemma 3.2. \hfill \Box

Lemma 3.6 (Termination) Every AC1-unification procedure in which, after every application of Collapse 1, Collapse 2, Free Decompose, AC Decompose and AC1 Decompose all newly introduced atomic problems with variables on at least one side are eliminated using Delete, Check and Eliminate terminates on every AC1-unification problem that contains only one AC-symbol.
**Proof.** The limitation on the presence of only one AC-symbol enables us to use the same ordering as in the proof of Lemma 3.3 (see Definition 3.6). The proof for the rules that occur already in AC-Unify is similar to the proof of Lemma 3.3. The proof for the rule AC1 Decompose is similar to the proof for the rule AC Decompose. We discuss the remaining rules:

- **Collapse 1.** We prove that the application of **Collapse 1**, followed by the application of **Delete**, **Check**, and **Eliminate** on the newly introduced atomic problems with variables on at least one side, decreases the complexity of an AC1-unification problem.

Let \( e_1 = \exists z.(f(X) \geq t \land c, k) \), where either \( t \in V \) and \( t \in Var(f(X)) \) or \( \text{root}(t) \notin F_{AC1}, f \in F_{AC1}, \) and \( f(X) \notin subl(c) \), and let \( C_1 = e_1 \lor d \). Let \( C'_2 = \forall i \in \{1, \ldots, \mid X \mid \} \exists z. (\bigwedge_{i \in \{1, \ldots, \mid X \mid \} \mid i \neq h} X^i \geq \text{zero}(f) \land X^i \geq t \land c, k) \lor d \), and \( e'_2 = \exists z. (\bigwedge_{i \in \{1, \ldots, \mid X \mid \} \mid i \neq h} X^i \geq \text{zero}(f) \land X^h \geq t \land c, k) \), for some \( h \in [1, \mid X \mid] \).

1. If \( X^h, t \in V \) and \( X^h = t \) then **Delete** applies to \( X^h \geq t \). \( e_2 = \exists z. (\bigwedge_{i \in \{1, \ldots, \mid X \mid \} \mid i \neq h} X^i \geq \text{zero}(f) \land c, k) \), where \( \sigma = \{ X^i \mapsto \text{zero}(f) \mid X^j \in V, j \neq h \} \). We obtain \( \alpha(e_1) \geq \alpha(e_2) \) and \( \beta(e_1) \geq \beta(e_2) \). If \( f(X) \) does not occur in \( c \), then \( \gamma(e_1) > \gamma(e_2) \), and therefore \( \mu_c(e_1) > \mu_c(e_2) \).

Now, we assume that \( f(X) \) occurs in \( c \). We obtain \( \gamma(e_1) \geq \gamma(e_2) \) and \( \delta(e_1) \geq \delta(e_2) \). Since \( f(X) \sigma \) occurs in \( c \), the terms \( X^i \sigma \) do not count for \( \varepsilon(e_2) \). Furthermore, \( f(X) \) counts for \( \varepsilon(e_1) \), since \( f(X) \notin subl(c) \) (see condition to apply **Collapse 1**). Clearly, this occurrence of \( f(X) \) has no corresponding occurrence in \( e_2 \). Thus,
CHAPTER 3. UNIFICATION

\[ \varepsilon(e_1) > \varepsilon(e_2), \text{ and therefore } \mu_e(e_1) > e \mu_e(e_2). \]

2. If \( X^h, t \in V \) and \( X^h \neq t \) then **Eliminate** applies to \( X^h \overset{2}{=} t \), resulting in \( e_2 = \exists \varepsilon'(\wedge_{i \in \mathbb{N}} \exists X^i \sigma \neq \emptyset; X^i \sigma \in V \) \( X^i \sigma \overset{2}{=} \text{zero}(f) \land \alpha \sigma, k \sigma \), where \( \sigma = \{ X^j \mapsto \text{zero}(f) \mid X^j \in V; j \neq h \} \cup \{ t \mapsto X^h \} \). We obtain \( \alpha(e_1) \geq \alpha(e_2) \). The term \( t \) occurs under \( \@ \) and, since \( t \in \text{Var}(f(X)) \), under at least one other symbol. Therefore, we obtain \( \beta(e_1) \geq \beta(e_2) \). Clearly, \( \gamma(e_1) \geq \gamma(e_2) \). The atomic AC1-unification problem \( f(X) \overset{2}{=} t \) has no corresponding occurrence in \( e_2 \), thus \( \delta(e_1) > \delta(e_2) \), and therefore \( \mu_e(e_1) > e \mu_e(e_2) \).

3. If \( X^h \not\in V \) and \( t \in V \) then **Eliminate** applies to \( X^h \overset{2}{=} t \) and \( e_2 \) as in 2. Proof as in 1.

4. If \( X^h \in V \) and \( t \not\in V \) then **Eliminate** applies to \( X^h \overset{2}{=} t \). \( e_2 = \exists \varepsilon'(\wedge_{i \in \mathbb{N}} \exists X^i \sigma \neq \emptyset; X^i \sigma \in V \) \( X^i \sigma \overset{2}{=} \text{zero}(f) \land \alpha \sigma, k \sigma \), where \( \sigma = \{ X^j \mapsto \text{zero}(f) \mid X^j \in V; j \neq h \} \cup \{ X^h \mapsto t \} \). Proof as in 1.

5. If \( X^h \) and \( t \not\in V \) then \( e_2 = \exists \varepsilon'(\wedge_{i \in \mathbb{N}} \exists X^i \sigma \neq \emptyset; X^i \sigma \in V \) \( X^i \sigma \overset{2}{=} \text{zero}(f) \land X^h \sigma \overset{2}{=} \text{zero}(f) \land \alpha \sigma, k \sigma \), where \( \sigma = \{ X^j \mapsto \text{zero}(f) \mid X^j \in V; j \neq h \} \). Proof as in 1.

We proved that, after the exhaustive application of **Delete**, **Fail**, and **Eliminate**, every newly constructed existential constraint \( e_2 \) fulfills \( \mu(e_1) > e \mu(e_2) \); thus the complexity of the AC1-unification problem decreases.

- **Collapse 2.** We prove that the application of **Collapse 2**, followed by the application of **Delete**, **Check**, and **Eliminate** on the newly
introduced atomic problems with variables on at least one side, decreases
the complexity of an AC1-unification problem.

Let \( e_1 = \exists x. (f(X) \equiv g(Y) \land c, k) \), where \( f, g \in F_{AC1} \), and \( C_1 = e_1 \lor d \).

Let \( C'_2 = \)
\[ \forall i \in [1..|X|] \exists x.(\forall i \in [1..|X|] \exists x. i \neq h X^i \equiv g(Y) \land X^i \equiv g(Y) \land c, k) \lor \]
\[ \forall i \in [1..|P|] \exists x.(\forall i \in [1..|P|] \exists x. i \neq h zero(f) \equiv Y^i \land f(X) \equiv Y^i \land c, k) \lor d, \]
and \( e'_2 = \exists x.(\forall i \in [1..|X|] \exists x. i \neq h X^i \equiv g(Y) \land X^i \equiv g(Y) \land c, k) \), for some
\( h \in [1..|X|] \) (proof for \( e'_2 \) in second disjunction similar).

1. If \( X^h \in V \) then Eliminate applies to \( X^h \equiv g(Y) \), and \( e_2 = \exists x. \)
\[ (\forall i \in [1..|X|] \exists x. i \neq h X^i \equiv g(Y) \land \sigma, k \sigma), \]
where \( \sigma = \{ X^j \mapsto zero(f) \mid X^j \in V, j \neq h \} \cup \{ X^h \mapsto g(Y) \} \). We obtain \( \alpha(e_1) \geq \alpha(e_2) \). If \( |OP(X^h, e)| \geq 2 \), then \( \beta(e_1) > \beta(e_2) \). Otherwise, we
obtain \( \beta(e_1) \geq \beta(e_2) \), and since \( X^h \) occurs only under the AC-
symbol \( f \), Flatten applies at every application of \( \{ X^h \mapsto g(Y) \} \),
such that \( \gamma(e_1) > \gamma(e_2) \).

