Unification Algorithms for Boolean Rings

Bernard Crone-Rawe

SEKI Working Paper SWP-89-01

Unification Algorithms for Boolean Rings

Diplomarbeit

Bernard Crone - Rawe

Betreuer: Prof. Ph.D. Jörg Siekmann
Dipl.-Math. Hans-Jürgen Bürckert
Dr. rer.nat. Manfred Schmidt-Schauß

Fachbereich Informatik
Universität Kaiserslautern

Januar 1989

Erklärung:

Hiermit erkläre ich, daß die vorliegende Diplomarbeit selbstständig verfaßt habe und keine andere als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kaiserslautern, den 11.1.1989

Abstract.

We are interested in unification problems in free Boolean rings, i.e. we investigate equation solving over these structures. The two major issues are unification in free Boolean rings and unification in a combination of a free Boolean ring with free function symbols.

For unification in free Boolean rings we present the two approaches, which are pursued by today's scientific community. We disclose the origin of those methods, explain and verify them and reveal their advantages and disadvantages. Finally the acquired knowledge is used to introduce several improvements to the implementation of the algorithms.

The combination algorithm is a specific application of a more general algorithm operating on the combination of an arbitrary and a simple theory. Therefore besides describing the ideas and steps of the general algorithm, properties of the Boolean ring theory are utilized to build additional performance enhancing procedures into this algorithm. We receive a validation of this algorithm by comparing it to other algorithms of the relatively new field of combination algorithms.

Key words:

Boolean ring, Boolean algebra, Unification, Equational Theories, Theory-unification, Combination of Equational Theories, Theorem Prover.

Acknowledgements.

In particular I would like to express my gratitude to Hans-Jürgen Bürckert for his intensive supervision. Then I would like to thank Hans-Jürgen Ohlbach and Axel Präcklein for their comments during the implementation. I am grateful to Alfred Gabel for contributing an independent view to the documentation. Finally I thank the members of the AG Siekmann, who provided a scientifically stimulating and socially satisfying environment for all my activities.

Zusammenfassung.

In dieser Diplomarbeit werden Unifikationsprobleme in Bool'schen Ringen untersucht. Sie ist in zwei Teile gegliedert: Unifikation in freien Bool'schen Ringen, Unifikation in der Kombination eines freien Bool'schen Rings mit freien (uninterpretierten) Funktionssymbolen.

Für die Unifikation in freien Bool'schen Ringen werden die beiden Ansätze präsentiert, die heutzutage in der Wissenschaft verfolgt werden. Wir geben Aufschluß über den Ursprung der Ansätze, erklären und verifizieren die Algorithmen und zeigen ihre Schwächen und Stärken auf. Die gewonnen Erfahrungen wurden genutzt, um weitere Verbesserungen in die implementierten Versionen einzubauen.

Der Kombinationsalgorithmus ist eine spezielle Anwendung einer allgemeineren Prozedur, der die Kombination einer simplen und einer beliebigen Theorie behandelt. Die Ideen und Schritte des allgemeinen Algorithmus werden erklärt. Der Algorithmus wurde durch Einbau von Prozeduren weiterentwicklet, die spezielle Eigenschaften der Bool'schen Ringe ausnutzen. Die Einordnung des Algorithmus wird durch einen Vergleich mit anderen Algorithmen des relativen neuen Feld der Kombinationsalgorithmen gewonnen.

Schlüsselworte:

Bool'sche Ring, Bool'sche Algebra, Unifikation, Gleichheitstheorie, Theorieunifikation, Kombination von Gleichheitstheorien, Automatischer Beweiser.

Danksagungen.

Insbesondere möchte ich bei Hans-Jürgen Bürckert für die intensive Betreuung der Diplomarbeit bedanken. Hans-Jürgen Ohlbach und Axel Präcklein bin ich für ihre Tips während der Implementierung dankbar. Alfred Gabel bekunde ich Dank für seine Kommentare zu meinen Dokumentierungsversuchen. Schließlich bedanke ich bei all den Mitarbeitern der AG Siekmann, die mir für meine Diplomarbeit eine wissenschaftlich stimulierende sowie eine menschlich herzliche Umgebung bereitet haben.

CONTENTS:

1. MOTIVATION

Solving equations in particular algebraic theories, which is also known as E-unification (G. Plotkin [Plo 72], J. Siekmann [Sie 78], G. Huet and D.C. Oppen [HD 80], F. Fages and G. Huet [FH 83, 86]), has attracted a great deal of interest in recent years, as it is the basic inference mechanism in algebraic manipulation of formulae, automated reasoning and logic programming languages. A detailed account is given by J. Siekmann in his survey paper on unification [Sie 87].

Recently researchers are involved with unification in Boolean rings. They have been enticed both by its theoretical merits - the unitarity of Boolean ring unification itself (at most one most general unifier) and the existence of a deterministic normal form for Boolean ring terms - and by the practical relevance of Boolean ring unification. By enabling the manipulation of hardware descriptions it gives us the capacity to solve numerous problems in digital hardware design like simulation, verification, synthesis, simplification, specialization or debugging [Sim 87].

Another important application area is propositional logic. The propositions with their operators ($\wedge$, $\vee$, $\neg$, etc.) can be transformed into equivalent Boolean ring terms. To test, if a proposition is a tautology or unsatisfiable, we do not even need unification. It is sufficient to simplify the term and to look, if we obtain 1 or 0. For proposition with incorporated variables we are able to determine the most general values of those variables, which make the proposition a tautology or unsatisfiable, by unifying the corresponding term with 1 or 0.

Yet another view of Boolean rings is treated by R. Sikorski. In his treatise he stated, that there is also a set-theoretical aspect of Boolean rings beside its algebraic aspect [Sik 67]. This view regards Boolean ring theory as a generalization of the set-theoretical notions of a field of sets. Then we can generate statements on sets and the operators ($\cup$, $\cap$, $\setminus$) and use these statements to perform computations of sets as required in solving combinatorial problems.

Today unification in free Boolean rings is handled in two different fashions. One approach has been developed by U. Martin and T. Nipkov [MN 86], the other approach by W. Büttner and H. Simonis [BüSi 87]. When those methods were presented in papers and various conferences, a discussion about the origin of these methods arouse. It turned out, that the methods were comparatively old. They were invented at the turn of the 19th century. Büttner's and Simonis's idea takes the credit of being the oldest one. It is already described in G. Boole's book "The Mathematical Analysis of Logic" in 1847 [Boo 47]. This is this the book, where Boole introduces the notion of Boolean ring with the primitive operators <symmetric difference> and <intersection> rather than <union> and <intersection> (Boolean algebra) for representation of Boolean terms. Boole proceeds by successive elimination of variables thereby obtaining reduced problems. The Martin/Nipkov approach was first conferred by

L. Löwenheim in 1908 [Löw 08]. L. Löwenheim gave a formula for the most general solution of a Boolean equation expressed in terms of a particular solution. The two methods, along with a range of techniques for finding a particular solution, can be found in S. Rudenau's book on Boolean equations [Rud 74]. In this paper we aim to give a more detailed and still comprehensive study of Boolean ring unification. The examined unification methods consist of two versions of "Deducing from a Particular Solution", which differ in the way the particular solution is computed, and the method "Successive Variable Elimination". Our study consists of description, verification, comparison and implementation of the methods. We also give an account of several improvements for the algorithms. This adds up to a presentation of a set of performance enhanced algorithms with their pros and contras, which should enable the reader to select the right algorithm for his specific application.

Recently several papers are tackling unification problems in the combination of equational theories. The first to consider the combination of unification algorithms were M. Stickel [Sti 75, Sti 81], M. Livesey and J. Siekmann [LS 78] and F. Fages [Fag 84]. They combine associative and commutative functions (AC) with uninterpreted (free) function symbols. More general combination problems were examined by several other authors [Yel 87, Kir 85, Tid 86, Hero 86]. Their algorithms subject to some restrictions. C. Kirchner requires the theories to be cycle free and K. Yelick and A. Herold manage only regular and collapse free theories. E. Tidén considered the most general case so far, the combination of collapse free theories. Most recently some new algorithms have been developed, which have a more loosened restriction on the theories [BJS 88, Schm 88]. The algorithm by A. Boudet, J.-P. Jouannaud and M. Schmidt-Schauß [BJS 88] and the one by M. Schmidt-Schauß [Schm 88] handle the combination of an arbitrary and a simple theory. In the same year M. Schmidt-Schauß [Schm 88] presented another algorithm without any restrictions on the theories. In the second part of this thesis we study the combination of a free Boolean ring with free function symbols, a special case of the restricted Schmidt-Schauß algorithm. This algorithm was selected, as it is more suitable for our specific application. It utilizes the characteristics of the simple theory and thereby the efficiency is increased decisively.

The combination of a free Boolean ring with free function symbols is of particular interest, because its unification is decidable and finitary. In this paper we recall the combination algorithm as sketched by Schmidt-Schauß and present some realization possibilities for particular subproblems. We have incorporated additional functions based on the properties of the Boolean ring into the algorithm to improve its efficiency.

This paper is organized as follows. In Section 2 we introduce the necessary theoretical notions in order to define Boolean ring problems and their solutions. The chapter is divided into an algebraic and a unification part. The first part recalls the concepts of Boolean algebras and Boolean rings and an

important theorem due to M.H. Stone, which discloses their duality [Sto 36]. The second part of this section covers syntactic and equational unification as well as unification in the combination of equational theories. For both parts we are consistent with the notations of the common literature as by G. Grätzer [Grä 79], S. Burris and H. P. Sankappanavar [BS 79], J. Siekmann [Sie 87], G. Huet and D. C. Oppen [HD 80]. Section 3 deals with unification in free Boolean rings and Section 4 with unification in the combination of free Boolean rings with free function symbols. The examples chosen for the combination algorithm are mainly those discussed at the unification workshop at Val d'Ajol in 1987 [Kir 87]. Section 5 describes the implementation of the algorithms. It is aimed at readers, who intend to utilize our system. We give an account of used datastructures and different procedures available to the user. We also present some structures of the HADES (Highly Adaptable DEduction System) [Ohl 88], a theorem proving environment currently being developed at the University of Kaiserslautern. This is necessary, as we have implemented our algorithm directly in this system. In Section 6 other applications of our algorithm are discussed. In Chapter 7 we summarize the results. In the appendix we have included results from our test runs. In particular it consists of comparisons of the three unification algorithms for Boolean rings and test runs of the already mentioned well-known examples for the combination of Boolean rings with free function symbols.

2. INTRODUCTION

2.1 From Boolean Algebras to Boolean Rings

In this chapter I intend to describe briefly the concepts and basic properties of Boolean algebras and of Boolean rings. To some extent those have been used in designing and improving the unification algorithms for free Boolean rings. In particular I will recall the definitions of a Boolean algebra and a Boolean ring, discuss several normal forms of terms of the theories and I will recall the well-known theorem, which states the equivalence between Boolean algebras and Boolean rings.

Definition: **Boolean Algebra**

A Boolean algebra is a structure $\langle B, \cup, \cap, \neg, 0, 1\rangle$, where B is a set, $\cup$: $B \times B \to B$ and $\cap$: $B \times B \to B$ are binary operators called disjunction and conjunction, and $\neg$: $B \to B$ is an unary operation called negation. 0 an 1 are elements of B, with $0 \neq 1$. A Boolean algebra is a complemented distributive lattice. Therefore the following properties have to be satisfied:
for every $x, y, z \in B$:

commutativity of $\cup$	$x \cup y$	=	$y \cup x$	**$\langle B, \cup, \cap\rangle$ is a lattice**
commutativity of $\cap$	$x \cap y$	=	$y \cap x$	
associativity of $\cup$	$(x \cup y) \cup z$	=	$x \cup (y \cup z)$	
associativity of $\cap$	$(x \cap y) \cap z$	=	$x \cap (y \cap z)$	
absorption law for $\cup$	$x \cup (x \cap y)$	=	x	
absorption law for $\cap$	$x \cap (x \cup y)$	=	x	
distributivity	$x \cap (y \cup z)$	=	$(x \cap y) \cup (x \cap z)$	
	$x \cup (y \cap z)$	=	$(x \cup y) \cap (x \cup z)$	
zero element 0	$x \cap 0$	=	0	
unit element 1	$x \cup 1$	=	1	
complementation	$x \cup \neg x$	=	1	
	$x \cap \neg x$	=	0	

As we know the following equalities are valid in the Boolean algebra $\langle B, \cup, \cap, \neg, 0, 1\rangle$:

idempotency	$x \cup x$	=	x
	$x \cap x$	=	x
De Morgan laws	$\neg (x \cup y)$	=	$\neg x \cap \neg y$
	$\neg (x \cap y)$	=	$\neg x \cup \neg y$
double negation	$\neg\neg x$	=	x
absorption	$x \cup (\neg x \cap y)$	=	$x \cup y$
	$x \cap (\neg x \cup y)$	=	$x \cap y$

and we have the following equivalences

$$x \cup y = 0 \quad \Leftrightarrow \quad x = y = 0$$
$$x \cap y = 1 \quad \Leftrightarrow \quad x = y = 1$$

To be able to work with Boolean algebras we would like to have a representative form of terms. In Boolean algebras there are two different forms: the **conjunctive form** and the **disjunctive form.** The conjunctive form is defined as $\bigcap_i \bigcup_j \Phi_{ij}$, where each Φ_{ij} is atomic or negated atomic. The disjunctive form is defined as $\bigcup_i \bigcap_j \Phi_{ij}$, where again each Φ_{ij} is atomic or negated atomic. Each term of the Boolean algebra can be transformed into both the conjunctive form and the disjunctive form.This can easily be proven by induction on the length of the term using the distributivity and the De Morgan laws. So now we can transform each term into a deterministicly structured term. Yet to have a deterministic form, there should be only one conjunctive or disjunctive form possible. This is not the case in a Boolean algebra. For example in the Boolean algebra $\langle B, \cup, \cap, \neg, 0, 1\rangle$ with $B = \{x, y, z\}$, there are the two equal disjunctive forms $t_1 = (\neg x \cap y) \cup (x \cap y) \cup (x \cap z)$ and $t_2 = y \cup (x \cap z)$. The term $t_3 = (y \cup z) \cap x \cap (x \cup \neg z)$ can be transformed into another equal and smaller conjunctive form $t_4 = (y \cup z) \cap x$. Such a transformation, which reduces the term until no more reductions are possible, is called **minimization** and the reduced term is said to be in **normal form.** Several algorithms are known, which minimize terms into a conjunctive or disjunctive normal form. Now the questions arises, are the normal forms deterministic. Again we have to deny this attribute. Proof by example: The Boolean algebra $\langle B, \cup, \cap, \neg, 0, 1\rangle$ with $B = \{x, y, z\}$. The term $t_1 = (\neg x \cup y) \cap (\neg y \cup z) \cap (x \cup \neg z)$ is a conjunctive normal form and the term $t_2 = (x \cup \neg y) \cap (y \cup \neg z) \cap (\neg x \cup z)$ is an equal conjunctive normal form.

$$\begin{aligned} t_1 &= (\neg x \cup y) \cap (\neg y \cup z) \cap (x \cup \neg z) \\ &= ((\neg x \cap \neg y) \cup (\neg x \cap z) \cup (y \cap z)) \cap (x \cup \neg z) \\ &= (x \cap y \cap z) \cup (\neg x \cap \neg y \cap \neg z) \\ &= ((x \cap y) \cup (x \cap \neg z) \cup (\neg y \cap \neg z)) \cap (\neg x \cup z) \\ &= (x \cup \neg y) \cap (y \cup \neg z) \cap (\neg x \cup z) = t_2 \quad \blacksquare \end{aligned}$$

There also equal disjunctive normal forms like term $t_3 = (\neg x \cap y) \cup (\neg y \cap z) \cup (x \cap \neg z)$ and term $t_4 = (x \cap \neg y) \cup (y \cap \neg z) \cup (\neg x \cap z)$. So there is no deterministic minimized form, but naturally there are maximized disjunctive and conjunctive forms. And those can be deterministic, if we have defined an ordering on the terms. This is necessary because of associativity and commutativity of the conjunction and disjunction. To enable the definition of maximized forms, we define the concept of **minterm** and **maxterm**. In the Boolean algebra $<B, \cup, \cap, \neg, 0, 1>$ with $B = \{\Phi_1, \ldots, \Phi_n\}$ a minterm is $\bigcap_{i \in 1 \ldots n} \Phi_i$ and a maxterm is $\bigcup_{i \in 1 \ldots n} \Phi_i$, where each Φ_i is atomic or negated atomic. Then a disjunction of minterms is the maximized disjunctive form and a conjunction of maxterms the maximized conjunctive from. Provided the generating set B is finite, it is possible to convert each term into those forms. However the termsizes would grow exponentially, if we work with maximized terms.