2. If \( X^h \notin V \) then \( e_2 = \exists x.(\forall i \in [1..|X|] \exists x. i \neq h X^i \equiv g(Y) \land X^h \equiv \)
\[ g(Y) \sigma \land \sigma, k \sigma), \]
where \( \sigma = \{ X^j \mapsto zero(f) \mid X^j \in V, j \neq h \} \).
For \( \Gamma \in \{ \alpha, \beta, \gamma, \delta, \epsilon \} \), we obtain \( \Gamma(e_1) \geq \Gamma(e_2) \), and additionally
\( \zeta(e_1) > \zeta(e_2) \), since all positions in \( e_2 \) have a corresponding position
in \( e_1 \) (the terms \( X^h \sigma \) have their corresponding position in \( f(X) \)),
and the position \( \langle \lambda, \emptyset \rangle \) in \( f(X) \) has no corresponding position in \( e_2 \).

We proved that, after the exhaustive application of Delete, Fail, and
Eliminate, every newly constructed existential constraint \( e_2 \) fulfills
\[ \mu(e_1) \geq_\varepsilon e_2; \text{ thus the complexity of the AC1-unification problem decreases.} \]

\[ \square \]

### 3.4 AC1-Unification with Zero-Disequations

During the solving AC1-unification problems with zero-disequations, we transfer zero-disequations of the form \( t \neq 0 \) in the unsolved part to the solved part, in which only zero-disequations of the form \( x \neq 0 \), where \( x \in V \), are allowed. We shall resolve zero-disequations similarly to atomic AC1-unification problems. In addition any substitution resulting from a solution of a unification problem must fulfill all zero-disequations.

All terms in the solved part are kept in normal form with respect to the rewrite system Zero. This ensures, together with the normal form computation with respect to Flatten, that if a term \( t =_{AC1} 0 \) for some \( 0 \in F_{\text{zero}} \) then \( t = 0 \), so that violations of zero-disequations can easily be detected.

**Notation:** We extend the function \( \downarrow_{\text{zero}} \) from terms to conjunctive constraints. Let

\[ c = \bigwedge_{i \in [1..n]} r_i \overset{2}{=} s_i \land \bigwedge_{i \in [1..n]} x_i \neq f_i \]

be a conjunctive constraint. Then

\[ c \downarrow_{\text{zero}} = \bigwedge_{i \in [1..n]} r_i \downarrow_{\text{zero}} \overset{2}{=} s_i \downarrow_{\text{zero}} \land \bigwedge_{i \in [1..n]} x_i \neq f_i. \]
The set $\text{Disequation-AC1-Unify}$ of transformation rules consists of the rules in $\text{AC1-Unify}$, where the rule $\text{Eliminate}$ is replaced by the rules $\text{Eliminate Pass}$ and $\text{Eliminate Clash}$, and the new rules $\text{Disequation Delete}$, $\text{Disequation Eliminate Pass}$, $\text{Disequation Eliminate Clash}$, and $\text{Disequation Decompose}$ in Table 3. The union of the relations described by the rules in $\text{Disequation-AC1-Unify}$ is denoted by $\Rightarrow_{\text{Disequation-AC1-Unify}}$.

**Definition 3.8** A $\text{disequation-AC1-unification procedure}$ is a subset $U$ of the relation $\Rightarrow_{\text{Disequation-AC1-Unify}}$ such that every normal form with respect to $U$ is a normal form with respect to $\Rightarrow_{\text{Disequation-AC1-Unify}}$.

Similar to the previous sections, we state the soundness, completeness, and termination of disequation-AC1-unification procedures. We give only the parts of the proofs that differ from the corresponding proof in the previous section.

**Lemma 3.7 (Soundness)** Every disequation-AC1-unification procedure preserves the unifiers: If $C_1 \Rightarrow^*_{\text{Disequation-AC1-Unify}} C_2$, then $\text{Sol}(C_1) = \text{Sol}(C_2)$.

**Proof.**

- **Eliminate Pass.** similar to $\text{Eliminate}$ in the previous section. (Clearly, the application of $\downarrow_{\text{Zero}}$ has no impact on the set of solutions of a constraint.)

- **Eliminate Clash.** Let $C_1 = \exists z. (x = t \land c, k) \lor d$. Clearly, $\text{Sol}(x = f \land x \neq f) = \emptyset$. Thus if for some atomic AC1-unification problem $x = f$ in $(x = t \land k \sigma) \downarrow_{\text{Zero}}$ the zero-disequation $x \neq f$ occurs in $k$, then $\text{Sol}(C_1) = \text{Sol}(d)$. 
Eliminate Pass:
\[ \exists z_\tau(x \neq t \land c, k) \lor d \Rightarrow \exists z_\gamma(\alpha \land g \sigma, (x \neq t \land k \sigma) \downarrow_{\text{Zero}}) \lor d \]

where \[ \begin{align*}
\sigma &= \{ x \mapsto t \} \\
k &= g \land k'
\end{align*} \]
\[ g = \bigwedge_{s \in \text{Var}(\sigma)} \bigwedge_{x \in V} s \neq 0 \\
x \notin \text{Var}(t)
\]
if \[ \begin{align*}
&\text{for no } x \neq f \text{ in } (x \neq t \land k \sigma) \downarrow_{\text{Zero}} \\
x \neq f \text{ occurs in } k
\end{align*} \]

Eliminate Clash:
\[ \exists z_\tau(x \neq t \land c, k) \lor d \Rightarrow d \]

if \[ \begin{align*}
x &\in V \\
x &\notin \text{Var}(t)
\end{align*} \]

for some \[ x \neq f \text{ in } (x \neq t \land k \sigma) \downarrow_{\text{Zero}} \\
x \neq f \text{ occurs in } k
\]

where \[ \sigma = \{ x \mapsto t \} \]

Disequation Delete:
\[ \exists z_\tau(f(X) \neq 0 \land c, k) \lor d \Rightarrow \exists z_\gamma(c, k) \lor d \]

if \[ f \notin E_{AC1} \]

Disequation Eliminate Pass:
\[ \exists z_\tau(x \neq 0 \land c, k) \lor d \Rightarrow \exists z_\gamma(c, x \neq 0 \land k) \lor d \]

if \[ \begin{align*}
x &\in V \\
x &\neq 0 \notin k
\end{align*} \]

Disequation Eliminate Clash:
\[ \exists z_\tau(x \neq 0 \land c, k) \lor d \Rightarrow d \]

if \[ \begin{align*}
x &\in V \\
x &\neq 0 \in k
\end{align*} \]

Disequation Decompose:
\[ \exists z_\tau(f(X) \neq 0 \land c, k) \lor d \Rightarrow \bigvee_{g \in DNF} \exists z_\gamma(g \land c, k) \lor d \]

where \( DNF \) is the set of conjunctions in the disjunctive normal form of
\[ (X^1 \neq 0 \lor X^2 \neq \text{zero}(f) \lor X^3 \neq \text{zero}(f) \lor \cdots \lor X^{1|X|} \neq \text{zero}(f)) \land \\
(X^1 \neq \text{zero}(f) \lor X^2 \neq 0 \lor X^3 \neq \text{zero}(f) \lor \cdots \lor X^{1|X|} \neq \text{zero}(f)) \land \\
\vdots \\
(X^1 \neq \text{zero}(f) \lor X^2 \neq \text{zero}(f) \lor X^3 \neq \text{zero}(f) \lor \cdots \lor X^{1|X|} \neq 0) \]

Table 3: The transformation rules Disequation-AC1-Unify
**Disequation Delete.** Let $C_1 = \exists \exists^*(f(X) \neq 0 \land c, k) \lor d$, where $f \notin F_{AC1}$. AC1 can only change the root symbols of terms with root$(t) \in F_{AC1}$. Thus any substitution is in Sol$(f(X) \neq 0)$ and Sol$(C_1) = Sol(\exists \exists^*(c, k) \lor d)$.

**Disequation Eliminate Pass.** trivial.

**Disequation Eliminate Clash.** similar to Eliminate Clash.

**Disequation Decompose.** Let $C_1 = \exists \exists^*(f(X) \neq 0 \land c, k) \lor d$, where $f \in F_{AC1}$. The zero-disequation $f(X) \neq 0$ is violated, if and only if for one term $X^i$ in $X$ holds $X^i =_{AC1} 0$ and for all other terms $X^j$ in $X$ holds $X^j =_{AC1} zero(f)$. The negation of this fact is expressed by the formula in Disequation Decompose, of which DNF is a disjunctive normal form. Distributivity yields that Sol$(C_1) = Sol(\forall \forall_{\exists \exists^* (g \land c, k) \lor d})$.

\[\square\]

**Lemma 3.8 (Completeness)** Every normal form with respect to Disequation-AC1-Unify is an AC1-unification problem with zero-disequations in solved form.

**Proof.** Let $C = \forall \forall_{i \in [1..k]} \exists \exists^*(c_h, k_h)$ be a normal form with respect to Disequation-AC1-Unify. From the argumentation in the proof of Lemma 3.5 follows that $c_h$ is a conjunction of zero-disequations for all $h \in [1..k]$. Assume $t \neq 0 \in c_h$. If $t \in V$ then Disequation Eliminate Pass or Disequation Eliminate Clash applies. If $t = f(X)$ then Disequation Decompose or Disequation Delete applies. Therefore, $c_h$ must be $\top$ for all $h \in [1..k]$. 
The proof of uniqueness and idempotency follows from the argumentation in the proof of Lemma 3.2.

We show that in every existential constraint in $C$, every replacement in the solved part fulfills all zero-disequations in the solved part.

Assume that the atomic $AC1$-unification problem $x \neq 0$ in $k_h$ violates the zero-disequation $x \neq 0$ in $k_h$. The atomic $AC1$-unification problem $x \neq 0$ cannot have been added to $k_h$ after the zero-disequation $x \neq 0$, since the conditions to apply Eliminate Pass would not have been fulfilled. On the other hand, the zero-disequation $x \neq 0$ cannot have been added to $k_h$ after the atomic $AC1$-unification problem $x \neq 0$, since the conditions to apply Disequation Eliminate Pass would not have been fulfilled, which is a contradiction. □

The simplest way to ensure the termination of disequation-$AC1$-unification procedures is to solve first the atomic unification problems and then the zero-disequations.

**Lemma 3.9 (Termination)** Every disequation-$AC1$-unification procedure that first applies the rules in $AC1$-Unify (including Eliminate Pass and Eliminate Clash) as described in Lemma 3.6 and then the rules Disequation Delete, Disequation Eliminate Pass, Disequation Eliminate Clash, and Disequation Decompose in any order, terminates on every $AC1$-unification problem with zero-disequations that contains only one $AC$-symbol.

**Proof.** The former process terminates due to Lemma 3.6. The termination of the latter process can be shown using an ordering based on the size of the terms in the zero-disequations of the unsolved parts of the existential constraints. □
CHAPTER 3. UNIFICATION

Note that violations of zero-disequations can be detected earlier, when the solving of atomic AC1-unification problems and the solving of zero-disequations are interleaved. However, in this case the termination proof is harder because in Eliminate Pass zero-disequations in the solved part are moved back to the unsolved part.
Chapter 4

Completion

In this chapter, we shall approach the task of completion of equation sets in the presence of associative commutative function symbols and identities. Prior to describing procedures for completion in Section 4.4, we shall present a notion of rewriting in AC1-theories in Section 4.1 that is reasonably efficient and—for our purpose—equivalent to the concept of rewriting modulo AC1. In Section 4.2, we introduce some operations on constraints that we shall use in Section 4.3 to interleave constraint solving with the process of completion. In Section 4.4, a variety of completion procedures is described using a set of transformation rules. Suitable application strategies for these rules are discussed in Section 4.5.

4.1 Constrained AC1-rewriting

Definition 4.1 A constrained equation is a triple $C \mid l \approx r$, where $C$ is a constraint, and $l$ and $r$ are terms.
CHAPTER 4. COMPLETION

If $E$ is a set of constrained equations, we write $s \rightarrow_E t$ to indicate that there exist terms $w$ and $l'$, a position $\pi$ in $w$, a substitution $\sigma$, and a constrained equation $C \mid l \approx r$ in $E$, such that $\sigma \in \text{Sol}(C)$, $s = w[l']_{\pi}$, $l' =_{AC1} l\sigma$, and $t = w[r\sigma]_{\pi}$. The relation $\rightarrow_E$ is called the rewrite relation induced by $E$. We say that $s$ rewrites to $t$ by $E$ if $s \rightarrow_E t$. A normal form with respect to $\rightarrow_E$ is said to be irreducible by $E$.

A set $E$ of equations will be called a constrained rewrite system if the rewrite relation $\rightarrow_E$ is the primary object of study. The constrained equations of a constrained rewrite system are also called constrained rewrite rules.

Several different ways have been devised to integrate equational theories into the rewriting process. The most general approach, applied to the $AC1$-case, is rewriting modulo $AC1$.

**Definition 4.2** We say that the constrained rewrite system $R$ rewrites the term $v$ to the term $w$ modulo $AC1$, denoted $v \xrightarrow{\sigma} _{R/AC1} w$, if there exist terms $v', w'$ such that $v =_{AC1} v' \rightarrow_R w' =_{AC1} w$.

We write $v \xrightarrow{\sigma} _{\mathcal{E}/AC1} w$ to specify the constrained rewrite rule and the substitution by which $v'$ is rewritten to $w'$.

**Definition 4.3** The constrained rewrite system $R$ is called convergent modulo $AC1$ if $\xrightarrow{R/AC1}$ is terminating and Church-Rosser modulo $AC1$.

The task of completion modulo $AC1$ of a given set $E$ of constrained equations is to construct a constrained rewrite system $R$ such that $AC1 \cup E$ and $AC1 \cup R$ define the same equational theory and $R$ is convergent modulo $AC1$.

Several weaker notions of rewriting in an equational theory are in use (for the general case, see [Bac 91]; for the $AC1$-case, see [JM 90]). We shall employ
our extended concept of subterms and positions to define yet another version of rewriting in the theory AC1.

**Definition 4.4** We say that the constrained rewrite system $R$ **AC1-rewrites** the term $v$ to the term $w$, denoted $v \xrightarrow{R_{AC1}} w$, if $v \downarrow_{\text{Flatten} \cup \text{Zero}} \Rightarrow R w$.

We write $v \xrightarrow{\sigma}_{AC1 \xi} w$ to specify the constrained rewrite rule and the substitution by which $v \downarrow_{\text{Flatten} \cup \text{Zero}}$ is rewritten to $w$.

**Proposition 4.1** If $r \equiv_{AC1} s \xrightarrow{R_{AC1}} t$ then there exists a term $u$ such that $r \xrightarrow{AC1 \setminus R} u \equiv_{AC1} t$.