Definition: **Boolean Ring**

As a Boolean ring we define a structure $<B, +, *, 0, 1>$, where B is a set, $+: B \times B \rightarrow B$ and $*: B \times B \rightarrow B$, called addition and multiplication. 0 an 1 are two elements of B, with $0 \neq 1$. The following axioms make a structure a Boolean ring.

for every $x, y, z \in B$:

associativity of +	$(x + y) + z$	$=$	$x + (y + z)$	
commutativity of +	$x + y$	$=$	$y + x$	$<B, +, 0>$
zero element 0	$x + 0$	$=$	$0 + x = x$	is an abelian group
inverse for +	$x + x$	$=$	0	
associativity of *	$(x * y) * z$	$=$	$x * (y * z)$	$<B, *>$ is a semigroup
distributivity	$x + (y * z)$	$=$	$(x + y) * (x + z)$	
	$x * (y + z)$	$=$	$(x * y) + (x * z)$	
commutativity of *	$x * y$	$=$	$y * x$	
unit element 1	$x * 1$	$=$	$1 * x = x$	
idempotency	$x * x$	$=$	x	

$<B, +, *, 0>$ is a ring

The Boolean ring <B, +, *, 0, 1> has some useful properties, which we will make repeatedly use of in the course of this paper:

(i) $x = y \Leftrightarrow x + y = 0$

(ii) $x^n = x$ $\quad n > 0$

(iii) $x * 0 = 0$

(iv) $x * (x + 1) = 0$

Proof:

(i) $x = y \Leftrightarrow x + y = y + y \Leftrightarrow x + y = 0$ | addition of y and application of inverse of +

(ii) $x^n = x^{n-1} = x^{n-2} = \ldots = x * x = x$ | repeated application of idempotency

(iii) $x * 0 = x * (y + y) = (x * y) + (x * y) = 0$ | application of idempotency

(iv) $x * (x + 1) = x * x + x * 1 = x + x = 0$ | application of distributivity and inverse of +

The similarities between Boolean rings and Boolean algebras lead to research aimed to determine their relationship. M.H. Stone discovered, that every Boolean algebra can be transformed into a Boolean ring and vice versa (by weak isomorphism [Grä 79]). He denoted this with the following well-known theorem [Sto 35].

Theorem 2.1.1:

a) Let $\mathbf{B} = \langle B, \cup, \cap, \neg, 0, 1\rangle$ be a Boolean algebra. Let $\mathbf{B}^{\otimes}$ be the structure $\langle B, +, *, -, 0, 1\rangle$, where the operations are defined by ($x, y \in B$):

(1.1) $x + y = (x \cap \neg y) \cup (\neg x \cap y)$ <symmetric difference>

(1.2) $x * y = x \cap y$

(1.3) $-x = x.$

Then $\mathbf{B}^{\otimes}$ is a Boolean ring.

b) Let $\mathbf{R} = \langle R, +, *, 0, 1\rangle$ be a Boolean ring. Let $\mathbf{R}^{\otimes}$ be the structure $\langle R, \cup, \cap, \neg, 0, 1\rangle$, where the operations are defined by ($x, y \in R$):

(1.4) $x \cup y = x + y + x * y$

(1.5) $x \cap y = x * y$

(1.6) $\neg x = x + 1.$

Then $\mathbf{R}^{\otimes}$ is a Boolean algebra.

c) Given **B** and **R** as above we have $\mathbf{B}^{\otimes\otimes} = \mathbf{B}$, $\mathbf{R}^{\otimes\otimes} = \mathbf{R}$.

Proof:

a) Let $x, y, z \in B$

i) zero element 0

$x + 0 = (x \cap 1) \cup (\neg x \cap 0) = x$

ii) evidently: associativity of +

iii) commutativity of +

$$\begin{aligned} x + (y + z) &= (x \cap \neg (y + z) \cup (\neg x \cap (y + z)) \\ &= (x \cap \neg ((y \cap \neg z) \cup (\neg y \cap z)) \cup (\neg x \cap ((y \cap \neg z) \cup (\neg y \cap z))) \\ &= (x \cap y \cap z) \cup (x \cap \neg y \cap \neg z) \cup (\neg x \cap y \cap \neg z) \cup (\neg x \cap \neg y \cap z)) \\ &= (x \cap y \cap z) \cup (x \cap \neg y \cap \neg z) \cup (y \cap \neg z \cap \neg x) \cup (z \cap \neg x \cap \neg y)) \end{aligned}$$

since this result is symmetric in x, y, z; it follows that

$x + (y + z) = z + (x + y) = (x + y) + z$ | applying ii)

iv) inverse for +

$x + x = (x \cap \neg x) \cup (\neg x \cap x) = 0$

v) by hypothesis: associativity, commutativity and idempotency for *

vi) distributivity

$$\begin{aligned} x + (y * z) &= (x \cap \neg y \cap \neg z) \cup (\neg x \cap y \cap z) \\ &= (x \cap \neg y) \cap (x \cap \neg z) \cup (x \cap \neg y) \cap (\neg x \cap z) \\ &\quad \cup (\neg x \cap y) \cap (x \cap \neg z) \cup (\neg x \cap y) \cap (\neg x \cap z) \\ &= ((x \cap \neg y) \cup (\neg x \cap y)) * ((x \cap \neg z) \cup (\neg x \cap z)) \\ &= (x + y) * (x + z) \end{aligned}$$

$$\begin{aligned} x * (y + z) &= (x \cap y \cap \neg z) \cup (x \cap \neg y \cap z) \\ &= ((x \cap y) \cap (\neg x \cup \neg z)) \cup ((\neg x \cup \neg y) \cap (x \cap z)) \\ &= ((x \cap y) \cap \neg (x \cap z)) \cup (\neg (x \cap y) \cap (x \cap z)) \\ &= (x * y) + (x * z) \end{aligned}$$

vii) unit element 1

$x * 1 = 1 * x = x$

$\Rightarrow$ $\mathbf{B}^{\otimes}$ is a Boolean ring.

b) Let $x, y, z \in R$

i) evidently: commutativity of $\cup$

ii) by hypothesis: commutativity and associativity of $\cap$

iii) associativity of $\cup$

$x \cup (y \cup z) = x + (y + z + y * z) + x * (y + z + y * z)$

$= x + y + z + y * z + x * y + x * z + x * y * z$

since this result is symmetric in x, y, z; it follows that

$x \cup (y \cup z) = z \cup (x \cup y) = (x \cup y) \cup z$ | applying i)

iv) absorption law for $\cup$

$x \cup (x \cap y) = x + (x * y) + x * (x * y) = x$

v) absorption law for $\cap$

$x \cap (x \cup y) = x * (x + y + x * y) = x$

vi) distributivity

sufficient to prove one law, as in a lattice they imply each other:

$x \cap (y \cup z) = x * (y + z + y * z)$

$= x * y + x * z + x * y * z$

$= x * y + x * z + (x * y) * (x * z)$

$= (x \cap y) \cup (x \cap z)$

vii) zero element 0

$x \cap 0 = x * 0 = 0$

viii) unit element 1

$x \cup 1 = x + 1 + x * 1 = 1$

ix) complementation

$x \cup \neg x = x + (x + 1) + x * (x + 1) = x + x + 1 + x + x = 1$

$x \cap \neg x = x * (x + 1) = x + x = 0$

$\Rightarrow \mathbf{R}^{\otimes}$ is a Boolean algebra.

c1) Let $\mathbf{B} = \langle B, \cup, \cap, \neg, 0, 1 \rangle$ be a Boolean algebra with $x, y \in B$ and $\mathbf{B}^{\otimes} = \langle B, +, *, -, 0, 1 \rangle$.

i) $x * y = x \cap y$

ii) $x + 1 = (x \cap \neg 1) \cup (\neg x \cap 1) = \neg x$

iii) $x + y + x * y = x + y * (x + 1)$

$= (x \cap \neg (y \cap \neg x)) \cup (\neg x \cap (y \cap \neg x))$

$= x \cap \neg y \cup x \cup \neg x \cap y$

$= x \cup \neg x \cap y$ | absorption

$= x \cup y$

Thus $\mathbf{B}^{\otimes\otimes} = \mathbf{B}$.

c2) Let $\mathbf{R} = \langle R, +, *, 0, 1\rangle$ be a Boolean ring with $x, y \in R$ and $\mathbf{R}^{\otimes} = \langle R, \cup, \cap, \neg, 0, 1\rangle$.

i) $(x \cap \neg y) \cup (\neg x \cap y) = (x * (y + 1)) + ((x + 1) * y) + ((x * (y + 1)) * ((x + 1) * y)$
$= (x + x * y) + (y + x * y) + 0 = x + y$

ii) $x \cap y = x * y$

Thus $\mathbf{R}^{\otimes\otimes} = \mathbf{R}$. ■

The theorem implies that certain properties of terms of a Boolean ring can be transformed into equivalent properties of terms of a Boolean algebra and vice versa. What that means for the Boolean ring unification algorithms, which are presented in this thesis, is discussed in Chapter 6, where we confer several applications of our algorithms.

We decided to use a deterministic minimal form for terms in our algorithms. This has the effect, that we are able to detect equal terms by a fast syntactical comparison.The Boolean ring has such a normal form after we have defined an ordering on terms. This is necessary because of AC properties of addition and multiplication. In this thesis we use two versions of normal forms for Boolean rings, the **disjunctive normal form** and the **polynomial** form, as introduced by Martin/Nipkov [MN 86]. The polynomial form originates in the disjunctive normal form. The difference will be pinpointed in the following section.

Let B be a Boolean ring $\langle V \cup C, +, *, 0, 1\rangle$, where C and V are disjoint set of symbols interpreted as constants and variables. The disjunctive normal form of term in B is defined as $\Sigma_i \prod_j b_{ij}$, where $b_{ij} \in V \cup C \cup \{1\}$. Every term can be easily transformed to the disjunctive normal form by applying the distributive laws. The polynomial form can be computed from the disjunctive normal form by collecting all factors of each element of V^n into sums. In other words, the polynomial form is $\Sigma_{U \subseteq V} c_U v_U$, where v_U is a product of variables ($\prod_{x \in U} x$, $U \subseteq V$) and c_U is a sum of products of constants ($\Sigma_i \prod_j d_{ij}$, $d_{ij} \in C$). A polynomial term is **homogeneous**, if $c_{\emptyset} = 0$. This means, that all direct subterms have variables in them.

Examples: t_1 is equal to t_2

disjunctive normal form: $t_1 = (x * y * a) + (x * y * b) + (x * z) + (a * b)$

polynomial form: $t_2 = (a + b) * (x * y) + (x * z) + (a * b)$

homogeneous term: $t_3 = (a + b) * (x * y) + (x * z)$

Notational Remark:

In this paper from now, we will sometimes omit the multiplication symbols and use only the

essential parenthesis for Boolean ring terms. This is a common procedure, which enables a better perception (example: $t_3 = (a + b)xy + xz$).

2.2 Unification

This chapter gives a survey of unification as needed in the sequel. For more detailed information see Siekmann's survey paper on unification [Sie 88]. We sketch the unification concept and introduce theory-unification. We discuss the attributes of Boolean ring unification as a special case of the equational theory and the combination of Boolean ring and free theory unification as a case of the combination of equational theories.

Unification deals with problems of the following kind. Let s, t be two terms. Compute a substitution σ of the variables in s and t by terms, whose application to the terms s and t satisfies $\sigma s = \sigma t$. This substitution is called solution of the equation s = t or **unifier** of s and t. More precisely we can define substitutions with the help of some algebraic notions. A **signature** Σ is a set of function symbols, where a non-negative integer, the **arity**, is assigned to each function $f \in \Sigma$. Function symbols with arity 0 are also called constants. An **algebra** A is a pair (A, Σ), where A is the carrier and Σ a signature, such that every n-ary function f assigns an operation $f^A: A^n \rightarrow A$. With $\tau(V, \Sigma)$ we denote the **free term algebra** over a signature Σ and a countably infinite set of Variables V. A **substitution** σ is an endomorphism on $\tau(V, \Sigma)$ such that $\{x \in V \mid \sigma x \neq x\}$ is finite. A substitution is usually represented by a set of variable term pairs $\sigma = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ with $x_i \neq t_i$. The set $DOM(\sigma) = \{x_1, \ldots, x_n\}$ is called **domain**, the set $COD(\sigma) = \{t_1, \ldots, t_n\}$ is the **codomain** and the set $VCOD(\sigma) = Variables(COD(\sigma))$ is called the **variables introduced** by σ. The **cardinality** of a substitution is the number of variables in its domain. The **restriction** of a substitution σ to a set of variables V is substitution $\sigma_{|V}x = \sigma x$ for $x \in V$ and $\sigma_{|V}x = x$ otherwise. A substitution λ is an **instance** of a substitution σ, if $\lambda = \tau\sigma$ for a substitution τ. The substitution σ is more **general** than substitution λ. In order to have access to subterms of term t, we use occurrences [Hue 80]. The subterm of term t at occurrence π is denoted by $t\backslash\pi$ and the term constructed form t by replacing the subterm at the occurrence π by term s is denoted as $t[\pi \leftarrow s]$.

2.2.1 The Syntactic Theory

The syntactic theory is the theory of syntactic equality, where terms are equal, if they are syntactical equal. J. Herbrand [Her 30] was the first to disclose, that there is a most general solution, which is unique up to variable renaming, to any equation s = t, called **most general unifier** of s and t, indicated by mgu (s, t). Then A. Robinson [Rob 65] published a paper on the resolution principle, in which he presented a unification algorithm and proved that it computes the most general unifier. We will give a version of the **Robinson** algorithm by a set of rules being applied to the equation system $\Gamma = \{s_1 = t_1, \ldots, s_n = t_n\}$. The notation goes back to a similar set by Herbrand [Her 30] (x, y, z_i are variables and p_i, q_i, s_i, t are terms and f, g are different function symbols). Let Γ be an equation system.

(i) Remove equations $t =^{1)} t$ from Γ.
(ii) Replace every equation $f(p_1, \ldots, p_n) = f(q_1, \ldots, q_n)$ by the equations $p_1 = q_1, \ldots, p_n = q_n$.
(iii) Substitute for every equation x = t, if $x \notin$ Variables(t), all other occurrences of x in Γ by t.
(iv) Replace every equation t = x by x = t, if t is non variable term.

Failure in these two cases:
(v) equation $f(p_1, \ldots, p_n) = g(q_1, \ldots, q_n)$ in Γ. < failure by **clash** >
(vi) equation x = t in Γ and x occurs in t < failure by **occur check** >

An equation system Γ is solved, if there are only equations of the type $z_i = s_i$ $(1 \leq i \leq n)$ and all variables z_i are pairwise different and occur nowhere else in Γ. The most general unifier is the substitution $\sigma = \{z_1 \leftarrow s_1, \ldots, z_n \leftarrow s_n\}$.
The representation of a set of related unification problems cannot only be achieved by an equation system, but also by an multi-equation system. A multi-equation system (MS) consists of multi-equations. A multi-equation (ME) $t_1 = \ldots = t_n$ is shorthand for the equations $t_1 = t_2$, $t_2 = t_3$, ..., $t_{n-1} = t_n$. Therefore any equation system can be easily converted to a multi-equation system using the transitivity of the relation "=" and vice versa. A. Martelli and U. Montanari have developed a unification algorithm [MM 79], which operates on multi-equation systems.

1) In the algorithm '=' is seen as a directed operation.

Again we give a version of the algorithm as a set of rules (x, x_i are a variables, s_i, t_i are terms, T is a set of terms and r_i is a term, which is not a variable; a ME is represented as a set):

(i) Merge:

Replace two multi-equations $(ME_1 \cup x)$, $(ME_2 \cup x)$ by the multi-equation $(ME_1 \cup ME_2)$

(ii) Decomposition:

Replace a multi-equation of the form $ME = \{f(s_1, \ldots, s_n), f(t_1, \ldots, t_n)\} \cup T$
by multi-equations $\{f(s_1, \ldots, s_n)\} \cup T, \{s_1, t_1\}, \ldots, \{s_n, t_n\}$

(iii) Occurs in check:

Let there be a set of multi-equations $\{ME_1, \ldots, ME_n\}$ such that there is a variable x_i and a term t_i in ME_i.
If $x_i \in$ Variables(t_{i+1}) for $1 \leq i \leq n-1$, $x_n \in$ Variables(t_1),
then stop with **fail**.