**Proof.** Let $r \equiv_{AC1} s \xrightarrow{R_{AC1}} t$. The definition of $R_{AC1}$ implies that there exist terms $s'$ and $t'$ such that $s \equiv_{AC1} s' \Rightarrow_R t' \equiv_{AC1} t$. Let $r'$ be the normal form of $r$ with respect to Flatten $\cup$ Zero. Let $\pi$ be the position at which $\Rightarrow_R$ applies to $s'$. There exists a term $w$, a substitution $\sigma$, and a constrained rewrite rule $\mathcal{E} = C \mid (l \Leftrightarrow r)$ in $R$, such that $\sigma \in \text{Sol}(C)$, $s' = w[l\sigma]_{\pi}$ and $t' = w[r\sigma]_{\pi}$. From Lemma 2.1 follows that there exists a position $\pi'$ in $r'$ such that $r'|_{\pi'} \equiv_{AC1} s'|_{\pi}$. Therefore, the constrained rewrite rule $\mathcal{E}$ applies to $r'$ at position $\pi'$ resulting in a term $u$. Since $r' \equiv_{AC1} s'$ and $r'|_{\pi'} \equiv_{AC1} s'|_{\pi}$, we get $u \equiv_{AC1} t'$ and thus $u \equiv_{AC1} t$. Figure 1 depicts the situation. \hfill \Box

We can say that AC1-rewriting rewrites equivalence classes with respect to AC1 and is equivalent to rewriting modulo AC1, when we are only interested in one representantitive of the equivalence class of terms to which a term is rewritten. In particular, we obtain the following results.

**Proposition 4.2** (1) The relation $\xrightarrow{AC1 \setminus R}$ terminates if and only if $R_{AC1}$ terminates; (2) the relation $\xrightarrow{AC1 \setminus R}$ is Church-Rosser modulo AC1 if and only if $R_{AC1}$ is Church-Rosser modulo AC1.
**Figure 1:** Illustration to Proposition 4.1

**Proof.** (sketch) Using Proposition 4.1, we can show that if \( \overrightarrow{R/AC1} \) does not terminate then neither does \( \overrightarrow{AC1\backslash R} \) (see Figure 2). The reverse direction is trivial.

We can show by induction using Proposition 4.1 that if \( \overrightarrow{R/AC1} \) is Church-Rosser modulo \( AC1 \) then so is \( \overrightarrow{AC1\backslash R} \) (see Figure 3). The reverse direction is trivial. \( \Box \)

Now, we can rephrase a well-known fact (see [Huet 80]) using \( AC1 \)-rewriting instead of rewriting modulo \( AC1 \) in a lemma that shall be the foundation of our completion procedures.

**Lemma 4.1** Let \( R \) be constrained rewrite system such that \( \overrightarrow{AC1\backslash R} \) terminates. The relation \( \overrightarrow{R/AC1} \) is Church-Rosser modulo \( AC1 \) if and only if \( \overrightarrow{AC1\backslash R} \) is locally confluent modulo \( AC1 \).

**Remark:** \( AC1 \)-rewriting will be reasonably efficient, when all terms, including the left and right hand side of constrained rewrite rules are kept in normal
Figure 2: Illustration to Proposition 4.2 (Termination)

Figure 3: Illustration to Proposition 4.2 (Church-Rosser)
form with respect to Flatten∪Zero. In this case, AC1-matching procedures as described in Appendix A can be applied.

During the process of completion, local confluence modulo AC1 is tested whenever a new constrained rewrite rule is found. On failure of this test, “critical pairs” are added. In rewriting modulo an equational theory, “extensions” of rewrite rules are often used to test for local confluence. The concepts of critical pairs and extensions are captured by the following lemma that implies a criterion for local confluence modulo AC1.

**Lemma 4.2 (Critical Pair Lemma)** Let $E_1 = C_1 \mid l \approx r$ and $E_2 = C_2 \mid g \approx d$ be constrained rewrite rules that have no variables in common. There exist terms $t$, $v$ and $w$ such that $v \xrightarrow{AC_1 \setminus E_1} t \xrightarrow{AC_1 \setminus E_2} w$ if and only if

- there exist terms $v'$ and $w'$ such that $v \xrightarrow{AC_1 \setminus E_2} v' =_{AC_1} w' \xrightarrow{AC_1 \setminus E_1} w$, or
- there exists a subterm $l'$ in $l$ and a substitution $\sigma \in \text{Sol}(C_1) \cap \text{Sol}(C_2)$ such that $l' \sigma =_{AC_1} g \sigma$, or
- there exists a subterm $g'$ in $g$ and a substitution $\sigma \in \text{Sol}(C_1) \cap \text{Sol}(C_2)$ such that $g' \sigma =_{AC_1} l \sigma$, or
- there exists a subterm $l''$ in $l+x$ and a substitution $\sigma \in \text{Sol}(C_1) \cap \text{Sol}(C_2)$ such that $l'' \sigma =_{AC_1} g \sigma$, where $\text{root}(l) = + \in F_{AC}$ and $x$ is a new variable, or
- there exists a subterm $g''$ in $g+x$ and a substitution $\sigma \in \text{Sol}(C_1) \cap \text{Sol}(C_2)$ such that $g'' \sigma =_{AC_1} l \sigma$, where $\text{root}(g) = + \in F_{AC}$ and $x$ is a new variable.
4.2 Some Operations on Constraints

Let $C_1 = V_{i \in [1..n]} \exists z_i^e . (c_i, k_i)$ and $C_2 = V_{j \in [1..m]} \exists z_j^f . (c_j^f, k_j^f)$ be disjunctive constraints and let $DNF = V_{i \in [1..n]} \exists z_i^e . g_i$ be a disjunctive normal form of the formula $V_{i \in [1..n]} \exists z_i^e . c_i \land k_i \land V_{j \in [1..m]} \exists z_j^f . c_j^f \land k_j^f$. Then $C_1 \land C_2$ denotes the disjunctive constraint $V_{i \in [1..n]} \exists z_i^e . (g_i, \top)$.

**Proposition 4.3** For any two constraints $C_1$ and $C_2$ holds $Sol(C_1 \land C_2) = Sol(C_1) \cap Sol(C_2)$.

The application of a substitution $\sigma$ to a constraint $C = V_{i \in [1..n]} \exists z_i^e . (c_i, k_i)$ is defined to be the constraint $V_{i \in [1..n]} (c_i \sigma \land k_i \sigma, \top)$.

**Proposition 4.4** If $Ran(\sigma) \cap Var(C) = \emptyset$, $Dom(\sigma) \cap \bigcup_{i \in [1..n]} z_i = \emptyset$ and $\sigma \in Sol(C)$, then $Sol(C \sigma) = \{ \tau \sigma \mid \tau \in Sol(C) \}$.

Let $C = V_{i \in [1..n]}(\wedge_{j \in [1..m]} l_{i,j} \neq 0_{i,j} \land \wedge_{j' \in [1..m]} l'_{i,j'} \neq 0'_{i,j'})$ be a disjunctive constraint that contains no atomic unification problem and let $DNF = V_{i \in [1..\omega]} \wedge_{j \in [1..\omega]} s_{i,j}^2 \neq 1_{i,j} \land \wedge_{j' \in [1..\omega]} s'_{i,j'}^2 \neq 1'_{i,j'}$ be a disjunctive normal form of the formula $V_{i \in [1..n]}(\wedge_{j \in [1..m]} l_{i,j} \neq 0_{i,j} \lor \wedge_{j' \in [1..m]} l'_{i,j'} \neq 0'_{i,j'})$. Then the negation of $C$, denoted $\overline{C}$ is defined to be the constraint $V_{i \in [1..\omega]}(\wedge_{j \in [1..\omega]} s_{i,j}^2 \neq 1_{i,j} \land \wedge_{j' \in [1..\omega]} s'_{i,j'}^2 \neq 1'_{i,j'})$.

**Proposition 4.5** If $C$ is a disjunctive constraint that contains no atomic unification problem, then $\Sigma(V(C)) = Sol(C) \cup Sol(\overline{C})$.

4.3 Auxiliary Transformations

AC1-completion can be viewed as a transformation process on pairs consisting of a set of unoriented constrained equations on the left hand side and an
CHAPTER 4. COMPLETION

constrained rewrite system on the right hand side.

In order to interleave constraint solving with completion, we introduce the set $\text{Aux}$ of transformation rules in Table 4. Each transformation rule in $\text{Aux}$ represents a binary relation on pairs of sets of constrained equations. We denote the union of these relations by $\Rightarrow_{\text{Aux}}$.

**Proposition 4.6 (Soundness)** If $E; N \Rightarrow_{\text{Aux}} E'; N'$ then the relations $\overset{\Rightarrow_{\text{Aux}}}{\Rightarrow_{\text{AC1}}} E; N$ and $\overset{\Rightarrow_{\text{Aux}}}{\Rightarrow_{\text{AC1}}} E'; N'$ are equal.

Note that in **Equation Distribute** and **Rule Distribute** the constraint $C$ may be $\bot$. In this case, the rule or equation is discarded.

4.4 The Transformation Rules

**AC1-Complete**

Table 5 shows the set $\text{AC1 - Complete}$ of transformation rules, whose goal is to generate from a given set of constrained equations a constrained rewrite system that is convergent modulo $\text{AC1}$. Each transformation rule describes a relation on pairs of sets of constrained equations. The union of these relations is denoted by $\Rightarrow_{\text{AC1-Complete}}$. 
**Equation Unify:**

\[ E \cup \{ C \mid I \approx r \}; N \Rightarrow E \cup \{ C' \mid I \approx r \}; N \]

if \( C \Rightarrow \text{Disequation–AC1–Unify} \ C' \)

**Rule Unify:**

\[ E ; N \cup \{ C \mid I \approx r \} \Rightarrow E ; N \cup \{ C' \mid I \approx r \} \]

if \( C \Rightarrow \text{Disequation–AC1–Unify} \ C' \)

**Equation Distribute:**

\[ E \cup \{ \forall h \in [1..l] e_h \mid I \approx r \}; N \Rightarrow E \cup \bigcup_{h \in [1..l]} \{ e_h \mid I \approx r \}; N \]

**Rule Distribute:**

\[ E ; N \cup \{ \forall h \in [1..l] e_h \mid I \approx r \} \Rightarrow E ; N \cup \bigcup_{h \in [1..l]} \{ e_h \mid I \approx r \} \]

**Equation Apply:**

\[ E \cup \exists \sigma (T, \forall i \in [1..n] x_i \neq 0_i \land \forall j \in [1..m] y_j \not\equiv l_j) \mid I \approx r \}; N \]

\[ \Rightarrow E \cup \{ \forall i \in [1..n] x_i \neq 0_i \mid l \sigma \approx r \sigma \}; N \]

where \( \sigma = \bigwedge_{j \in [1..m]} y_j \not\equiv l_j \)

if \( \exists \sigma (T, \forall i \in [1..n] x_i \neq 0_i \land \forall j \in [1..m] y_j \not\equiv l_j) \) in solved form

**Rule Apply:**

\[ E ; N \cup \exists \sigma (T, \forall i \in [1..n] x_i \neq 0_i \land \forall j \in [1..m] y_j \not\equiv l_j) \mid I \approx r \}

\[ \Rightarrow E ; N \cup \{ \forall i \in [1..n] x_i \neq 0_i \mid l \sigma \approx r \sigma \}

where \( \sigma = \bigwedge_{j \in [1..m]} y_j \not\equiv l_j \)

if \( \exists \sigma (T, \forall i \in [1..n] x_i \neq 0_i \land \forall j \in [1..m] y_j \not\equiv l_j) \) in solved form

---

**Table 4:** The transformation rules **Aux**
Remarks:

- We assume given a predicate $T$ such that if $T \mathcal{E}$ for all rules $\mathcal{E}$ in $R$ then $\xrightarrow{T \mathcal{E} R}$ terminates (see Section 4.5 for further discussion of termination).

- The symbol $\simeq$ is used to denote unoriented equations in a commutative way: $E \cup \{ C \mid l \simeq r \}$ denotes that either $C \mid l \approx r$ or $C \mid r \approx l$ occurs in a set of constrained equations and the remainder of this set is denoted by $E$.

- The symbol $\triangleright$ in Collapse denotes the strict part of the encompassment ordering, which is defined as follows: $s \triangleright t$ if some subterm of $s$ is an instance of $t$, but not vice versa.

Definition 4.5 An $AC_1$-completion procedure is a subset of the relation $\text{Aux} \cup AC_1 \rightarrow \text{Complete}$.

Lemma 4.3 (Soundness) If $E; N \Rightarrow_{\text{AC}_1 \text{-Complete}} E'; N'$ then the relations $\xrightarrow{\text{AC}_1 \cup E \cup N}$ * and $\xrightarrow{\text{AC}_1 \cup E' \cup N'}$ * are equal.

We describe the fairness of a derivation in $\text{Aux} \cup AC_1 \rightarrow \text{Complete}$. Let $(CP(N); N) = \Rightarrow_{\text{Deduce}}^1 (\emptyset; N)$ be the pair of sets of constrained equations resulting from exhaustively applying the rule Deduce to the constrained rewrite system $N$. $CP(N)$ is called the set of critical equations of $N$. Let $(EXT(N); N) = \Rightarrow_{\text{Deduce Extended}}^1 (\emptyset; N)$ be the pair of sets of constrained equations resulting from exhaustively applying the rule Deduce Extend to
CHAPTER 4. COMPLETION

\[
E \triangleq N \quad \Rightarrow \quad E \cup \{C_1 \land C_2 \land l \xrightarrow{\varphi} g \mid l[d]_\varphi \approx r\} ; \ N
\]
\[
\text{if } \begin{cases} C_1 \mid l \approx r \text{ and } C_2 \mid g \approx d \in N \\ \pi \in Fpos(l) \end{cases}
\]

Deduce Extended:

\[
E \triangledown N \quad \Rightarrow \quad E \cup \{C_1 \land C_2 \land (l + x) \xrightarrow{\varphi} g \mid (l + x)[d]_\varphi \approx r\} ; \ N
\]
\[
\text{if } \begin{cases} C_1 \mid l \approx r \text{ and } C_2 \mid g \approx d \in N \\ \pi \in Fpos(l + x) \end{cases}
\]

Orient:

\[
E \cup \{C \mid l \approx r\} ; \ N \quad \Rightarrow \quad E \triangledown N \cup \{C \mid l \approx r\}
\]
\[
\text{if } T(C \mid l \approx r)
\]

Simplify:

\[
E \cup \{C_1 \mid l \approx r\} ; \ N \quad \Rightarrow \quad E \cup \{C_1 \land C_2 \theta \mid l' \approx r\} \cup \{C_1 \land \overline{C_2} \theta \mid l \approx r\} ; \ N
\]
\[
\text{if } \begin{cases} C_2 \mid g \approx d \in N \\ l \xrightarrow{\theta} \overline{AC1} \land \overline{C_2} \mid g \approx d \end{cases}
\]

Delete:

\[
E \cup \{C \mid l \approx r\} ; \ N \quad \Rightarrow \quad E \triangledown N
\]
\[
\text{if } l \triangledown \overline{AC1} r
\]