From the solved multi-equation system, where each multi-equation consists of a set of variables and at most one non-variable term, we can easily construct the most general unifier. For a detailed account of the used datastructures see Chapter 5.

2.2.2 Equational Theory

In the previous chapter we discussed syntactical unification, where function symbols have no properties. We now consider unification, where properties of function symbols are defined by equations on terms. An **equational theory** is defined as a pair (Σ, E), where E is a set of equations, the **axioms**, and Σ is the signature, which contain at least the function symbols occurring in the axioms. Such symbols are called **interpreted** function symbols and the remaining ones, those not occurring in the axioms, **free** function symbols. We usually drop the signature in the definitions above, when it is clear from the context. An algebra A **satisfies** E or is a model of E ($A \models E$), if for every assignment of elements in A to variables in an equation $t_l = t_r$ in E, the corresponding equation is valid in A. We say an equation is a consequence of E ($E \models s = t$), if every model A of E also satisfies $s = t$. For $E \models s = t$ we will use $s =_E t$ and instead of saying $s = t$ is a consequence of E we will call s and t **E-equal**. Note that the relation $=_E$ is a congruence relation on the free term algebra $\tau(V, \Sigma)$. A finite set of equations $\Gamma = \{(s_i, t_i) \mid 1 \leq i \leq n\}$ together with a theory E is called an equation system or E-unification problem. A substitution σ is a solution or an **E-unifier** of Γ, if $\sigma s_i =_E \sigma t_i$ $(1 \leq i \leq n)$. E-equality of substitutions is defined as $\sigma =_E \tau$, if $\forall\, x \in V\ \sigma x =_E \tau x$. E-equality of substitutions,

which is restricted to a set of variables W ($\forall x \in W\ \sigma x =_E \tau x$), is denoted by $\sigma =_E \tau\ [W]$ and we say σ and τ are **E-equal on W**. We say a substitution σ is an **E-instance** of a substitution τ or τ is more **general** than σ modulo the theory E on a set variables W ($\tau \leq_E \sigma\ [W]$), if there exists a substitution λ with $\lambda\tau x =_E \sigma x$ for all $x \in W$. The set of all E-unifiers for an E-unification problem Γ is denoted by $U_E(\Gamma)$. A **complete** set $cU_E(\Gamma)$ of unifiers of Γ is a set satisfying correctness ($cU_E(\Gamma) \subseteq U_E(\Gamma)$) and completeness ($\forall \sigma \in U_E(\Gamma)\ \exists\ \tau \in cU_E(\Gamma): \tau \leq_E \sigma\ [V(\Gamma)]$). Such a complete set is **minimal** or a set of most general unifiers, if it additionally satisfies minimality ($\forall \sigma, \tau \in cU_E(\Gamma)\ \tau \leq_E \sigma\ [V(\Gamma)] \Rightarrow \tau = \sigma$). Minimal sets are designated as $\mu U_E(\Gamma)$.

This following commonly known classification of equational theories and unification problems is based on the cardinality of $\mu U(\Gamma)$: A theory E is **unitary**, if $\mu U(\Gamma)$ exists and has at most one element, **finitary**, if $\mu U(\Gamma)$ exists and is always finite, **infinitary**, if $\mu U(\Gamma)$ exists and is sometimes infinite and **nullary**, if $\mu U(\Gamma)$ does not always exist.

Finally we explain some terms, which are generally used to classify equational theories. An equation $s =_E t$ is **regular**, if Variables(s) = Variables(t). An equation is a **collapse** equation, if it is of the form $t =_E x$, where t is non-variable term and x is a variables. A theory is regular, if all axioms are regular and a theory is collapse free, if no axiom is a collapse axiom. A theory is cycle free or **simple,** if a term never can be E-equal to one of its proper subterms, which is equivalent to $\forall x \forall t\ x \in \text{Variables}(t) \Rightarrow U_E(x = t) = \emptyset$. Note that a cycle free theory is regular and collapse fee, but that the converse is false [BHS 87].

The Boolean ring theory is a special case of an equational theory. Its unification is unitary and there are minimal algorithms, which will be presented in Chapter 3. The theory itself is neither regular nor collapse free nor simple [BHS 87].

Syntactic unification, which can be seen as unifying equations, where just free function symbols occur, will be called unification in the **empty** or **free theory**. This unification is also unitary and there are minimal algorithms, which have been described in the previous chapter. The empty theory is simple (occur check of Robinson algorithm [Rob 65]) and hence collapse free and regular.

2.2.3 Combination of Equational Theories

Sometimes we have unification problems, which involve function symbols of different theories. These problems are handled in two totally different manners. On one side we have universal unification [Sie 88], on the other the idea of combination unification algorithms. While universal unification algorithms handle problems for a specified class of theories (input: an equation system and axioms), the combination algorithms try to obtain a unifier for $E = E_1 \cup E_2$ by combining known algorithms of E_1 and E_2. Again two cases are distinguishable: (i) the axioms of the two theories may have common function symbols like for associativity and commutativity [Sti 81] or (ii) the axioms of the theories cover different function symbols. The second part of this thesis deals with combination of theories with disjoint function symbols: We combine Boolean ring theory with free theory. This combination is of particular interest, because its unification is decidable and finitary.

Let all theories E_i contain the same set of free constants, but disjoint sets of function symbols. Let s be a term, then s has the **syntactical theory** E_i, if the top-level function symbol belongs to theory E_i. A term is a **proper E_i-term**, if s is an E_i-term but not a variable or a free constant. A subterm s of t is called **E_i-alien**, if every proper superterm of s in t is an E_i-term, but s is not an E_i-term. A subterm s of t is an **alien** subterm, if s and t have different theories. Note that free variables and free constants do not count as alien subterms. If s is also the maximal subterm with this property, we call it **direct alien** subterm. A term is called **pure**, if it has no alien subterms, otherwise the term is called **mixed** or **general**. An equation, a multi-equation or a multi-equation system is called pure, if only pure terms of one theory are involved. A term t in the combination of theories E+ has the **semantical theory** E_j, if a 'maximally collapsed' term t' with $t' =_{E+} t$ has the syntactical theory E_j. Maximally collapsed expresses that a term can not be simplified into a smaller term equal in the combination.

Examples:

Let E+ be our combination of Boolean ring and free functions symbols.

Then t = a + a + f(b) is a mixed term with alien subterm f(b).

Its syntactical theory is Boolean ring; its semantical theory the free theory, as $t =_{E+} t' = f(b)$

3. UNIFICATION ALGORITHMS FOR PURE BOOLEAN RINGS

3.1 Method of Successive Variable Elimination

W. Büttner and H. Simonis [BüSi 87] published this algorithm in 1987 with the intention to enable the embedding of boolean expressions into logic programming. However the origin of the method goes back to the 19th century, when G.Boole [Boo 47] and later E.Schröder [Sch 90] conferred its description. In the algorithm the **most general BR-unifier** σ of two terms t_1 and t_2 is computed. This is accomplished by successive elimination of the variables occurring in the terms. Thereby we obtain a *smaller* problem after each step. It is sufficient to present a version of it, which computes the unifier of $\langle t = 0 \rangle_{BR}$, as every equation of two terms $t_1 =_{BR} t_2$ can be converted into $t = t_1 + t_2 =_{BR} 0$. In the following t, a, b are terms of a Boolean ring theory BR.

Algorithm: **"Variable Elimination" Method**

Input: term t of Boolean ring BR.
Output: most general BR-unifier σ of term t and 0.

Step 1)	If $t =_{BR} 0$, then $\sigma := \{\}$ is mgu of $\langle t = 0 \rangle_{BR}$.
Step 2)	If Variables(t) = $\emptyset$, then stop with **fail**.
Step 3)	Select a variable $x \in$ Variables(t) with $t = ax + b$ and $x \notin$ Variables(a, b).
Step 4)	Compute the most general unifier τ of the problem $\langle ab + b = 0 \rangle_{BR}$.
Step 5)	$\sigma := \tau \circ \{x \leftarrow b + x' (1 + a)\}$ is mgu of $\langle t = 0 \rangle_{BR}$.

The algorithm is based on the following lemma.

Lemma 3.1.1:
Let x, x' be variables and a, b be terms in a Boolean ring BR and x, x' $\notin$ Variables(a, b).
$\sigma = \tau \circ \{x \leftarrow b + x' (1 + a)\}$ is mgu of $\langle ax + b = 0 \rangle_{BR}$, if τ is mgu of $\langle ab + b = 0 \rangle_{BR}$

Proof:

by induction on the number of variables occurring in a term $t = ax + b$.

1. Base Case:

$|Variables(t)| = 1$

Then $t = ax_1 + b$ with a and b variable free elements of the ring, $\sigma = \{x_1 \leftarrow b + x'_1 (1 + a)\}$ and $\tau = \{\}$.

First we prove that σ is a unifier:

$$\begin{aligned}\sigma(ax_1 + b) &=_{BR} a (b + x'_1 (1 + a)) + b \\ &=_{BR} ab + ax'_1 + ax'_1 + b \\ &=_{BR} ab + b \\ &=_{BR} \tau (ab + b)\end{aligned}$$

The chain of equations above shows, that there is a unifier γ, which satisfies $\gamma(ax_1 + b) =_{BR} 0$. Now we have to prove, that each such unifier is an instance of the most general unifier σ. If $\lambda := \{x_1 \leftarrow \gamma(x_1)$, then $\lambda\sigma =_{BR} \gamma[x_1]$:

$$\begin{aligned}\lambda\sigma(x_1) &=_{BR} b + \lambda(x'_1) * (1 + a) \\ &=_{BR} b + \gamma(x_1) * (a + 1) \\ &=_{BR} b + a\,\gamma(x_1) + \gamma(x_1) \\ &=_{BR} \gamma(ax_1 + b) + \gamma(x_1) \\ &=_{BR} \gamma(x_1)\end{aligned}$$

2. Induction Case:

Suppose the lemma is true for $|Variables(t)| = n$.

Let now be $|Variables(t)| = n + 1$.

Then term $t = t_1 x_{n+1} + t_2$ and τ is a most general unifier of $\langle t_1 t_2 + t_1 = 0\rangle_{BR}$ with $Variables(t_1) \cup Variables(t_2) = \{x_1, \ldots, x_n\}$. We will show that $\sigma = \tau \circ \{x_{n+1} \leftarrow t_2 + x'_{n+1}(1 + t_1)\}$ is a most general unifier of $\langle t = 0\rangle_{BR}$ with $Variables(t) = \{x_1, \ldots, x_n, x_{n+1}\}$.

Analogously to the base case we can proof that σ is a unifier with $\sigma(t_1 x_{n+1} + t_2) =_{BR} \tau (t_1 t_2 + t_2)$.

Let τ be most general unifier of $\langle (t_1 t_2 + t_2) = 0\rangle_{BR}$

then $\sigma(x_i) = \tau(x_i)$ for $1 \le i \le n$

$\sigma(x_{n+1}) = \tau(t_2) + (\tau(t_1) + 1)\, x'_{n+1}$ mgu of t and 0.

Again we have to prove, that every unifier γ of the unification problem is an instance of the most general unifier σ. Let γ_r and λ_r be the restrictions of γ and λ to the variables $\{x_1, \ldots, x_n\}$. Using induction hypothesis we conclude $\gamma_r =_{BR} \lambda_r\tau$. Let λ be defined as $\lambda(x_i) = \lambda_r(x_i)$ for $1 \le i \le n$ and $\lambda(x'_{n+1}) = \gamma(x_{n+1})$. As $\sigma(x_i) = \tau(x_i)$ for $1 \le i \le n$, it is sufficient to show, that γ and $\lambda\sigma$ agree on

variable x_{n+1}.

$$\begin{aligned}\lambda\sigma(x_{n+1}) &=_{BR} \lambda_r\tau(t_2) + (\lambda_r\tau(t_1) + 1)\,\lambda(x'_{n+1}) \\ &=_{BR} \lambda\tau(t_2) + (\lambda\tau(t_1) + 1)\,\gamma(x_{n+1}) \\ &=_{BR} \gamma(t_2) + (\gamma(t_1) + 1)\,\gamma(x_{n+1}) \\ &=_{BR} \gamma(t_2) + \gamma(t_1 x_{n+1}) + \gamma(x_{n+1}) \\ &=_{BR} \gamma(x_{n+1}) \quad \blacksquare\end{aligned}$$

This lemma verifies, that the algorithm computes the most general unifier. Yet we still have to prove, that the algorithm terminates. This is achieved by proving, that the only recursion (step 4) is finite. Each recursive step decrements the variables occurring in term t. Obviously the recursion has to end, as a term's variable set is finite. In fact the recursion terminates, if one of two conditions is satisfied: check 2) no more variables occur in examined term t; check 1) the term created in the recursion is equal to 0, which initiates the finite resubstitution into the solutions (step 5).

This algorithm has one non deterministic step: the selection of a variable. This selection has a decisive influence on the **number of necessary recursions** and the **termsizes of codomain and recursion step terms.** A recursion step term is a term constructed for equation to be solved in the recursion step of the algorithm (step 4). The termsize relates to storage space needed and influences calculation time, as it is more difficult to acquire information out of bigger terms. The number of recursions is equivalent to the cardinality of the unifier and thereby effects the application of the unifier. Obviously the recursions influence the execution time of the algorithm. Therefore special care is taken to determine a criterion for variable selection resolving the non determinism. As the termsize of a codomain term directly depends on the termsize of the previous recursion step term, it is sufficient to keep recursion step terms as small as possible. Terms are represented in the mentioned disjunctive normal form in the implemented version of the algorithm. The number of summands of a term in disjunctive normal form is a good measurement of its size. We choose a variable selection criterion, which keeps such terms as small as possible:

Occurrences of variable in **as few as** possible summands of term in disjunctive normal form.

Now we show, how the occurrences influence the size of the next recursion step term:
Let us use Sum(t), the number of summands of the disjunctive normal form of term t, to measure the size of a term. We assume that terms constructed in the recursion step will not be simplified. Let t, t_x, a, b be terms and x a variable of a Boolean ring. If we compute a unifier for t = 0 with t = ax + b and $x \notin$ Variables(a) $\cup$ Variables(b), then we can calculate Sum(ab + b) of recursion step term t_x= ab + b from the number of summands Sum(a) and Sum(b) of terms a and b.

$$Sum(t) = Sum(ax + b) = Sum(ax) + Sum(b) = Sum(a) + Sum(b) \quad *)$$
$$1 \leq Sum(a) \leq Sum(ax + b) \ \wedge 0 \leq Sum(b) \leq Sum(ax + b) - 1$$

$$Sum(ab + b) = Sum(a) * Sum(b) + Sum(b) \qquad | \ Sum(b) = Sum(ax + b) - Sum(a) \text{ by } *)$$
$$= - Sum^2(a) + (Sum(ax + b) - 1) * Sum(a) + Sum(ax + b)$$

The extreme can be calculated by deriving Sum(ab + b) with respect to Sum(a).
$\Rightarrow$ <u>maximum</u> at Sum(a) = (Sum(t) - 1) / 2
maximum value is $Sum_{max}(ab + b) = (Sum(t) + 1) / 2)^2 + Sum(t)$

If we have a negated quadratic function with a maximum near the middle of its domain, then the minima are at the boundaries of the domain. Thereby the recursion step terms are the smallest at the lower and upper bound. As a variable selection criterion, which has to keep terms as small as possible, we select lower bound, occurrences of a variable in as few as possible summands, because this fact can be determined faster. ■

Examining the calculation of number of summands Sum(ab + b), one has to be reminded, that is the number before simplifications. So in reality Sum_{max} is only the upper margin. Nevertheless we are still able to use the result. This has two reasons: Primarily simplification process is time consuming and storage is affected by creation of intermediate terms. Secondly simplification depends heavily on specific cases, which makes it impossible to get a good general estimate. In the following example we show, how gravely the selection of the variable influences the behaviour of the algorithm. Two different selection sequences are presented. In the first one we apply our criterion.