Compose:

\[
E \triangledown N \cup \{C_1 \mid l \approx r\} \quad \Rightarrow \quad E \triangledown N \cup \{C_1 \land C_2 \theta \mid l \approx r'\} \cup \{C_1 \land \overline{C_2} \theta \mid l \approx r\}
\]
\[
\text{if } \begin{cases} C_2 \mid g \approx d \in N \\ r \xrightarrow{\theta} \overline{AC1} \land \overline{C_2} \mid g \approx d \end{cases}
\]

Collapse:

\[
E \triangledown N \cup \{C_1 \mid l \approx r\} \quad \Rightarrow \quad E \cup \{C_1 \land C_2 \theta \mid l' \approx r\} \cup \{C_1 \land \overline{C_2} \theta \mid l \approx r\}
\]
\[
\text{if } \begin{cases} C_2 \mid g \approx d \in N \\ l \xrightarrow{\theta} \overline{AC1} \land \overline{C_2} \mid g \approx d \end{cases}
\]

Table 5: The transformation rules AC1-Complete
the constrained rewrite system $N$. $CP(N)$ is called the set of extensions of $N$. Let $E_0; N_0 \Rightarrow_{\text{Aux} \cup \text{AC1-Complete}} E_1; N_1 \Rightarrow_{\text{Aux} \cup \text{AC1-Complete}} \cdots$ be a derivation in $\text{Aux} \cup \text{AC1-Complete}$. The set of constrained equations $\bigcup_{i=0}^{\infty} (\bigcap_{i=n}^{\infty} N_i)$ is denoted by $N_\infty$.

A derivation $E_0; N_0 \Rightarrow_{\text{Aux} \cup \text{AC1-Complete}} E_1; N_1 \Rightarrow_{\text{Aux} \cup \text{AC1-Complete}} \cdots$ is said to be fair, if

$$\bigcup_{\mathcal{E} \in CP(N_\infty) \cup EXT(N_\infty)} (\frac{r}{AC1\mathcal{E}}) \subseteq \bigcup_{i=0}^{\infty} (\frac{r}{AC1\mathcal{E}_i}).$$

In a fair derivation $E_0; N_0 \Rightarrow_{\text{Aux} \cup \text{AC1-Complete}} E_1; N_1 \Rightarrow_{\text{Aux} \cup \text{AC1-Complete}} \cdots$ all critical equations and critical extensions of $N_\infty$ are instances of equations in the union of all $E_i$ in the sense that whenever $r \xrightarrow{AC1\mathcal{E}} s$ for two terms $r$ and $s$ and a constrained equation $\mathcal{E} \in CP(N_\infty) \cup EXT(N_\infty)$ there exists an $i \in \mathbb{N}$ and a constrained equation $\mathcal{E}' \in E_i$ such that $r \xrightarrow{AC1\mathcal{E}'} s$. An $\text{AC1}$-completion procedure is fair, if all derivations it produces are fair.

**Lemma 4.4 (Completeness)** Let the derivation $E_0; N_0 \Rightarrow_{\text{Aux} \cup \text{AC1-Complete}} E_1; N_1 \Rightarrow_{\text{Aux} \cup \text{AC1-Complete}} \cdots \Rightarrow_{\text{Aux} \cup \text{AC1-Complete}} \emptyset; N_k$ be fair. For all terms $v$ and $w$ holds: If $v \xrightarrow{AC1\mathcal{E}_0 \cup N_0} w$ then there exist terms $u$ and $u'$ such that $v \xrightarrow{AC1\mathcal{E}_k} u \xrightarrow{AC1\mathcal{E}_k} u'$ such that $u = AC1 \ x \ x' = AC1 \ y \ x' \ x''$.

**Proof.** (sketch) As in [JM 90], the completeness can be shown using proof orderings, a technique developed in [BD 89]. A rewrite system on proofs, based on the transformation rules in $\text{AC1-Complete}$, must be defined and the correctness and termination of this rewrite system must be established. \[\square\]
CHAPTER 4. COMPLETION

Table 6: The transformation rule **Equation Split**

4.5 Application Strategies

The transformation rules **AC1-Complete** describe a large class of procedures, not all of which are suitable for completion. An obvious limitation is imposed by the negation of constraints in the rules **Simplify**, **Compose** and **Collapse**. Since negation of constraints is only defined on constraints that contain no atomic unification problems, we require that the rules in $N$ are always in normal form with respect to **Aux**.

The difficulties in applying the rule **Orient** are to find an appropriate predicate $T$ and to transform a given constrained equation $C \mid l \approx r$ such that $T$ is fulfilled. Both problems are solved in [JM 90], where a criterion for termination of constrained rewrite systems is developed and an algorithm to generate a constraint for a constrained equation such that the criterion is fulfilled is presented.

Let $\mathcal{E} = C \mid l \approx r$ be an AC1-equation in $E$ and $C'$ a constraint such that $T(C \land C' \mid l \approx r)$. We can split $\mathcal{E}$ into two parts: $\mathcal{E}' = C \land C' \mid l \approx r$ and $\mathcal{E}'' = C \land \overline{C'} \mid l \approx r$. Now, we can apply **Orient** to $\mathcal{E}'$. $\mathcal{E}''$ remains in $E$. We can schematize this operation by the transformation rule **Equation Split** shown in Table 6, whose soundness follows from Propositions 4.3 and 4.5. Regarding the application of the transformation rules in **Aux** to the unoriented equations, we shall discuss two strategies.


The first strategy is to apply Aux eagerly, transforming all occurring constraints into solved constraints consisting of conjunctions of zero-disequations. This leads to a total separation of completion and unification as in [JM 90], but maintaining the possibility of constraining unoriented equations.

The second strategy consists in postponing the solving of particularly hard unification problems and “shelving” the unoriented equations in which they occur. Proceeding with the completion process, the rule Simplify may apply to the postponed unification problem and make it easier to solve. In applying the rule Orient, we may solve unification problems in unoriented equations ( lazily) only when no other equation can be oriented.

**Example 4.1** For the unification problem

$$(x + 1) \ast (y + 1) \ast (z + 1) \trianglerightequal u \ast v \ast w,$$

216 solutions are computed. In the presence of the rewrite rule $x + 1 \trianglerightequal f(x)$, we can simplify the problem to

$$f(x) \ast f(y) \ast f(z) \trianglerightequal u \ast v \ast w,$$

for which only 27 solutions are computed.

We can show the termination of lazy unification under certain conditions, imposed by the complexity measure in Definition 3.5, on the form of the rules with which the unification problems are simplified.
Chapter 5

Implementation

In this chapter, we present an overview of the experimental environment for Term Rewriting in Associative Commutative theories with Identities, TRACONE, which is currently being implemented in the programming language Standard ML (see [HMM 86]) in Version 75 of New Jersey.

The overall structure of TRACONE is outlined in Figure 4. Each module represents an SML-functor that can be separately compiled, allowing for an easy testing and recombination of the modules and a fast loading of the system. We shall briefly discuss the main features of the modules.

- A variety of modules is designed to provide basic abstract data types and functions. The module parser offers the facility to read data from a file and lexically analyze it. The abstract data structure of a finite binary relation and specializations of it are provided by the modules relation, function, and ordering. Various versions of the abstract data types of sets and multisets are provided, allowing the user to specify the equality of elements and the ordering in which the elements are stored.
Figure 4: The structure of TRACONE
The module *signature* is designed to interpret a user given signature, providing for multiple types, declaration of infix notation, associativity and commutativity of function symbols, and identity symbols.

In *variable*, multi-typed variables are specified, including a mechanism to make data structures variable distinct, and a utility to generate new variables.