Example:

Let C = {a, b, c, d, e}, V = {x, y, z, x', y', z'} and t a term of the free Boolean ring BR over <C,+, *, 0, 1> generated by V. Let R be a sequence of variables and t_R a recursion step term, where R symbolizes the already executed recursion steps with the variables selected, accordingly unifier U_R and Oc(t, x) the number of occurrences of variables x in products of term t.

t = xa + yad + yae + zab + zc + zd

Oc(t, x) = 1, Oc(t, y) = 2, Oc(t, z) = 3

1. run: $R_1 = xz$

t_x = (a + 1) * (yad + yae + zab + zc + zd) | simplifies to

$=_{BR}$ zac + zad + zc + zd

Oc(t, z) = 4

t_{xz} = (ac + ad + c + d + 1) * 0 | simplifies to

$=_{BR}$ 0

U_{xz}= {x ← x' + yad + yae + z'ab + z'ac + z'ad + x'a

z ← z' + z'ac + z'ad + z'c + z'd }

2. run: $R_2 = yz$

t_y = (ad + ae + 1) * (xa + + zab + zc + zd) | simplifies to

$=_{BR}$ xa + xad + xae + zab + zabd + zabe + zacd + zace + zad + zade + zc + zd

Oc(t, x) = 3, Oc(t, z) = 9

t_{yz} = (ab + abd + abe + acd + ace + ad + ade + c + d + 1) * (xa + xad + xae) | simplifies to

$=_{BR}$ xa + xab + xabd + xabe + xac + xacd + xace + xad + xade + xae

U_{vz}= {x ← x'a + x'ab + x'abd + x'abe + x'ac + x'acd + x'ace + x'ad + x'ade + x'ae

z ← z' + x'ab + x'abd + x'abe + x'ac + x'acd + x'ace + x'ade + z'ab + z'abd+ z'abe + z'acd + z'ace + z'ad + z'ade + z'c + z'd

y ← x' + x'ad + x'ae + z'abd + z'abe + z'acd + z'ace + z'ad + z'ade + y'ad + y'ae }

This is a typical example; in the first run, where our criterion is used, we get smaller terms and even a recursion step less, which in turn generates a unifier with one domain variable less. A comparison of

the unifiers shows, that the codomain terms of the first run are less complicated. Because of those reasons the unifiers of the first case are better applicable.

If we examine the unifiers computed by this algorithm we are able to detect the following properties. Although one would assume, that unifying a term with n variables leads to a unifier cardinality of n, this is not always the case. Recursion can stop earlier, if simplification produces the trivial equation $<0 = 0>_{BR}$ or if simplification eliminates variables in a recursion step term. Therefore the **cardinality of the unifier** is less or equal to the variables in the term to be unified. The **number of variables introduced** is equal to the cardinality of the unifier; one for each domain variable. Another feature is, that we are able to detect the variable **selection sequence** of the "Variable Elimination" method in the unifier. The selected variables are the domain variables of the unifier. By Lemma 3.1.1 each domain variable has at most one related (renamed) variable in the variable set of the corresponding codomain term. This related variable does not occur in variable sets of codomain terms of later selected variables. Yet the related variable may occur in variable sets of codomain terms of earlier selected variables. Therefore often we can deduce the selection sequence from the variable sets of codomain terms. Regarding the variable sets of codomain terms we can say, that those sets belonging to earlier selected variables are often supersets of variable sets of codomain terms of later selected variables. So the codomain variables are not totally mixed, which is profitable for application of a unifier.

Until now we have regarded the cases, where there is a unifier for the problem. What is, if there is none? Then this algorithm will select one variable after another and create the corresponding recursion terms. At the end we will have a term without variables, which is not equal to 0, and stop with failure. So in this case we would have wasted a lot of computation time. Therefore, if it is likely that terms are not unifiable, we would like to apply a unifiability test first. However is there such a test? The answer is yes, such a test was developed by G.Boole in 1847 [Boo 47]. This test is incorporated in one of the following Boolean ring unification algorithm.

3.2 Method of Deducing from a Particular Solution

U. Martin and T. Nipkov [MN 86] proposed this idea in 1986 for application in set theory and in propositional calculus. The main idea of this algorithm is to find a particular solution for an equation $\langle t_1 = t_2 \rangle_{BR}$ and to generate the general solution from this particular solution. The first one to give an account of this method was L. Löwenheim [Löw 08]. He presented the following theorem:

Theorem 3.2.1:
Let t_1 and t_2 be two terms of free Boolean ring BR over the Boolean ring B and let $b_1, \ldots, b_n$ be elements of B and Variables(t_1) $\cup$ Variables(t_2) = $\{x_1, \ldots, x_n\}$ the used variables.
When $\sigma t_1 =_{BR} \sigma t_2$ with $\sigma = \{x_i \leftarrow b_i \mid 1 \le i \le n\}$
then the substitution

$$\tau := \{x_i \leftarrow x'_i + \gamma(t_1 + t_2) * (x'_i + b_i) \mid 1 \le i \le n\} \qquad \text{with } \gamma = \{x_i \leftarrow x'_i \mid 1 \le i \le n\}$$

is a most general unifier of t_1 and t_2.

Example:
Let $t_1 = ax + by$ and $t_2 = a$. A solution is $\sigma = \{x \leftarrow 1, y \leftarrow 0\}$.

$$\begin{aligned} \text{Then } \tau &= \{x \leftarrow x' + (ax' + by' + a)(x' + 1), \quad y \leftarrow y' + (ax' + by' + a)(y' + 0)\} \\ &=_{BR} \{x \leftarrow x' + bx'y' + by' + ax' + a, \quad y \leftarrow y' + ax'y' + by' + ay'\} \end{aligned}$$

is a most general unifier of t_1 and t_2.

Proof of Theorem 3.2.1:
1.step: $\tau = \{x_i \leftarrow x'_i + \gamma(t_1 + t_2) * (x'_i + b_i) \mid 1 \le i \le n\}$ with $\gamma = \{x_i \leftarrow x'_i \mid 1 \le i \le n\}$is a unifier of t_1 and t_2, if $\sigma = \{x_i \leftarrow b_i \mid 1 \le i \le n\}$ is a unifier.

We can split $(t_1 + t_2)$ into the homogeneous part t_{hom} and the variable free part c.
Then we can write $t_1 + t_2 =_{BR} t_{hom} + c$.
$\sigma t_1 =_{BR} \sigma t_2 \Leftrightarrow \sigma t_{hom} + c =_{BR} 0 \Leftrightarrow \sigma t_{hom} =_{BR} c$

$$\begin{aligned} \tau t_1 + \tau t_2 &=_{BR} \tau t_{hom} + c =_{BR} \{x_i \leftarrow x'_i + \gamma(t_{hom} + c) * (x'_i + \sigma x_i) \mid 1 \le i \le n\} t_{hom} + c \\ &=_{BR} \{x_i \leftarrow (1 + \gamma t_{hom} + c)x'_i + (\gamma t_{hom} + c)\sigma x_i \mid 1 \le i \le n\} t_{hom} + c \\ &=_{BR} \{x_i \leftarrow \gamma((1 + t_{hom} + c)x_i + (t_{hom} + c)\sigma x_i) \mid 1 \le i \le n\} t_{hom} + c \\ &=_{BR} \gamma\{x_i \leftarrow ((1 + t_{hom} + c)x_i + (t_{hom} + c)\sigma x_i) \mid 1 \le i \le n\} t_{hom} + c \\ &=_{BR} \gamma(t_{hom}(1 + t_{hom} + c) + (t_{hom} + c)\,\sigma t_{hom}) + c \end{aligned}$$

$=_{BR} \gamma(t_{hom}(1 + t_{hom} + c) + (t_{hom} + c)\, c) + c$
$=_{BR} \gamma(ct_{hom} + ct_{hom} + c) + c$
$=_{BR} 0$ ■

2. step: Now suppose there exits a δ such that $\delta t_1 =_{BR} \delta t_2$. Now we have to prove, that it is an instance of the most general unifier τ. Let $\lambda x'_i = \delta x_i$ for $1 \le i \le n$.

$\lambda\tau =_{BR} \lambda\{x_i \leftarrow x'_i + \gamma(t_1 + t_2) * (x'_i + b_i) \mid 1 \le i \le n\}$
$=_{BR} \{x_i \leftarrow \lambda x'_i + \lambda\gamma(t_1 + t_2) * (\lambda x'_i + b_i) \mid 1 \le i \le n\}$
$=_{BR} \{x_i \leftarrow \delta x_i + \delta(t_1 + t_2) * (\delta x_i + b_i) \mid 1 \le i \le n\}$
$=_{BR} \{x_i \leftarrow \delta x_i + 0 * (\delta x_i + b_i) \mid 1 \le i \le n\}$
$=_{BR} \{x_i \leftarrow \delta x_i \mid 1 \le i \le n\}$
$=_{BR} \delta\,[x_1, \ldots, x_n]$ ■

This theorem allows to compute the general solution from a particular one. Several methods for determination of a particular solution are known. I have implemented two of these methods, which will be presented in the following paragraphs. For other methods consult S. Rudeanu's book on boolean functions and equations [Rud 74]. Both implemented versions use the polynomial form of a term. The first one looks for a particular solution in a subring generated by the coefficients of the term, the other one in a subring generated by the constants occurring in the term.

To explain those methods we recall some definitions and a theorem and present two lemmata:

Definitions:

Let B be a Boolean ring.

A subset $D = \{d_1, \ldots, d_n\}$ of B with $d_i \neq 0$ is called a **basis** for B, if

1) each element $b \in B$ can be expressed as linear combination of elements of D:

$$b = \sum_{1 \le i \le n} b_i d_i \qquad \text{with } b_i \in \{0, 1\}$$

2) the elements of D are independent:

$$0 = \sum_{1 \le i \le n} b_i d_i \Leftrightarrow \forall_{1 \le i \le n}\, b_i = 0.$$

The elements of D are called **orthogonal**, if and only if

$$d_i d_j = 0 \text{ for } i \neq j.$$

So a subset $D = \{d_1, \ldots, d_n\}$ of B is an **orthogonal basis**, if D is a basis for B and its element are orthogonal. Let us call the d_i the **basisvectors** of B.

Lemma 3.2.2: If the elements of a set are orthogonal, then they are also independent.

Proof:

Let $D = \{d_1, \ldots, d_n\}$ be orthogonal.

$0 = \Sigma_i b_i d_i \Leftrightarrow b_j d_j = \Sigma_{i \neq j} b_i d_i$ | * $b_j d_j$

$\Leftrightarrow b_j d_j = b_j d_j \Sigma_{i \neq j} b_i d_i \Leftrightarrow b_j d_j = \Sigma_{i \neq j} b_j b_i d_j d_i$ | orthogonality

$\Leftrightarrow b_j d_j = 0$ | $d_j \neq 0$

$\Leftrightarrow b_j = 0$ ■

Theorem 3.2.3:

Let B be a Boolean Ring <C, +, *, 0, 1> with an arbitrary set C and for all $G \subseteq C$ let

$$d_G = (\Pi_{g \in G} g)\ (\Pi_{h \in C \setminus G} (h + 1))$$

Then the non-zero d_G are pairwise distinct and form a unique orthogonal basis of B.

Proof:

Experts can deduce this theorem from standard results about semisimple Artinian rings - see [Hers 68]. We present a direct proof:

a) orthogonality:

$d_G * d_G = d_G \neq 0$ and, if $G \neq H$, then $d_G * d_H = 0$, as there is at least one element $c \in C$, which contributes c to one vector and c + 1 to the other one.

b) linear independency can be deduced from orthogonality by Lemma 3.2.2.

c) distinctiveness:

Let $G \neq H$ and assume $0 \neq d_G = d_H$. Then $d_G * d_G = d_G * d_H$. By orthogonality $d_G = 0$. ↯

d) every element $b \in B$ is a linear combination of basis D:

Let $b = \Sigma_{U \subseteq C} b_U c_U$ with $b_U \in \{0, 1\}$ and $c_U = \Pi_{c \in U} c$.

We have $c_U = c_U * 1 =^{d1)} c_U \Sigma_{W \subseteq C} d_W =^{d2)} \Sigma_{W \subseteq U} d_W$ as linear combination of D.

Thus b is a linear combination of D.

d1) $1 = \Sigma_{W \subseteq C} d_W$

Proof by induction: If $C = \{c\}$, then $1 = c + (c + 1)$. Now let $C' = C \cup \{c\}$ with non emtpy C.

Then $1 = \Sigma_{W \subseteq C} d_W = ((1 + c) + c)\ \Sigma_{W \subseteq C} d_W = \Sigma_{W \subseteq C'} d'_W$.

d2) $c_U d_W = d_W$, if $U \subseteq W$ and 0 otherwise

If one element c of c_U is not in w, then c + 1 appears in d_W making $c_U d_W = 0$. Otherwise all c of c_U appear as c in d_W making $c_U d_W = d_W$.

e) uniqueness:

We have to prove, if P is another orthogonal Basis then P = D.

Suppose $p \in P$. Then $p = 1 * p = \Sigma_{W \subseteq C} pd_W$. There has to be at least one subset $W \subseteq C$ with $pd_W \neq 0$. As D is orthogonal basis, $pd_W =^{e1)} d_W$ and as P is one, $pd_W = p$. Thus $p \in D$ and $P \subseteq D$. By applying the above argument interchanged, we have $D \subseteq P$. and thereby P = D.

e1) If $p = \Sigma_{W \subseteq C} b_W d_W$ with $b_W \in \{0, 1\}$, then for $U \subseteq C$ $pd_U = b_U d_U \in \{0, d_U\}$

Multiply the equation with d_U, we have $pd_U = d_U \Sigma_{W \subseteq C} b_W d_W = b_U d_U$, which is equal to 0, if $b_U = 0$ and equal to d_U otherwise.

■

Lemma 3.2.4:

Let B be a Boolean Ring <C, +, *, 0, 1> with a set C of free constants and for all $G \subseteq C$ let

$$d_G = (\Pi_{g \in G} g) (\Pi_{h \in C \setminus G} (h + 1))$$

Then the d_G are pairwise distinct and form a unique orthogonal basis of B.

Proof:

This lemma can be deduced from Theorem 3.2.3. We just have to prove, that there is no vector $d_{G'}$, which is equal to 0.

Let b be element of boolean ring B. As neither $c * b \in C$ nor (c + 1)b can be expressed by the generating set C without element c, no element $d_G = (c + 1) (\Pi_{g \in G, g \neq c} g) (\Pi_{h \in C \setminus G, h \neq c} (h + 1))$ and no element $d_G = c (\Pi_{g \in G, g \neq c} g) (\Pi_{h \in C \setminus G, h \neq c} (h + 1))$ can be equal to 0. ■

Examples:

a) Theorem 3.2.3

Let C = {a, b, a +b}

$d_1 = a * b * (a + b) = 0$

$d_2 = a * b * (a + b + 1) = ab$

$d_3 = (a + 1) * b * (a + b) = (a + 1) b$

$d_4 = (a + 1) * b * (a + b + 1) = 0$

$d_5 = a * (b + 1) * (a + b) = a (b + 1)$

$d_6 = a * (b + 1) * (a + b + 1) = 0$

$d_7 = (a + 1) * (b + 1) * (a + b) = 0$

$d_8 = (a + 1) * (b + 1) * (a + b + 1) = (a + 1) * (b + 1)$

Then basis is set $D = \{d_2, d_3, d_5, d_8\}$.

b) Lemma 3.2.4

Let C = {a, c}

Then basis is set D = {ac, a (c +1), (a +1) c, (a +1) (c +1)}.

Now we present two methods to determine particular solutions. We recall an approach suggested by U. Martin and T. Nipkov [MN 86], which is based on Theorem 3.2.3, and present our own approach, which relies on Lemma 3.2.4. Even though the lemma is derived from the theorem, technically Martin/Nipkov have to execute an additional test (non-zero condition). Therefore we begin with a detailed explanation of our approach in Chapter 3.2.1 and then recall the Martin/Nipkov method in Chapter 3.2.2 as a modification.

3.2.1 Particular Solution in a Ring Generated by Constants

The first approach to find a particular solution for the equation $\langle t = 0 \rangle_{BR}$, is to look for solutions in the Boolean ring BR. Yet surely one can restrict the search space to a ring, which subjects to our specific equation $\langle t = 0 \rangle_{BR}$. This would be a free Boolean ring B over the ring A = <C, +, *, 0, 1> generated by V, where C = Constants(t) and V = Variables(t). Nevertheless the method presented here restricts the search space even further. We look for a solution in Boolean ring A. This is derived from the fact, that the existence of a most general unifier implies the existence of a **ground** unifier. In a ground unifier all codomain terms are ground, i.e. the terms contain only constants. The proof for this fact is obvious, as every variable in the most general unifier can be instantiated by a constant.