The module *term* provides various functions operating on terms. The implementation of subterms and positions of terms is consistent with Section 2.1. Equality on terms is implemented as equality modulo $AC1$ and after every operation on terms the result is normalized with respect to $Flatten \cup Zero$.

The module *substitution* provides the application of substitutions to variables, terms and substitutions.

The notion of constraints in *constraint* is consistent with Section 2.4. Full and partial solving of constraints is supported, implementing the transformation rules in Section 3.4. For the basic simplification steps in the $AC$- and $AC1$-case,

the module *simplifier* is used. It represents an implementation of the function $dio$ in [Fag 84], using

the solving of linear diophantine equations, implemented in *diophant* and based on [Huet 78].
CHAPTER 5. IMPLEMENTATION

- The module matching provides constrained AC1-matching and represents an optimized implementation of the transformation rules given in Appendix A.

- The modules rules and trs offer mechanisms to build AC1-rewrite systems and compute normal forms, whereas

- the modules equation and equationset provide methods to build sets of unoriented equations.

- In the module termorderings, various versions of recursive path orderings ([Der 82]) are integrated into the framework of AC-path orderings (see [BP 85]). The user can specify a precedence ordering and the status of free function symbols (lexicographic left, lexicographic right, and multiset). AC-symbols have the status multiset.

In Appendix B, the use of TRACONE is demonstrated on a variety of examples.
Bibliography


Appendix A

AC1-matching

A term $s$ AC1-matches a term $t$ via the substitution $\sigma$, if $s =_{AC1} t\sigma$. Just like AC1-unification, AC1-matching can be described with transformation rules. In Table 7, we give the set AC1-Match of transformation rules. Note that we require both terms to be in normal form with respect to Flatten∪Zero. AC1-matching procedures can be defined similar to AC1-unification procedures (see Definition 3.7) and can be proved to be sound and complete similar to Lemmas 3.4 and 3.5. All AC1-matching procedures terminate as can be shown using an ordering based on (the multiset extension of the set extension of) a simple ordering on terms.

Remarks:

- We employ the notation from section 3.2 but can do without existential quantification. The symbol $\models_{\mathcal{M}}$ is not commutative.

- In the rules AC Decompose and AC1 Decompose, the following notations are used: $\hat{f}(X)$ denotes zero($f$) if $|X| = 0$, $X^1$ if $|X| = 1$,
Delete:
\[(r_{=^2_{Ms}} \land c, k) \lor d \Rightarrow (c, k) \lor d\]
if \(r =_{AC1} s\) and \(r\) ground

Fail:
\[(f(X)_{=^2_{Mg}}(Y) \land c, k) \lor d \Rightarrow d\]
if \(f \notin F_{AC1}\) and \(f \neq g\)

Eliminate:
\[(x_{=^2_{Mt}} \land c, k) \lor d \Rightarrow (c, x_{=^2_{Mt}} \land k) \lor d\]
if \(\begin{cases} x \in V \\ \text{no } x_{=^2_{Mt'}} \text{ with } t' \neq_{AC1} t \text{ in } k \end{cases}\)

Clash:
\[(x_{=^2_{Mt}} \land c, k) \lor d \Rightarrow d\]
if \(\begin{cases} x \in V \\ \text{some } x_{=^2_{Mt'}} \text{ with } t' \neq_{AC1} t \text{ in } k \end{cases}\)

Free Decompose:
\[(f(X)_{=^2_{Mf}}(Y) \land c, k) \lor d \Rightarrow (\land_{i \in \{1..|X|\}} X^i_{=^2_{Mf}}(p(i)) \land c, k) \lor d\]
if \(f \in F_{free}\)

AC Decompose:
\[(f(X)_{=^2_{Mf}}(Y) \land c, k) \lor d \Rightarrow \land_{p \in ACPart\{f\}}(\land_{i \in \{1..|X|\}} X^i_{=^2_{Mf}}(p(i)) \land c, k) \lor d\]
if \(f \in F_{AC}\) and \(f \notin F_{AC1}\)

AC1 Decompose:
\[(f(X)_{=^2_{Mf}}(Y) \land c, k) \lor d \Rightarrow \land_{p \in AC1Part\{f\}}(\land_{i \in \{1..|X|\}} X^i_{=^2_{Mf}}(p(i)) \land c, k) \lor d\]
if \(f \in F_{AC1}\)

AC1 Collapse:
\[(f(X)_{=^2_{Mt}} \land c, k) \lor d \Rightarrow \land_{i \in \{1..|X|\}, i \neq i}(X^i_{=^2_{Mt}} \land c, k) \lor d\]
if \(f \in F_{AC1}\) and \(t \in V\) or root(t) \(\neq f\)

| Table 7: The transformation rules AC1-Match |
and \( f(X) \) if \(|X| > 1\). When \( f \in F_{AC} \) and \( X \) and \( Y \) are sequences of terms, then \( ACPartit(f, X, Y) \) is the set of all mappings \( p \) from \([1..|X|]\) to the set of all subsequences of \( Y \) such that for all \( j \in [1..|Y|] \) holds
\[
|Y|_{Y^j} = \sum_{i \in [1..|X|]} |p(i)|_{Y^j},
\]
and such that for all \( i \in [1..|X|] \) holds \(|p(i)| \geq 1\). When \( f \in F_{AC1} \) then \( AC1Partit \) is the set of all mappings \( p \) from \([1..|X|]\) to the set of all subsequences of \( Y \) such that for all \( j \in [1..|Y|] \) holds
\[
|Y|_{Y^j} = \sum_{i \in [1..|X|]} |p(i)|_{Y^j}.
\]

- When we want to find the substitutions with which a term \( s \) matches a term \( t \), we apply the rules in \textbf{AC1-Match} exhaustively to the matching problem \((t \downarrow_{Flatten\cup Zero}^2 = M s \downarrow_{Flatten\cup Zero}^2 \land T) \lor \bot\).

- We obtain the matching substitutions from a solved form \( \forall_{i \in [1..n]} (T, k_i) \) by removing redundant atomic problems from \( k_i \) for each \( i \in [1..n] \). Redundant are problems of the form \( x =^2_M y \), where \( x = y \), and problems \( x =^2_M t \), for which \( x =^2_M t' \) occurs in the rest of \( k_i \). Note that in the latter case, \( t =_{AC1} t' \) (see condition for \textbf{Eliminate}).

- A complete set of matching substitutions is computed. If a most general complete set of matching substitutions is wanted, redundant substitutions must be eliminated in an additional pass over the resulting set of substitutions.
Appendix B

Examples

We demonstrate the use of TRACONE by giving an example session. The result of an evaluation of an SML-statement is indicated by “>>”.

B.1 Rewriting

The file sign contains the signature from Example 2.1 in the syntax of TRACONE.

(* sign *)

\[

da : x \\
b : x \\
c : x \\
d : x \\
e : x \\
- : x x \\
f : x x \\
\]

76
\[ g \; : \; x \times \rightarrow x \]
\[ ac \; \text{infix} \; # \; : \; x \times \rightarrow x \]
\[ ac \; \text{infix} \; + \; : \; x \times \rightarrow x \]
\[ \text{zero} \; 0 \; \text{to} \; + \]
\[ ac \; \text{infix} \; * \; : \; x \times \rightarrow x \]
\[ \text{zero} \; 1 \; \text{to} \; * \]

The file \texttt{sign} is interpreted as a signature and a string as a term on this
signature.

\begin{verbatim}
val sign = read_signature "sign";
val t1 = Term.parse "1 + (0 + (f(x_1) + f(x_2)))" sign;
Term.print t1;
\end{verbatim}

\begin{verbatim}
\[ \gg \; f(x_1) + f(x_2) \]
\end{verbatim}

Note that terms are reduced to normal form with respect to \textit{Flatten} \cup \textit{Zero}
after every operation on terms such as parsing. \textit{TRACONE} uses the extended
notion of subterms and positions as described in Section 2.1.

\begin{verbatim}
val t2 = Term.parse "g(0,x_1) * (x_1 + x_2) * 0" sign;
Term.print t2;
\end{verbatim}

\begin{verbatim}
\[ \gg \; (x_1 + x_2) * g(0,x_1) * 0 \]
\end{verbatim}

\begin{verbatim}
positions t2;
\end{verbatim}

\begin{verbatim}
\[ \gg \; [\text{position (\[ [],[2,3])}, \text{position (\[ [],[1,3])}, \text{position (\[ [],[1,2]),}
\] \text{position (\[ [],[]), \text{position (\[3,[]), \text{position (\[2,[]),}
\] \text{position (\[2,2],[]), \text{position (\[2,1],[]), \text{position (\[1,[]),}
\] \text{position (\[1,2],[]), \text{position (\[1,1],[]))]
\end{verbatim}

\begin{verbatim}
Term.list_print (subterms t2);
\end{verbatim}
\[
\begin{align*}
\gg & \ g(0, x_1) \ast 0 \\
& (x_1 + x_2) \ast 0 \\
& (x_1 + x_2) \ast g(0, x_1) \\
& (x_1 + x_2) \ast g(0, x_1) \ast 0 \\
& 0 \\
& g(0, x_1) \\
x_1 \\
& 0 \\
x_1 + x_2 \\
x_2 \\
x_1
\end{align*}
\]

Note that the order of the subterms in an \textit{AC}-term may change. The file \textsc{ring} contains a convergent \textit{AC1}-rewrite system for the algebra \textit{commutative ring with unit} (see [BPW 89]).

(* \textsc{ring} *)

\[
\begin{array}{l}
\frac{u + (-v) + v}{u} \Rightarrow u \\
\frac{-(-u)}{u} \Rightarrow u \\
\end{array}
\]

\[
\begin{array}{l}
u = 0 \quad v = 0 \\
\frac{- (u + v)}{(-u) + (-v)} \\
\frac{w * (u + v)}{w * u + (w * v)} \\
u = 1 \\
\frac{u * 0}{0} \\
u = 1 \\
\frac{(-v) * u}{-(v * u)} \\
\end{array}
\]

The file \textsc{ring} is interpreted as a rewrite system and the terms \texttt{t3} and \texttt{t4} are reduced to normal form with respect to this rewrite system using \textit{AC1}-matching.
val ring = Trs.read "ring" sign;
val t3 = Term.parse "f(x_1) + f(x_2) + (¬(f(x_1)))" sign;
Term.print t3;
  >>= (¬(f(x_1))) + f(x_2) + f(x_1)
val normal = Trs.normal_form ring t3;
Term.print normal;
  >>= f(x_2)
val t4 = Term.parse "f((x_1 + x_2) * g(x_3,x_1))" sign;
Term.print t4;
  >>= f((x_1 + x_2) * g(x_3,x_1))
val normal = Trs.normal_form ring t4;
Term.print normal;
  >>= f((x_1 * g(x_3,x_1)) + (x_2 * g(x_3)))

Note that the rewriting facilities of TRACONE represent the implementation of a programming language based on term rewriting, in which functions can be declared as being associative commutative and as identities to associative commutative functions.

**B.2 Constraint Solving**

The solving of the constraint \( C_1 = x_1 \# x_2^2 = x_3 \# x_4 \) demonstrates the solving of linear diophantine equations and the introduction of new variables.

val solutions_1 = Constraint.solve C_1;
Constraint.list_print solutions_1;
>> (* Constraint: *)
   (* Substitution *)
   x_1  <-  x_6  #  x_5
   x_2  <-  x_8  #  x_7
   x_3  <-  x_8  #  x_6
   x_4  <-  x_7  #  x_5

   (* Constraint: *)
   (* Substitution *)
   x_1  <-  x_3  #  x_5
   x_4  <-  x_2  #  x_5

   (* Constraint: *)
   (* Substitution *)
   x_1  <-  x_5  #  x_4
   x_3  <-  x_2  #  x_5

   (* Constraint: *)
   (* Substitution *)
   x_2  <-  x_3  #  x_5
   x_4  <-  x_5  #  x_1

   (* Constraint: *)
   (* Substitution *)
   x_3  <-  x_2
\[ x_4 \leftarrow x_1 \]

(* Constraint: *)

(* Substitution *)

\[ x_2 \leftarrow x_5 \# x_4 \]

\[ x_3 \leftarrow x_5 \# x_1 \]

(* Constraint: *)

(* Substitution *)

\[ x_3 \leftarrow x_1 \]

\[ x_4 \leftarrow x_2 \]

val number_of_solutions = length solutions_1;

>> val number_of_solutions = 7 : int

The presence of the indentity 0 for the function symbol + makes all but one solution of \( C_2 = x_1 + x_2 \# x_3 + x_4 \) redundant.

val solutions_2 = Constraint.solve C_2;

Constraint.list_print solutions_2;

>> (* Constraint *)

(* Substitution *)

\[ x_1 \leftarrow x_6 + x_5 \]

\[ x_2 \leftarrow x_8 + x_7 \]

\[ x_3 \leftarrow x_8 + x_6 \]
\[ x_4 \leftarrow x_7 + x_5 \]

\[
\text{val number_of_solutions = length solutions_2;}
\]

\[ \gg \text{val number_of_solutions = 1 : int} \]

The following examples are taken from the benchmark acuni in [BHK 88] (acuni-001, acuni-036, acuni-064, acuni-079, acuni-096) and were run on a SUN-4 (Sparc).

For \( C_3 = x#a#b^2=c#d#e \), we obtain:

\[ \gg \text{Computing time for constraint solving in seconds: 0.450000} \]

\[
\text{val number_of_solutions = 2 : int}
\]

For \( C_4 = x#y#a^2=u#v#w#c \), we obtain:

\[ \gg \text{Computing time for constraint solving in seconds: 3.200000} \]

\[
\text{val number_of_solutions = 204 : int}
\]

For \( C_5 = x#x#a^2=u#v#c#d \), we obtain:

\[ \gg \text{Computing time for constraint solving in seconds: 1.190000} \]

\[
\text{val number_of_solutions = 60 : int}
\]

For \( C_6 = x#x#y^2=u#v#c#d \), we obtain:

\[ \gg \text{Computing time for constraint solving in seconds: 3.580000} \]

\[
\text{val number_of_solutions = 228 : int}
\]

For \( C_7 = x#x#x^2=u#v#w#c \), we obtain:
Computing time for constraint solving in seconds: 172.070000

val number_of_solutions = 6006 : int

The solving of the constraint $C_8 = x_1 \neq 1 \land x_1 = x_2 + x_3$ demonstrates the application of the transformation rule Disequation Decompose in Disequation-AC1-Unify.

val solutions_8 = Constraint.solve C_8;
Constraint.list_print solutions_8;

(* Constraint: *)
(* Substitution *)
x_1 <= x_2 + x_3
(* Disequations *)
x_2 /= 0 x_2 /= 1

(* Constraint: *)
(* Substitution *)
x_1 <= x_2 + x_3
(* Disequations *)
x_3 /= 0 x_2 /= 0

(* Constraint: *)
(* Substitution *)
x_1 <= x_2 + x_3
(* Disequations *)
x_2 /= 1 x_3 /= 1
(* Constraint *)
(* Substitution *)
\[ x_1 \leftarrow x_2 + x_3 \]
(* Disequations *)
\[ x_3 \neq 0 \quad x_3 \neq 1 \]

val number_of_solutions = length solutions_8;

>> val number_of_solutions = 4 : int