The computation of a particular solution σ of $\langle t = 0 \rangle_{BR}$ is equivalent to solve $\langle 0 = \sigma t \rangle_{BR}$.We will determine another equivalent set of equations and present an approach to solve them:

Let t be a term in the polynomial form $t = \Sigma_{U \subseteq V} c_U v_U$, where $v_U = \Pi_{x \in U} x$, $x \in$ Variables(t) and $c_U = \Sigma_{i \in 1..m} \Pi_{j \in 1..n} a_{ij}$, $a_{ij} \in$ Constants(t) and let A be a Boolean ring <C, +, *, 0, 1> with C = Constants(t). By Lemma 3.2.4 set $D = \{d_G = (\Pi_{g \in G} g)\ (\Pi_{h \in C \setminus G} (h + 1)) \mid G \subseteq C\}$ is the orthogonal basis with $2^{|C|}$ elements. Because every ground solution σ can be represented by the basisvectors of the orthogonal basis, $\sigma = \{x \leftarrow \Sigma_{G \subseteq C} x_G d_G \mid x \in \text{Variables(t)}\}$ with $d_G \in D$ and $x_G \in \{0,1\}$.

Let σ be a ground unifier of t and 0. Then we have the following equation chain.

$$
\begin{aligned}
0 &= \sigma t = \sigma \Sigma_{U \subseteq V} c_U v_U \\
&= \Sigma_{U \subseteq V} c_U \Pi_{x \in U} \sigma x && \mid x \leftarrow \Sigma_{G \subseteq C} x_G d_G \text{ (represented by basisvectors)} \\
&= \Sigma_{U \subseteq V} c_U \Pi_{x \in U} (\Sigma_{G \subseteq C} x_G d_G) && \mid \text{the orthogonality of the basisvectors implies:} \\
& && \mid d_G * d_H = 0, \text{ if } G \neq H \text{ and} \\
& && \mid d_G * d_H = d_G, \text{ if } G = H
\end{aligned}
$$

$= \Sigma_{U\subseteq V} c_U (\Sigma_{G\subseteq C} d_G \prod_{x\in U} x_G)$ | integrating c_U into each sum
$= \Sigma_{U\subseteq V} \Sigma_{G\subseteq C} c_U d_G \prod_{x\in U} x_G$
$= \Sigma_{G\subseteq C} \Sigma_{U\subseteq V} c_U d_G \prod_{x\in U} x_G$

As the basisvectors "d_G" are defined to be independent, we can split the equation into $2^{|C|}$ equivalent equations:

$\forall i \subseteq C \quad 0 = \Sigma_{U\subseteq V} c_U d_i \prod_{x\in U} x_i$

The term is divided into an inhomogeneous and a homogeneous part. Hence

$\forall i \subseteq C \quad 0 = c_\emptyset d_i + \Sigma_{U\subseteq V, U\neq\emptyset} c_U d_i \prod_{x\in U} x_i$ or equivalently

$\forall i \subseteq C \quad c_\emptyset d_i = \Sigma_{U\subseteq V, U\neq\emptyset} c_U d_i \prod_{x\in U} x_i$

Every c_i are expressed by the basisvectors of the orthogonal basis:

$c_i \leftarrow \Sigma_{G\subseteq C} y_{iG} d_G,\ y_{iG} \in \{0,1\}$

Hence

$\forall i \subseteq C \quad (\Sigma_{G\subseteq C} y_{\emptyset G} d_G)\ d_i = \Sigma_{U\subseteq V, U\neq\emptyset} (\Sigma_{G\subseteq C} y_{UG} d_G)\ d_i \prod_{x\in U} x_i$

The orthogonality of the basisvectors implies that

$d_G * d_H = 0$, if $G \neq H$ and

$d_G * d_H = d_G$, if $G = H$

Applying these equations we get

$\forall i \subseteq C \quad y_{\emptyset i} d_i = \Sigma_{U\subseteq V, U\neq\emptyset} y_{Ui} d_i \prod_{x\in U} x_i$

Division by d_i implies

$\forall i \subseteq C \quad y_{\emptyset i} = \Sigma_{U\subseteq V, U\neq\emptyset} y_{Ui} \prod_{x\in U} x_i$

Three different conditions can occur, if we try to solve these equations.

c1) $y_{\emptyset i} = 0 \quad \Rightarrow x_i = 0$ for all $x \in V$.

c2) $y_{\emptyset i} = 1$, all $y_{Ui} = 0 \quad \Rightarrow$ there is no solution, as $1 = \Sigma_{U\subseteq V, U\neq\emptyset} 0$ has no solution.

c3) $y_{\emptyset i} = 1$, some $y_{Ui} = 1 \quad \Rightarrow$ there is always a solution

From the $\prod_{x\in U} x_i$ with $y_{Ui} = 1$ select the one with the smallest set of variables $S(U') \subseteq V$. For all variables x' of S(U'), set x'_i to 1 and the others x" of V\S(U'), set x''_i to 0. This leads that all other $\prod_{x\in U} x_i$ with $y_{Ui} = 1$ evalute to 0, as they contain a member x''_i.

If there always has been a solution, we create the ground unifier σ from the independently computed results.

$\sigma = \{x \leftarrow \Sigma_{i\subseteq C} x_i d_i \mid x \in V\}$ ■

Example:

Let t = ax + by + a.

Then our Boolean ring A is <C, +, *, 0, 1> with C = {a, b}and the orthogonal basis is the set D = $\{d_1, d_2, d_3, d_4\}$ with d_1 = ab, d_2 = a (b +1), d_3 = (a +1) b and d_4 = (a +1) (b +1)}. Then x can be expressed as $x = x_1d_1 + x_2d_2 + x_3d_3 + x_4d_4$ and the elements y and a acccordingly. Note that for simplification purposes, we did not choose the set indices of x and d as above.

$0 = ax + by + a$

$= ax_1d_1 + ax_2d_2 + ax_3d_3 + ax_4d_4 + by_1d_1 + by_2d_2 + by_3d_3 + by_4d_4 + ad_1 + ad_2 + ad_3 + ad_4$

Now the independency of the basisvectors d_i allows us to split the equation in four other ones:

<=> $\forall_i\ 1 \le i \le 4$

$0 = ax_id_i + by_id_i + ad_i$ | $a = ab + ab + a = d_1 + d_2$

| $b = ab + b + ab = d_3 + d_4$

<=> $\forall_i\ 1 \le i \le 4$

$0 = (d_1 + d_2)x_id_i + (d_3 + d_4)y_id_i + (d_1 + d_2)d_i$ | orthogonality of d_i

$$
\begin{array}{lllll}
 & 1 * d_1 = 1 * x_1d_1 + 0 * y_1d_1 & 1 = 1 * x_1 + 0 * y_1 & 1 = x_1 & 0 = y_1 \\
<=> & 1 * d_2 = 1 * x_2d_2 + 0 * y_2d_2 & <=> \ 1 = 1 * x_2 + 0 * y_2 & <=> \ 1 = x_2 & 0 = y_2 \\
 & 0 * d_3 = 0 * x_3d_3 + 1 * y_3d_3 & 0 = 0 * x_3 + 1 * y_3 & 0 = x_3 & 0 = y_3 \\
 & 0 * d_4 = 0 * x_4d_4 + 1 * y_4d_4 & 0 = 0 * x_4 + 1 * y_4 & 0 = x_4 & 0 = y_4
\end{array}
$$

<=> $x = ab + a\,(b + 1) = ab + ab + a = a$

$y = 0$

The special solution is

$\sigma = \{x \leftarrow a, y \leftarrow 0\}$.

The most general unifier is

$\tau = \{x \leftarrow x' + (ax' + by' + a)(x' + a)$

$y \leftarrow y' + (ax' + by' + a)(y' + 0)\}$

$=_{BR} \{x \leftarrow x' + ax' + bx'y' + aby' + a$

$y \leftarrow y' + ax'y' + by' + ay'\}$

Above we have outlined an algorithm, which computes a ground unifier σ. The following facts were used to improve the implemented version of the algorithm:

1) If we collect the independent solutions $x \leftarrow \Sigma_{i \in C}\, x_id_i$ for each variables, we just have to consider such presolutions x_i with value 1. The ones with value 0 do not contribute to the solution, as

multiplying a basisvector with 0 is equal to 0. Now, if the inhomogeneous part of the term t is expressed by basisvectors $d_{i'}$, equations of basisvectors d_j with $j \neq i'$ (case c1) are not needed, as the x_j are equal to 0 for every $x \in V$. Thereby we create only the subset of the orthogonal basis, which is necessary to express the inhomogeneous part.

2) After the construction of the polynomial form of term t the homogeneous part is sorted in a fashion, which is optimal for case c3, where the independant parts of a particular solution are computed. This means, that the $c_{U'}v_{U'}$ with a smaller variable set $v_{U'}$ are in front of the term-argumentlist. Under such circumstances the first $c_{U'}v_{U'}$, which satisfies any condition, is always the one with smallest variable set of all satisfying $c_{U'}v_{U'}$.

Now we give a detailed account of the whole algorithm with our modifications.

Algorithm: **"Basis of Constants" Method**

Input: term $t \in BR$, a free Boolean ring over Boolean ring $A = \langle C, +, *, 0, 1\rangle$ generated by V such that V = Variables(t) and C = Constants(t).

Output: BR-unifier σ of term t and 0.

Step 1) Convert t into the polynomial form

$t = \Sigma_{U \subseteq V} c_U v_U$.

Step 2) Split t into homogeneous and inhomogeneous part.

$t_{hom} = \Sigma_{U \subseteq V,\, U \neq \emptyset} c_U v_U$.

$t_{inhom} = c_\emptyset$

Step 3) Sort t_{hom} with the criterion "smallest" v_U sets.

Step 4) Compute the part D' of the orthogonal basis D of initial Boolean ring A, which is needed to represent $c_\emptyset$.

$D' = \{d_G = (\Pi_{g \in G} g)\,(\Pi_{h \in C \setminus G} (h + 1)) \mid G \subseteq C \wedge c_\emptyset d_G \neq 0\}$

Step 5) For all $d \in D'$, if set $\{U \mid c_U d \neq_{BR} 0\}$ empty, then **fail**

else let N(d) = first element of $\{U \mid c_U d \neq_{BR} 0\}$ in order of t_{hom}.

Ground unifier σ consists of elements

$\{x \leftarrow \Sigma\{d \mid d \in D', x \in N(d)\}$ for each $x \in V$.

Example:

Let $t = ayx + ax + byx + a$.

1) $t = (a + b)yx + ax + a$
2) $t_{hom} = (a + b)yx + ax$
   $t_{inhom} = a$
3) $t_{hom} = ax + (a + b)yx$
4) $D' = \{ab, a(b + 1)\}$
5) $N(ab) = \{x\}$, as $ab * a =_{BR} ab \neq_{BR} 0$
   $N(a(b + 1)) = \{x\}$, as $a(b + 1) * a =_{BR} a(b + 1) \neq_{BR} 0$
   $\sigma = \{x \leftarrow ab + a(b + 1) =_{BR} a, y \leftarrow 0\}$

3.2.2 Particular Solution in a Ring Generated by Coefficients

This algorithm, which was presented earlier by Martin/Nipkov [MN 86], is similar to the one above. We also look for a particular solution to equation $t =_{BR} 0$ in a Boolean ring $A <C, +, *, 0, 1>$. The difference is that C is not the set of constants of term t, but the coefficients of term t in polynomial form $\Sigma_{U \subseteq V} c_U v_U$. Now we have to decide, if we can apply Lemma 3.2.4 as in the previous algorithm or if we have to use Theorem 3.2.3. To apply the lemma the coefficients have to be free constants. Because this is not necessarily true, we have to base our algorithm on the theorem. Thereby the orthogonal basis is set $D = \{d_G = (\Pi_{g \in G} g)(\Pi_{h \in K \setminus G}(h + 1)) \mid G \subseteq K = \{c_U \mid c_U \in \text{term}\} \wedge d_G \neq 0\}$. We may still use the algorithm sketch of the previous chapter, if we extend the computation of the basisvectors with the test mentioned above. The implemented version of the algorithm is upgraded by some improvements. For explanation let us recapture the final part of the algorithm, where we solve these independent equations:

$$\forall i \subseteq C \quad y_{\emptyset i} d_i = \Sigma_{U \subseteq V, U \neq \emptyset} y_{Ui} d_i \Pi_{x \in U} x_i$$
$$\Leftrightarrow \forall i \subseteq C \quad y_{\emptyset i} = \Sigma_{U \subseteq V, U \neq \emptyset} y_{Ui} \Pi_{x \in U} x_i$$

We know, that three different conditions can occur while solving these equations.

c1) $y_{\emptyset i} = 0 \Rightarrow x_i = 0$ for all $x \in V$.
c2) $y_{\emptyset i} = 1$, all $y_{Ui} = 0 \Rightarrow$ there is no solution, as $1 = \Sigma_{U \subseteq V, U \neq \emptyset} 0$ has no solution.
c3) $y_{\emptyset i} = 1$, some $y_{Ui} = 1 \Rightarrow$ there is always a solution

As we use basisvectors based on the coefficients, we are able to relate each case to a set of basisvectors.

c1) $D_1 = \{d_G = \{(c_\emptyset + 1)\ (\prod_{g \in G} g)\ (\prod_{h \in C \setminus G} (h + 1)) \mid G \subseteq C' = C \setminus \{c_\emptyset\}\}$

c2) $D_2 = \{d_G = \{c_\emptyset \prod_{h \in C'} (h + 1) \wedge C' = C \setminus \{c_\emptyset\}\}$

c3) $D_3 = \{d_G = \{c_\emptyset\ (\prod_{g \in G} g)\ (\prod_{h \in C \setminus G} (h + 1)) \mid G \subseteq C' = C \setminus \{c_\emptyset\}\}$

We already know, that each of the cases fulfils a different task in the algorithm. The equations of case c1 do not contribute to the solutions, the ones of case c2 conclude non-unifiability and only the last set determines the ground unifier. This is the foundation for several improvements:

1) Regarding case c2, we discover, that the set D_2 contains only one element. This makes it extremely suitable for a unifiability test. In fact this test has been discovered already by Boole [Boo 47] and is known as **Boole's test**:

Let term $t = \Sigma_{U \subseteq V} c_U v_U$ be element of Boolean ring BR.

If $c_\emptyset * \prod_{U \subseteq V,\ U \neq \emptyset} (c_U + 1) =_{BR} 0$, then the t and 0 are unifiable.

This test is integrated into the procedure, which determines the basisvectors. Thereby non-unifiability is detected before the time consuming task of solution collection.

2) Just like in the other implementation, we do not create equations of case c1. Accordingly just subset $D' = D_2 \cup D_3$ of the orthogonal basis is constructed. This improvement can be enhanced. In case c3 we check the summands $c_U v_U$ of the homogeneous term part to obtain the "smallest" v_U with $c_U d_i \neq 0$. If we use the simplified basisvectors d_i, we really would have to execute all the multiplications. However the d_i are products of $ç_U$, which are inverted or not inverted c_U. We know that by orthogonality $c_U \cdot d_i$ is only then not theory-equal to 0, if basisvector d_i is a product with multiplier c_U in non-inverted form. Our idea is to store the basisvector together with the "smallest" v_U part of a non-inverted c_U of the basisvector. Thereby a direct link is established between the equations to be solved and the v_U parts necessary for the collection of the solutions.

To enable a fast detection of the "smallest" v_U, the homogeneous part is sorted in that fashion in advance.

Algorithm: **"Basis of Coefficients" Method**

Input: term $t \in BR$, a free Boolean ring over Boolean ring $A = \langle C, +, *, 0, 1\rangle$ generated by V such that V = Variables(t) and C = Coefficients(t).

Output: BR-unifier σ of term t and 0.

Step 1) Convert t into the polynomial form

$t = \Sigma_{U \subseteq V} c_U v_U$. | C is the set of c_U.

Step 2) Split t into homogeneous and inhomogeneous part.

$t_{hom} = \Sigma_{U \subseteq V,\, U \neq \emptyset} c_U v_U$.

$t_{inhom} = c_\emptyset$

Step 3) Sort t_{hom} with the criterion "smallest" v_U sets.

Step 4 a) Apply Boole's test

If $c_\emptyset * \prod_{U \subseteq V,\, U \neq \emptyset} (c_U + 1) \neq_{BR} 0$ then **fail**.

b) Compute the part D' of the orthogonal basis D of initial Boolean ring A, which is needed to represent $c_\emptyset$

$D' = \{d_G = c_\emptyset (\prod_{g \in G} g) (\prod_{h \in C \setminus G} (h + 1)) \mid G \subseteq C' = C \setminus \{c_\emptyset\} \wedge d_G \neq_{BR} 0\}$

c) Let $H = \{(d_G, v_g) \mid d_G \in D'\}$ with g = first(G) in order of t_{hom} and gv_g a summand in t_{hom}.

Step 5) Ground unifier σ consists of elements

$\{x \leftarrow \Sigma\{d_G \mid x \in v_g, (d_G, v_g) \in H\}$ for each $x \in V$.

Example:

Let $t = ayx + ax + byx + a$.

1) $t = (a + b)yx + ax + a$

2) $t_{hom} = (a + b)yx + ax$ $\quad t_{inhom} = a$

3) $t_{hom} = ax + (a + b)yx$

4) a) $a * (a + 1) * (a + b + 1) =_{BR} 0 \Rightarrow$ unifiable

b) $d_{a(a+b)} = a * a * (a + b) =_{BR} a(b + 1)$

$d_a = a * a * (a + b + 1) =_{BR} ab$

$d_{(a+b)} = a * (a + 1) * (a + b) =_{BR} 0$

$d_\emptyset = a * (a + 1) * (a + b + 1) =_{BR} 0$

$D' = \{d_{a(a+b)}, d_a\}$

c) $H = \{< d_{a(a+b)}, \{x\}>, < d_a, \{x\}>\}$

5) $\sigma = \{x \leftarrow a(b + 1) + ab =_{BR} a, y \leftarrow 0\}$

3.3 Some Additional Constraints

When we work with those three different Boolean ring unification algorithms, we discover that they can be improved. Our work in that area can be splitted in three fields: detection of non-unifiability, direct generation of a general solution and direct generation of a special solution.

For the detection of **non-unifiability** the first idea is to modify Boole's test [Boo 47]. However this test is quite time expensive and expects the terms to be in polynomial form. In case of the recursive algorithm this means the term to be tested would have to be converted from disjunctive into polynomial form. Therefore another much faster test is used. This test is able to detect many non-unifiability cases. It originates in the recursive algorithm. The non-unifiability of term t with 0 is tested in two steps. First we check for $t \neq_{BR} 0$ and then we detect non-unifiability, if we find no variables in term t. As we use simplifiers for terms, the first examination is really just a syntax check. The variable check, on the other hand, is supported by the theorem proving environment HADES [Ohl 88]. Consequently this check was introduced to the iterative algorithms. Note, that a fail of this test does not imply unifiability of the term with 0.

There are two criterions, which allow an immediate calculation of a **general unifier.**

1. Let x be a variable, $t = t_1 + x$ be a term and $x \notin$ Variables(t_1).
   Then $\tau := \{x \leftarrow t_1\}$ is the most general unifier of t and 0.
2. Let x be a variable, $t = x * t_1$ be a term and $x \notin$ Variables(t_1).
   Then $\tau := \{x \leftarrow x' + x't_1)\}$ is the most general unifier of t and 0.

Proof:

1. As $\langle t_1 + x = 0\rangle_{BR}$ is equivalent to $\langle t_1 = x\rangle_{BR}$, we can recall a proof which is valid for every theory E [BHS 87].

Let mgu be $\tau = \{x \leftarrow t_1\}$ and σ be any unifier of x and t_1, where $x \notin$ Variables(t_1). Then $\sigma =_E \sigma\tau$ [Variables(x,t_1)], as $\sigma\tau x =_E \sigma t_1 =_E \sigma x$ and $\sigma\tau y =_E \sigma y$ for $y \neq x$. ■

2. Let $\tau = \{x \leftarrow x' + x't_1)$ be mgu and let σ any unifier for the equation $\langle t_1 * x = 0\rangle_{BR}$. Then $\sigma =_{BR} \lambda\tau$ [Variables(x,t_1)] with $\lambda = \sigma \circ \{x' \leftarrow x\}$, as

$\lambda\tau x = \sigma \circ \{x' \leftarrow x\}(x' + x't_1) =_{BR} \sigma(x + xt_1) =_{BR} \sigma x + \sigma x t_1 =_{BR} \sigma x$ and

$\lambda\tau y =_{BR} \lambda y =_{BR} \sigma y$ for $y \neq x$. ■

Those checks are implemented in all three algorithms. In the recursive algorithm these checks are also performed during each recursion step. The specific place in the algorithm is just before variable selection. Obviously a second possibility would have been to use the checks for improvement of the variable selection criterion. However for our application we choose to improve the time factor rather than to stick to the modularity concept.

We found two criterions, which allow the computation of a **special unifier.**

1. Let $t = \Sigma_i t_i$ be a term in disjunctive normal form, such that term t is homogeneous, i.e. $Variables(t_i) \neq \emptyset$ for every i.
   Then $\sigma := \{x \leftarrow 0 \mid x \in Variables(t)\}$ is a unifier of t and 0.
2. Let t be in the polynomial form $\Sigma_{U \subseteq V} c_U v_U$. If there is a subterm $c_{U'} v_{U'}$, which consist only of variables and there is no other subterm with a variable set, which is a subset of variable set $v_{U'}$, then we can select variable $x' \in Variables(v_{U'})$ and
   $\sigma := \{x \leftarrow c_\emptyset \mid x = x'$
   $x \leftarrow 1 \mid x \in Variables(v_{U'}) \setminus \{x'\}$,
   $x \leftarrow 0 \mid x \in Variables(t) \setminus Variables(v_{U'})\}$ is a unifier of t and 0.
   Example: $t = xy + (a + b)\,xz + bvz + z + a + e$
   $\sigma = \{x \leftarrow a + e, y \leftarrow 1, z \leftarrow 0, v \leftarrow 0\}$

Proof:

The methods for the proof is application of the unifier:

1. $\sigma t = \sigma \Sigma_{i \in 1..n} t_i$
   $=_{BR} \Sigma_{i \in 1..n} \sigma t_i$ | as $Variables(t_i) \subseteq Variables(t)$
   $=_{BR} \Sigma_{i \in 1..n} 0_i$
   $=_{BR} 0$ ■
2. $\sigma t = \sigma \Sigma_{U \subseteq V} c_U v_U$
   $=_{BR} \Sigma_{U \subseteq V} \sigma c_U v_U$
   $=_{BR} \sigma c_\emptyset v_\emptyset + \sigma c_{U'} v_{U'} + \Sigma_{U \subseteq V, U \neq \emptyset, U \neq U'} \sigma c_U v_U$ | $c_{U'} = 1$
   $=_{BR} \sigma c_\emptyset + \sigma v_{U'} + \Sigma_{U \subseteq V, U \neq \emptyset, U \neq U'} \sigma c_U v_U$ | application of unifier σ
   $=_{BR} c_\emptyset + c_\emptyset * \Pi_{x \in U \setminus \{x'\}} 1 + \Sigma_{U \subseteq V, U \neq \emptyset, U \neq U'} 0$
   $=_{BR} c_\emptyset + c_\emptyset$
   $=_{BR} 0$ ■

These criterions are not applied in the recursive algorithm. Even though one would be able to generate the most general unifier using Löwenheim's theorem [Löw 08]. This has three reasons: First the second criterion is very specific and second this criterion is based on the polynomial form. This information is not directly available, as the "Variable Elimination" method uses the disjunctive normal form. Yet most importantly it is valid for both criterions, that computation in this fashion is not necessarily superior to one by the recursive algorithm. A detailed comparison of the methods is given in the next Chapter 3.4.

3.4 Comparison of the Methods

Comparing the three algorithms the first observation is that there really are only two different approaches as suggested by the paragraphing of this thesis. The main idea of the "Variable Elimination" method is a consecutively narrowing of the solution space. We will refer to this approach as method 1. The other two use properties of the Boolean ring theory, which allow the construction of a most general unifier from a particular solution. We first compare the two approaches of the particular solution method.

Let us distinguish the differences with a more complex example.

$t = bc + ab + ax + x + bxy + ay + acy$

i) "Basis of Constants" method (method 2)

1) $t = (bc + ab) + (a + 1)x + bxy + (a + ac)y$

2) $t_{hom} = (a + 1)x + bxy + (a + ac)y \qquad t_{inhom} = (bc + ab)$

3) $t_{hom} = (a + 1)x + (a + ac)y + bxy$

4) $t_{inhom} = abc + (a + 1)bc + abc + ab(c + 1) =_{BR} (a + 1)bc + ab(c + 1)$
$\Rightarrow D' = \{(a + 1)bc, ab(c + 1)\}$

5) $N((a + 1)bc) = \{x\}$, as
$(a + 1) * (a + 1)bc =_{BR} (a + 1)bc \neq_{BR} 0$
$N(ab(c + 1)) = \{y\}$, as
$(a + 1) * ab(c + 1) =_{BR} 0$
$(a + ac) * ab(c + 1) =_{BR} ab(c + 1) + abc(c + 1) =_{BR} ab(c + 1) \neq_{BR} 0$

6) $\sigma = \{x \leftarrow (a + 1)bc,$
$y \leftarrow ab(c + 1)\}$

ii) "Basis of Coefficients" method (method 3)

1) $t = (bc + ab) + (a + 1)x + bxy + (a + ac)y$

2) $t_{hom} = (a + 1)x + bxy + (a + ac)y \qquad t_{inhom} = (bc + ab)$

3) $t_{hom} = (a + 1)x + (a + ac)y + bxy$

4) a) $(bc + ab) * (a + 1 + 1) * (a + ac + 1) * (b + 1) \neq_{BR} 0 \Rightarrow$ unifiable

b) $(bc + ab) * (a + 1) * (a + ac + 1) * b =_{BR} abc + bc$

$(bc + ab) * a * (a + ac) * b =_{BR} abc + ab$

$D' = \{abc + bc, abc + ab\}$

c) $H = \{(abc + bc, \{x\}), (abc + ab, \{y\})\}$

5) $\sigma = \{x \leftarrow abc + bc =_{BR} (a + 1)bc,$

$y \leftarrow abc + ab =_{BR} ab(c + 1)\}$

Evidently those two algorithms are identical in their first three steps and the last step. Inbetween those steps the two algorithm progress differently. They use two different Boolean rings. One is a ring B_C generated by the constants C, the other one a ring B_K generated by the coefficients K. Important to the complexity are two points: the number of equations to be solved, which is equal to the number of necessary basisvectors, and the complexity of calculation of a single basisvector.

The number of basisvectors for the "Basis of Constants" method is equal to 2^C. Yet in our modified version we need only the basisvectors describing the inhomogeneous part. The size of this subset D'_C can range from 1 ($t_{inhom} = \prod_{c \in C} c$) to 2^C ($t_{inhom} = 1$). The case ($t_{inhom} = 0$) with zero necessary basisvectors is a handled a special branch of the algorithm, which computes directly a particular solution. Intuitively the average number of necessary basisvectors can be determined as 2^{C-1}. Thereby an average of $N_C = 2^{C-1}$ equations have to be solved. The calculation of one basisvector is easy, as the basisvectors are constructed from unary and independant constants. Therefore no simplification has to be executed. Yet in step five of the "Basis of Constants" approach, we perform additional checks for every basisvector of D' to detect the first coefficient satisfying the test condition. During one of these tests we would detect non-unifiability of terms. So non-unifiability is determined before the construction of the most general solution from a particular one. The complexity of method 2 is the product of time needed for one test of step five, the number of tests executable for one basisvector and the number of necessary basisvectors: $O(\text{method 2}) = m * k * 2^{C-1}$.

The number of basisvectors for the "Basis of Coefficients" method is less or equal to the number of basisvectors for the "Basis of Constants" method, as B_K is a subring of B_C (Proof: every basisvector of ring B_K can be represented as a linear combination of basisvectors of ring B_C). If both rings have the same decriptive power, then they will subject to an identical orthogonal basis (Proof: uniqueness

of the basis) and thereby will produce the same solution. The smaller cardinality of B_K's basis might have been an advantage for method 3. Yet, while computing the basis, we compute several vectors equal to 0. The number of this vector is $v \leq 2^K - 2^C$ with 2^C the size of the basis of B_C. K is limited to $0 \leq K \leq \text{minimum}(2^V, 2^C)$, where 2^C is the number of possible different coefficients and 2^V the number of possible different variable strings. Hence, even though the cardinality of its basis is smaller compared to method 2, it is extremely probable, that in most cases a lot of the unnecessary vectors equal to 0 are computed. For our modified version we just need the basisvectors describing the inhomogeneous part $N_K = |D'_K| \leq 2^{C-1}$. While computing this set D'_K of basisvectors we construct 2^{K-1} vectors, of which $w \geq 2^{K-1} - 2^{C-1}$ are equal to 0. Hence the proportion of basisvectors to vector equal to 0 are the same as in the unmodified version. Computation of one basisvector is difficult, as it is computed from n-place coefficients, which might share some constants. Therefore simplifications have to be executed. After computing the basisvector of D'_K, we have to solve a corresponding number of equations. The equations are easily solved, as the basisvectors are to be compared with the coefficients they are based on. Non-unifiability is detected before we start to compute any part of the particular solution by applying Boole's test. The complexity of method 3 is time used to compute a basisvector multiplied by the number of necessary basisvectors: $O(\text{method 3}) = 1 * 2^{K-1}$.

Table 3.4.1: Comparison of particular solution methods

Attributes	Method 2	Method 3
ring generators:	constants	coefficients
	unary	n-place
# ring generators:	C = # constants	K = # coefficients
possible basisvectors:	2^C	2^K
basisvectors:	2^C	$2^K - v \leq 2^C$
vectors computed for D':	$1 \ldots 2^C$	2^{K-1}
D':	$1 \ldots 2^C$	$2^{K-1} - w \leq 2^{C-1}$
computation of D':	fast	slow
	no simplification	simplification
tests detecting, which coefficient $d_ik = 0$:	$1 \ldots K$	0
non-unifiability detection:	during one of tests	before calculation of D'
complexity:	$O(l * K * 2^{C-1})$	$O(m * 2^{K-1})$

To sum up the comparison we have the following advantages and disadvantages for the two approaches:

1) If the orthogonal form has more coefficients than constants, then the "Basis of Constants" method is superior to the "Basis of Coefficients" method.

2) An orthogonal form with more constants than coefficients, benefits the "Basis of Coefficients" method, as the number of constants directly influences the number of the equations to be solved in the "Basis of Constants" method.

3) Non-unifiability is detected almost at the same time (depends on, which of the 1... k tests of method 2 fails).

The advantage of method 2 in the calculation of **one** basisvector is partially used up by the disadvantage of necessity of computation of **one** corresponding coefficient. Therefore the time behaviour of the algorithms is pretty much the same for "normal" cases, where the number of constants and coefficients are alike. In fact it is exponential in variables and constants. If we extend the methods with the construction of the most general unifier, then those complexities get even more similar. Our test runs have proven that most of the time is spent for creation of the most general unifier. All points considered, method 2 has a slight advantage.

Now let us compare those methods with the "Variable Elimination" method.

Method 1 solves our example in the following fashion:

$t = bc + ab + ax + x + bxy + ay + acy$

$t_x = (a + 1 + by + 1)(bc + ab + ay + acy)$

$=_{BR} abc + ab + ay + acy + bcy + abcy$

$t_{xy} = (a + ac + ab + abc + 1) * (abc + ab)$

$=_{BR} 0$

$\tau = \{\}$

$\tau = \{\} \circ \{y \leftarrow abc + ab + y'(a + ac + bc + abc + 1)\}$

$=_{BR} \{y \leftarrow abc + ab + ay' + acy' + bcy' + abcy' + y'\}$

$\tau = \{y \leftarrow abc + ab + ay' + acy' + bcy' + abcy' + y'\}$

$\circ \{x \leftarrow (bc + ab + ay + acy) + x'(a + by)\}$

$=_{BR} \{x \leftarrow abx'y' + bx'y' + bcx'y' + ax' + abx' + abcx' + abc + ab,$

$y \leftarrow abc + ab + ay' + acy' + bcy' + abcy' + y'\}$

Method 2 and Method 3 would compute this most general unifier:

τ = { x ← ax'y' + abcx'y' + acx'y' + bx'y' + bcx'y' + ax' + abx' + abcx' + abc + ab,
y ← x'y' + ax'y' + abx'y' + abcx'y' + bx'y' + ay' + abcy' + acy' + bcy' + ab + abc}

Method 1 consists of two major steps: variable elimination and resubstitution of reproductive solutions. Mainly the first step with the operation <(a + 1) * b> is responsible for the complexity of the algorithm. We have already disclosed, that during elimination the termsize n increases squarewise in the worst case. Mathematically this results in a worst case complexity of ${}_{n}2^{2*V}$. Yet the Boolean ring itself sets a limit to the termsize, which is determined by the free constants C and the available variables in each recursion step. This limit is approximately $(|V| + |C|)^2$. Assuming the termsize would just double during elimination, which may be achieved by good variable selections, we would still have a complexity of 2^V * n. This adds up to an average, which is between exponential and hyperexponential in variables. If the time behaviour would be really that bad, then method 1 would not be applicable. Yet one has to remember, that simplification will rigorously cut down the termsizes and thereby the complexity. This is also true for the resubstitution step, which is similar to the application of Löwenheim's theorem of method 2 and 3. Method 1 substitutes some variables with terms, while the other methods mainly multiply the particular solution and the term itself. Therefore simplification is a vital process in all cases.

Another disadvantage of the first method is its late detection of non-unifiability. The elimination step has to be executed for all variables. This is especially serious, as this is the most time consuming step in the algorithm. Method 2 discovers non-unifiability, while solving one of its equations. Method 3 executes a special test before even starting the calculation.

Nevertheless method 1 has also some advantages. Those lie in the computed unifier itself. While all variables of the term are included in the domain of method 2 and 3, it is possible for method 1, that some variables are not in the domain. This means that the cardinality of method 1's unifier is less or equal than of the other methods. One of the main objectives of the variable selection for elimination, is to maximize the occurrences of this effect.

Example from Chapter 3.1

t = xa + yad + yae + zab + zc + zd

Method 1

$\tau = \{x \leftarrow x' + yad + yae + z'ab + z'ac + z'ad + x'a,$
$z \leftarrow z' + z'ac + z'ad + z'c + z'd\}$

Method 2/3

$\sigma = \{x \leftarrow 0, z \leftarrow 0, y \leftarrow 0\}$

$\tau = \{x \leftarrow x' + adx'y' + aex'y' + abx'z' + cx'z' + dx'z' + ax',$
$z \leftarrow z' + ady'z' + aey'z' + ax'z' + abz' + cz' + dz',$
$y \leftarrow y' + ax'y' + aby'z' + cy'z' + dy'z' + ady' + aey'\}$

Another favourable attribute of method 1 is, that the newly introduced variables do not occur in all codomain terms as in the other methods. Therefore there is less interdependency between the codomain terms and these terms are smaller. This is naturally advantageous for application of a unifier.

Table 3.4.2: Comparison of three unification algorithm for Boolean rings

attributes \ methods	Method 1	Method 2	Method 3
ununifiability detection	late	early	early
unifier cardinality	$\leq \lvert V(t)\rvert$ $\leq \lvert$ unifier 2/3 $\rvert$	normal version: $= \lvert V(t)\rvert$ improved version: $\leq \lvert V(t)\rvert$ $\geq \lvert$ unifier 1 $\rvert$	normal version: $= \lvert V(t)\rvert$ improved version: $\leq \lvert V(t)\rvert$ $\geq \lvert$ unifier 1 $\rvert$
unifier codomain	smaller terms	larger terms	larger terms
termoperations	extremly large <(a + 1) * b>	small to large <basisvector test> <construction of mgu>	middle to large <calculation of basis> <construction of mgu>

Summarizing our comparison we draw the following conclusions. Method 2 and 3, which are variations of the particular solution method, have a similar, almost identical complexity. We favour Method 2, as it tackles most unification problems slightly better. In comparison to the method of "Successive Variable Elimination", Method 1, we tend to Method 2 or 3, because they detect non-unifiability earlier and react exponentially better to variables. There are still some advantages for

Method 1, which lie in the application of unifiers; especially, if this method is able to compute a unifier with a smaller cardinality. Yet our improvements diminish the occurrences of this later effect.

In the appendix we give results of several test runs. We have selected three representative sets from our test sequences developed during the course of this thesis. The results presented originate in test runs executed in August 1988 on a Symbolics Lispmachine.

One sequence, Sequence 3, compares the "Variable Elimination" method, method 1, with the methods of "Particular Solution" method, method 2/3. The comparison covers cpu-time, number of cons cells used and cardinality of computed unifiers. Technically Sequence 3 is divided into four subsequences, which have an identical basis structure, where the variable set is increased from one example to another. The subsequences differ in a "constant" factor, which is added to each term. This sequence shows, that algorithm1 is capable of computing unifiers with a smaller cardinality. Yet, if the example involves more than four variables, method 1 needs decisively more computation time and storage cells. Increasing the constant set of the problem term intensifies this effect.

Two typical effects occur in Sequence 3. On the one hand method 1 computes a unifier with a smaller cardinality, meaning it executes less recursions, on the other hand method 2 and method 3 directly generate a particular solution (special solution 2). Even though those two effects balance each other, we took care that in the other sequences all unifiers have the same cardinality, which is equal to the cardinality of the variable set, and that no particular or general unifiers are generated directly. The latter allows a comparison of all three methods, as method 2 and method 3 proceed differently. In Sequence 1 we increase the variable set from one example to the next. Here we detect, that method 2 is just slightly better than method 3. Just as in the previous sequence, method 2 and method 3 perform better than method 1, when we increase the variable set. Sequence 2 consists of unification problems, where non-unifiability is detected. Just as in Sequence 1, the "Particular Solution" methods perform equally well and decisively better than the "Variable Elimination" method. This effect is amplified, when the variable set is incremented form one example to another. This is as expected, because the number of executed recursions of method 1 is directly related to the variable set.

Concluding the interpretation of the test results, we state that all methods perform almost equally well, if the size of variable set is smaller than four. Yet, if it is bigger, method 1 uses often too much time and storage. We choose not to present the computed unifiers, as they are too big and not that expressive to an inexperienced eye. A comparison of the unifiers shows that those for method 2 and method 3 are almost always identical. This is partially due to the fact, that they produce the same solution, if the subring relation of the two rings is just a relation of equivalence. Unifiers of method 1 are less complicated, which is already indicated by the smaller cardinality in sequence one. Thereby we find the conclusions drawn above supported in all aspects.

3.5 Prospects and Possible Improvements

As discussed in the previous chapter, the unifier's cardinality is of great significance to application of a unifier. Therefore it is aimed to compute a unifier with a reduced cardinality. This seems only possible for the method of "Successive Variable Elimination", as the other method's cardinality is generally determined by variable set of the term to be unified. The reduction would also decisively decrease the complexity of the algorithm, as we would execute less recursion steps. To establish this effect we have to develop a better variable selection criterion. As such a criterion is difficult to discover theoretically because of the possible simplifications, this leaves only the practical approach with a massive number of test runs. Both choices do not seem satisfying. Any serious work in the direction of smaller cardinality should start with exhaustive reflections about a possible minimal unifier, a unifier with a minimal number of domain variables. Having solved this problem one could search for a reduction algorithm, which has a unifier as input and computes a BR-equal smaller unifier or even better the minimal unifier. This algorithm could be applied on the results of the particular solutions methods. If the variable selection of the "Variable Elimination" method can not be improved to compute a minimal unifier, we could apply the reduction algorithm on its solutions as well. Nevertheless it may be assumed, that such a reduction algorithm would be time intensive; this makes a sensible use doubtful.

Therefore we presume, that research in the direction "Detection of Most General Unifiers and Particular Unifiers" is more rewarding. We have already introduced our results in that area into the unification algorithms covering a great deal of cases.

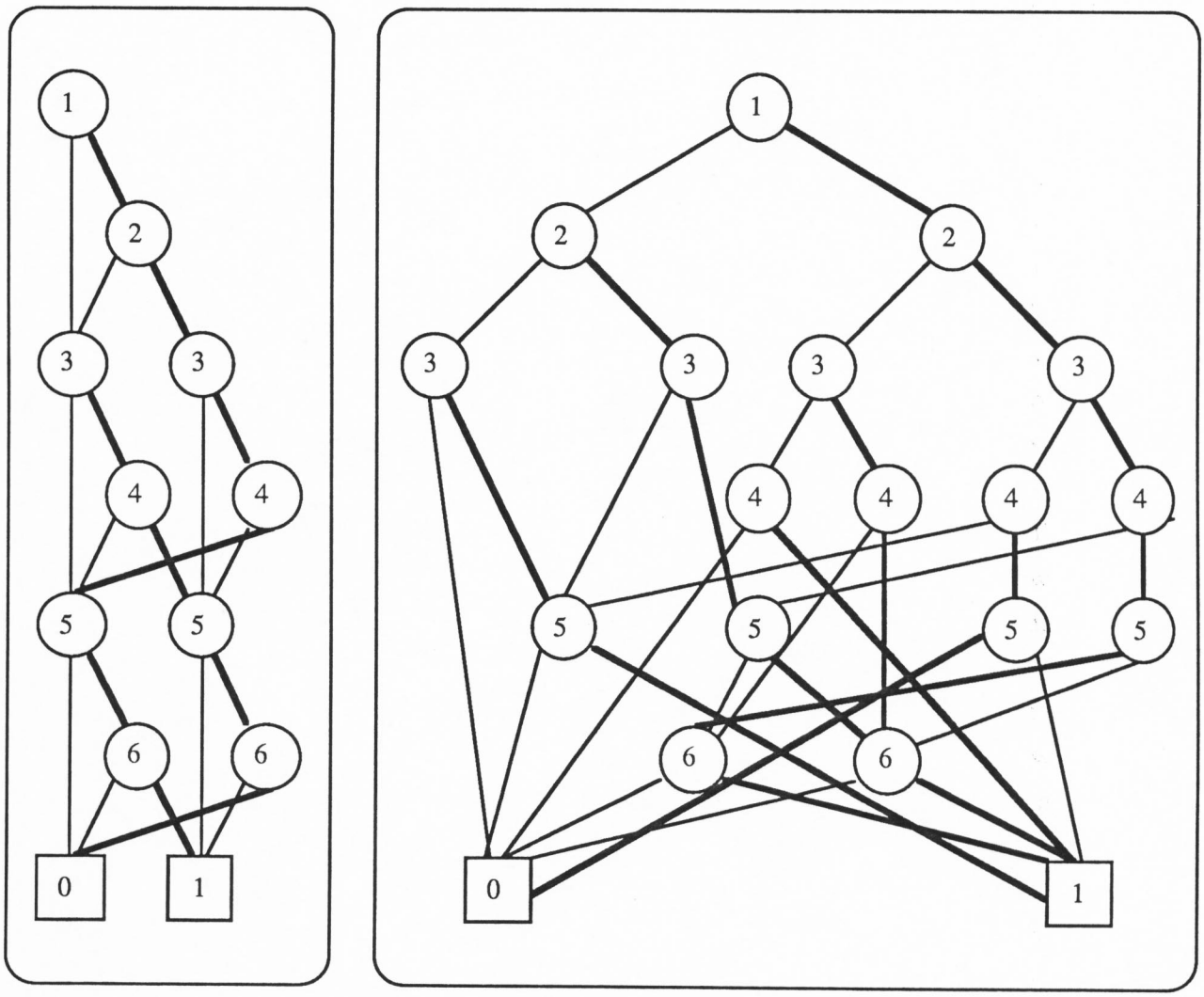
An alternative non-algorithmic approach would be the selection of a different datastructure. R.E.Bryant proposed directed acyclic graphs (DAGS) for representing Boolean functions [Bry 86]. Besides the obvious conversion problems caused by the necessity of connection to the HADES [Ohl 88], it has some additional disadvantages. Let us sketch his ideas for a better understanding. With his DAGS he represents a term by describing its value (leafs) with the possible values (arcs) of all variables (vertices) occurring in a term; identical subgraphs are shared. Bryant offers an associated set of algorithms for detection of satisfiability, equality, etc. Those algorithms operate directly on the graph. Thereby he achieves almost linear complexity excluding graph creation. Even though he considered the Boolean algebra as main application, one can also use them for the Boolean ring. Nevertheless Bryant's approach has a disadvantage:

The representation size of the graph depends heavily on the argument ordering.

The following example, where value 1 is represented by a thick line and a value 0 by a thin line, pinpoints this effect.

Example in a Boolean ring B<Ø, +, *, 0, 1> over variables V = {x_1, x_2, x_3, x_4, x_5, x_6}:

$x_1x_2 + x_3x_4 + x_5x_6$ $x_1x_4 + x_2x_5 + x_3x_6$



Bryant has not found a criterion, which determines the optimal ordering other than by explicitly testing the possibilities. For his applications it was sufficient to find an acceptable ordering by analyzing the problem domain. Yet because of the exponential termgrowth connected with problems in our field we need a criterion, which automatically chooses an optimal ordering.

Nevertheless our main dilemma is, that he expects the algebra to be an initial, or 2-element algebra. In our application however we have C, a set of constants, in the signature. This decisively increases the graph size, as not only two arcs originate from every vertex, but 2^C arcs. Therefore we used our datastructure with its structure sharing properties.

4. COMBINATION OF BOOLEAN RING & FREE THEORY

4.1 Unification Algorithm

This algorithm is based on an algorithm of M. Schmidt-Schauß, which handles unification in the combination of an arbitrary and a simple theory [Schm 88]. When he presented this algorithm at the 9th Conference of Automated Deduction, he used the combination of the Boolean ring (arbitrary) and the free (simple) theory as one of his examples. Our work adapts, realizes and advances his ideas. We contribute improvements to several steps of the algorithm and change its structure to make it more suitable to this specific combination of theories. Yet we are consistent with the basic notions of M. Schmidt-Schauß [Schm 88].

The combination algorithm for Boolean ring theory with free function symbols consists of four major parts. In order of their application these are preprocessing, identification, cycle elimination and postprocessing. This Chapter 4.1 is structured analogously. The first Section 4.1.1 gives some introductory notes on the algorithm and presents definitions and standard procedures, which are applied in all parts of the algorithm. In Section 4.1.2 we present pre- and postprocessing, in Section 4.1.3 the identification and finally in Section 4.1.4 the cycle elimination.

4.1.1 Introduction

We will consider the process of unification as a sequence of transformation steps that start with a given system of equations, the unification problem, and stops with a set of systems of equations, the set of solutions. This complies with the ideas of Martelli/Montanari [MM 82], Kirchner [Kir 85] and Schmidt-Schauß [Schm 88]. We will use multi-equations instead of equations to represent our problems. Thereby we have MS as a set of multi-equations, where each multi-equation ME_i is a set of terms $\{t_{i1}, \ldots, t_{ini}\}$ also denoted as $t_{i1} = \ldots = t_{ini}$. As mentioned earlier every system of equations can be transformed into this form. We use $s = t \in MS$ synonymously with $s, t \in ME$, where ME is a multi-equation. By merging we will achieve that, if $r = s$ and $s = t$ are in a multi-equation system MS, then $r = t$ in MS or equivalently that all multi-equations are disjoint. Often we will use equations of the form $S = T$, where the uppercase letters denote sets of terms and $S = T$ means a conjunction of all equations $s_i = t_j$ for $s_i \in S$ and $t_j \in T$.

The terms of the unification problems investigated are in the combination of Boolean ring theory with free function symbols. We already explained that there are deterministic normal forms for terms in the Boolean ring, if we have defined an ordering on the terms. The normal form we have selected is

the disjunctive normal form. Obviously we can extend this deterministic normal form to terms in the combination of Boolean ring with free function symbols.

This idea has several advantages:

1) We can drop the distinction between syntactical and semantical theory of a term as they are identical. By Schmidt-Schauß [Schm 88] the syntactical theory of a term is the theory of its topsymbol and the semantical theory is the theory of the topsymbol of the term in maximal collapsed form. Thereby in the following we will employ the expression theory of a term.

2) For all theories concerned, the Boolean ring theory, the free theory and their combination theory-equal terms are also syntactical equal.

We use this classifications of terms:

free term:	a term is member of the free theory
boolean term:	a term is member of the Boolean ring theory
compound term:	a term is not a variable nor a constant
significant variable:	a free variable is present in original problem
significant constant:	a free constant is present in original problem
auxiliary variable:	a free variable is not present in original problem
auxiliary constant:	a free constant is not present in original problem

We will use the expression '**auxiliaries**' to denote the set of auxiliary variables and auxiliary constants. 'Auxiliary constants' are auxiliary variables, which are interpreted as constants on theory level .

There are two applications of auxiliaries: Renaming (variables only) and Abstraction

A cycle in a multi-equation system (MS) is a set $\{x_i = t_i \mid 1 \leq i \leq n\}$ with $x_i = t_i$ in MS, where x_i is an auxiliary and t_i is a compound term, such that x_{i+1} occurs in t_i for $1 \leq i \leq n - 1$ and x_1 occurs in t_n. A system of multi-equation MS is in **sequentially solved** form, if every multi-equation has at most one compound term and contains no cycles. It is in **solved** form, if no auxiliary of MS occurs in a compound term of MS. From a multi-equation system in solved form we can obviously construct an idempotent most general unifier.

The combination algorithm will be described by a set of rules, which transform a multi-equation system MS_0 into a set of multi-equation systems in the solved form. For every solution σ of MS_0 there has to be an element in this set, which can be used to create a solution more general than σ. The rules of the algorithm work on multi-equations in our own representation, which in Chapter 5 is denoted by 'theory.termlist' to distinguish them from the multi-equations of the input and output systems. Our multi-equations have three disjunct components, which are roughly based on the

theories involved:

$ME := ME^V = ME^F = ME^B$, where ME^X denotes a set of terms of a specific type $x \in \{V, B, F\}$.

(V) **Variables**: significant variables

(F) **Free**: free compound terms, free and boolean constants (0, 1) and auxiliary variables used in the free compound terms of MS

(B) **Boolean**: boolean compound terms and auxiliaries used in boolean compound terms of MS.

The multi-equation system MS can be divided accordingly, i.e., when $MS = \{ME_i \mid 1 \leq i \leq n\}$, then $MS^X = \{ME_i^X \mid 1 \leq i \leq n\}$. Note that auxiliaries used in boolean and free part are disjunct and auxiliary constants occur only in the boolean part (see Section 4.1.3). The difference to a classification, which would have partitioned terms strictly according to their theories, is that boolean constants are stored together with free constants in the free part. The reasons for this decision are special properties of the constants: Boolean and free constants can be handled by unification algorithms of both theories. Unification algorithms of the free theory can regard boolean constants as free constants and unification algorithms of Boolean ring theory are equipped to handle free constants. Of course, Boolean ring unification algorithms will have to take into account, that all constants are stored in the free part.

The following rules are applied in all steps of the algorithm to reduce multi-equation systems to smaller equivalent systems. They will be referred to as **reduction rules**:

Rule: Trivial Multi-equations.
$MS \;\&\; ME \Longrightarrow MS,$
if ME contains only one element.

Rule: Auxiliaries. $MS \;\&\; ME^x \Longrightarrow MS \;\&\; ME^x \setminus \{z\},$
if z is an auxiliary in ME^x and does not occur in any term of MS^x or $ME^x \setminus \{z\}$; $x \in \{B, F\}$

Rule: Merge. $ME_1^V = ME_1^F = ME_1^B \;\&\; ME_2^V = ME_2^F = ME_2^B$
$\Longrightarrow ME_1^V \cup ME_2^V = ME_1^F \cup ME_2^F = ME_1^B \cup ME_2^B,$
if there are terms $t_1 \in ME_1^x$ and $t_2 \in ME_2^x$, which are syntactical equal; $x \in \{B, F, V\}$

Rule: Equal Terms. $ME^x \Longrightarrow ME^x \setminus \{t\},$
if a component x of ME contains term t twice; $x \in \{B, F, V\}$

Obviously these rules do not change the set of solutions. The restrictions to different components are legitimate, as the term sets as well as the sets of in terms occurring auxiliaries are disjunct.

4.1.2 Pre- and Postprocessing

The first goal of preprocessing is to transform the multi-equation system into **unfolded normal form** (UNF). Such an equation system has these properties: All terms are pure, all variables contained in compound terms are auxiliary and compound terms of the free and boolean part have disjoint sets of variables and every auxiliary of MS occurs in a compound term in MS.

We accomplished this transformation by application of the partitioning rule, the unfolding rule and the renaming rule:

Rule: Partitioning. $ME \Longrightarrow ME^V = ME^B = ME^F$

with $ME^V = \{t \mid t \in ME \text{ is a variable}\}$

$ME^B = \{t \mid t \in ME \text{ is a boolean compound term}\}$

$ME^F = \{t \mid t \in ME \text{ is a free compound term or boolean or free constant}\}$

Rule: Unfolding. $ME \cup \{t\}^B \Longrightarrow ME \cup \{t[\pi \leftarrow x_B]\}^B \,\&\, \{\}^V = \{x_B\}^B = \{s\}^F,$

if s is a direct alien compound subterm of a boolean term t at occurrence π and x_B is a new auxiliary variable. (analogously for a free term t)

Rule: Renaming. $MS^B \Longrightarrow \{x \leftarrow x_B\}MS^B \,\&\, \{x\}^V = \{x_B\}^B = \{\}^F,$

if $x \in$ Variables(t) for some term t of input multi-equation system MS_0 occurring in MS^B and x_B is a new auxiliary variable. (analogously for free part of MS)

At the start of the algorithm we convert multi-equations of system MS_0, which represents the unification problem, into multi-equations of our internal representation. This achieved by application of the rule Partitioning, which splits multi-equations into their disjunct components. Then we replace or unfold occurrences of direct alien compound terms in term t by an auxiliary variable. For every replacement we add to the system a multi-equation, which has two filled components: The auxiliary variable is stored in the component related to the theory of term t, while the alien is placed in the component related to the alien's theory. This unfolding process is also called **variable abstraction.** After completion of unfolding we have just pure terms in our multi-equation system. Finally we rename the significant variables in the boolean and free part of our converted system. Note that occurrences of a significant variable x in the boolean part is replaced by a different auxiliary variable than occurrences in the free part. This makes the variable sets of the components of a multi-equation system disjunctive. Two reduction rules are applicable in this transformation process of a unification problem into UNF: Merging and Equal Terms. These rules reestablish disjunctiveness of the term sets of the multi-equations; after unfolding, when we have abstracted the same alien with two different auxiliary variables, and after renaming, when a significant variable occurs in the free and the boolean

part of the converted system. All these rules, Partitioning, Unfolding and Renaming do not change the set of solutions. Thereby these rules and the reduction rules are called **'don't care' rules.**

Example: Transformation of $MS_0 = \langle f(x) = x + f(y) \rangle$ into unfolded normal form.
Let BR = <+, *, 0, 1> be a Boolean ring, f a unary free function symbol, x, y are significant variables and z, x_F, y_F, x_B are auxiliary variables.

Partitioning: $\langle \{\}^V = \{ f(x) \}^F = \{ x + f(y) \}^B \rangle$
Unfolding: $\langle \{\}^V = \{ f(x) \}^F = \{ x + z \}^B$
$\{\}^V = \{ f(y) \}^F = \{ z \}^B \rangle$
Renaming: $\langle \{\}^V = \{ f(x_F) \}^F = \{ x_B + z \}^B$
$\{\}^V = \{ f(y_F) \}^F = \{ z \}^B$
$\{ x \}^V = \{ y_F \}^F = \{\}^B$
$\{ y \}^V = \{ x_F \}^F = \{ x_B \}^B \rangle$

The final aim of preprocessing is to transform our multi-equation systems into **separated unfolded normal form**. Multi-equations of such a form have theory components (boolean and free), which contain at most one member. This transformation is achieved by unifying the terms in these two theory components. There are two rules:

Rule: F-Unification.

$\{ME_i^V = ME_i^F = ME_i^B \mid 1 \le i \le n\} \Longrightarrow \{ME_i^V = ME'^F_i = ME_i^B \mid 1 \le i \le n\}$,

if there is a free component of a multi-equation ME_j with $|ME_j^F| > 1$.
Let σ_F be a free mgu of free part of multi-equation system $MS^F = \langle \{ME_i^F \mid 1 \le i \le n\} \rangle_F$.
$ME'^F_i := \sigma_F ME_i^F$.

Rule: BR-Unification.

$\{ME_i^V = ME_i^F = ME_i^B \mid 1 \le i \le n\} \Longrightarrow \{ME_i^V = ME'^F_i = ME'^B_i \mid 1 \le i \le n\}$,

if the free component of every multi-equation fulfils $|ME_i^F| \le 1$ and there is a $|ME_j^P| > 1$ with
$ME_j^P := ME_j^B \cup \{c \mid c \in ME_j^F \text{ is a constant}\}$.
Let σ_B be a Boolean ring mgu of $\langle \{ME_i^P \mid 1 \le i \le n\} \rangle_{BR}$.
If $s \in ME_i^F$ is a constant, then
$ME'^F_i := ME_i^F$ and $ME'^B_i := \{\}$,
else if $t \in \sigma_B ME_i^B$ is a (significant) constant, then
$ME'^F_i := ME_i^F \cup \sigma_B ME_i^B$ and $ME'^B_i := \{\}$,
else $ME'^F_i := ME_i^F$ and $ME'^B_i := \sigma_B ME_i^B$.

Remarks:

After application of a unifier to a component x of a multi-equation ($\sigma_x ME^X$) this component consist of one element.

We do not have to add the unifier to the multi-equation system, as its domain consists of auxiliary variables, which are no longer present in the system after application of the unifier. Also it is sufficient to apply the unifier to the corresponding theory component of the system, as the sets of auxiliary variables occurring in terms of the theory part are disjunct.

Generally after an application of a unification rule we can apply the reduction rules. Reduction rules ensure that multi-equations stay disjoined and that the theory part of multi-equations, for which we had executed unification, contains at most one term. The other theory components might consist of more than one element; either caused by merging or by the transfer of a constants into the free part. In general this leads to an alternating application of unification rules (F-Unification BR-Unification)*, until both theory parts of all multi-equations contain at most one term.

The unification algorithm employed, which computes the unifier for F-Unification, is similar to the approach by Robinson [Rob 65]. Yet our algorithms works on multi-equations and thereby uses additional information provided by these multi-equations. For the unification algorithm, which computes the unifier for BR-Unification, we had to choose among the three methods of Chapter 3. All of them are capable of performing unification in a free Boolean ring, i.e. Boolean ring with free constants. As the unifiers are often applied to the system, an important characteristic of the unifier is its complexity. Therefore we decided to use the "Variable Elimination" method, as its unifiers are simpler and have a smaller complexity as the unifier computed by the two "Particular Solution" methods. The drawback of the "Variable Elimination" method was its speed. Yet we assume that the combination algorithm will handle mostly equation systems with "small" Boolean ring problems, as the complexity of the combination algorithm is already enormous (see Section 4.1.3 & 4.1.4). For cases with small Boolean ring problems the difference in speed is insignificant.

Example: Transformation of $MS_0 = \langle f(a + b + x) = f(a + b), y = f(x)\rangle$ into separated unfolded normal form.

Let BR = <+, *, 0, 1> be a Boolean ring, f a unary free function symbol, x, y are significant variables, a, b are free constants and z, w, x_F, x_B are auxiliary variables.

UNF: Execution of partitioning, unfolding and renaming.

$\langle \{ x \}^V = \{ x_F \}^F = \{ x_B \}^B$

$\{ y \}^V = \{ f(x_F) \}^F = \{\}^B$

$\{\}^V = \{ \underline{f(z), f(w)} \}^F = \{\}^B$

$\{\}^V = \{ z \}^F = \{ a + b + x_B \}^B$

$\{\}^V = \{ w \}^F = \{ a + b \}^B \rangle$

F-Unification: $\sigma_F = \{z \leftarrow w\}$, after application of σ_F the reduction rules remove third ME, merge the fourth and fifth ME and delete auxiliary variable w.

$\langle \{ x \}^V = \{ x_F \}^F = \{ x_B \}^B$

$\{ y \}^V = \{ f(x_F) \}^F = \{\}^B$

$\{\}^V = \{\}^F = \{ \underline{a + b, a + b + x_B} \}^B \rangle$

BR-Unification: $\sigma_B = \{x_B \leftarrow 0\}$, after application of σ_B the constant 0 is shifted. Then the reduction rules remove the third ME.

$\langle \{ x \}^V = \{ \underline{x_F, 0} \}^F = \{\}^B$

$\{ y \}^V = \{ f(x_F) \}^F = \{\}^B \rangle$

F-Unification: $\sigma_F = \{x_F \leftarrow 0\}$

$\langle \{ x \}^V = \{ 0 \}^F = \{\}^B$

$\{ y \}^V = \{ f(0) \}^F = \{\}^B \rangle$

Beyond postprocessing we have performed the identification and cycle elimination steps. At this stage of the algorithm the multi-equation systems are in sequentially solved form. The objective of postprocessing is to transform these systems into solved form. Thereby we have to replace all occurrences of auxiliaries by their computed values: either by a compound term of opposite theory or a significant variable. We proceed in two steps. First we eliminate the auxiliaries occurring in the boolean component of the multi-equation system. After this elimination the components of the system MS^V contain significant variables, MS^F pure free terms and boolean constants and MS^B mixed boolean compound terms. Note that the variables occurring in terms of MS^F and MS^B are no longer disjunct. Now we eliminate the auxiliaries used in the free part, which converts our system into solved form. For this form we drop the distinction between the components of multi-equations to get the form of the input multi-equation system MS_0.

For the elimination of boolean auxiliaries there are two nondeterministically applicable rules: one handles the auxiliary (abstraction) constants, the other the auxiliary variables. For elimination of free auxiliaries we need one rule.

Rule: Elimination$_{B.c}$. MS & ME ==> MS' & ME',
if $c \in ME^B$ is an auxiliary constant.
Then $ME^F = \{t\}$ with a pure free compound term t.
$MS' := MS^V = MS^F = [c \leftarrow t]\, MS^B$
If $|ME^V \cup ME^F| > 1$,
ME' := $ME^V = ME^F = \{\}^B$, else
ME' := $\{\}$.

Rule: Elimination$_{B.v}$. MS & ME ==> MS' & ME',
if $x \in ME^B$ is an auxiliary variable.
Select a significant variable $v \in ME^V$.
$MS' := MS^V = MS^F = [x \leftarrow v]\, MS^B$
If $|ME^V \cup ME^F| > 1$,
ME' := $ME^V = ME^F = \{\}^B$, else
ME' := $\{\}$.

Rule: Elimination$_{F}$. MS & ME ==> MS' & ME',
if $x \in ME^F$ is an auxiliary variable.
If $ME^B = \emptyset$, select a significant variable $v \in ME^V$ and let s := v, else
let s := t with t, a mixed boolean compound term of $ME^B = \{t\}$.
$MS' := MS^V = [x \leftarrow s]\, MS^F = [x \leftarrow s]\, MS^B$
If $|ME^V \cup ME^B| > 1$,
ME' := $ME^V = \{\}^F = ME^B$, else
ME' := $\{\}$.

To implement the elimination rules of postprocessing we are presented with a choice. We can implement the rules as explained above, which means we execute one replacement of an auxiliary after another. Or we combine the replacements into two substitutions (one for the boolean part and one for the free part), which are applied to the components as in the rules. We decided to use the later approach, as in that case we create less intermediate terms. We can easily construct an idempotent substitution from the replacements of the two boolean rules (Elimination$_{B.c}$ and Elimination$_{B.v}$), as the domain auxiliaries and the auxiliaries occurring in the codomain terms are disjunct. This is not the case, when we build the substitution for the free part. Yet we know that all cycles have been removed.

Therefore we can construct an idempotent unifier from these replacements too.

Postprocessing rules do not change the set of solutions, as they just eliminate auxiliaries, which did not occur in the input multi-equation system. So they can be included in the set of 'don't care rules'.

4.1.3 Identification

Before executing the identification step preprocessing has to be concluded. Thereby at this stage the multi-equation system is in separated unfolded form. It consists of multi-equations with theory components, which contain at most one term. The aim of the two middle steps, identification and cycle elimination, is to transform this system into a sequentially solved form, where every multi-equation contains at most one compound term of either theory and no cycles. To fulfill the second premise is the task of cycle elimination. Identification has to resolve those multi-equations in the system, which have a compound term in the free part and another compound term in the boolean part.

Adding different additional auxiliary variables to different multi-equations does not change the solution space. Therefore we may insert one auxiliary in the boolean part of every multi-equation with a compound term in the free part and a non-empty boolean part. Provided that all these free compound terms are non-unifiable, then these auxiliary variables can be treated as constant on the boolean component level. This method is called **constant abstraction** and we call such variables auxiliary "constants" or abstraction "constants". As we get multi-equations with two terms in the boolean part, BR-Unification is applicable. By the unification rule the boolean compound terms of multi-equations with compound terms in both theory parts **collapse** into the auxiliary constants.

Yet to employ constant abstraction, we have to assure the condition of non-unifiable different free compound terms. This is achieved by **identification**, which branches our multi-equation system into several other systems. Let NEW be the set of multi-equations with a free compound term and a non-empty boolean part. Then for every possible partition of NEW, there is one new multi-equation system, where the multi-equations are joined according to the sets in the partition. After such an identification it is sufficient to consider only those systems of multi-equations in our search space, which do not further identify multi-equations of the set NEW. Those solutions, which further identify multi-equations, are computed in another branch. Under these circumstances we can execute constant abstraction, as the free compound terms can no longer be unified, because this would join the multi-equations.

Below we give a detailed description of the identification step. In the identification step we proceed through this sequence of rules: $(I\ F\ C\ (B\ F)^*)^*$. F and B are the F-Unification and the BR-Unification rules as previously defined, while the identification rule (I) and the constant abstraction rule (C) are new.

The new rules:

Let OLD be set of multi-equations ME_i of multi-equation system MS, which have a previously abstracted compound term in the free part ME_i^F and a non-empty boolean part.

Let NEW be set of multi-equations ME_i of MS, which have a not yet abstracted compound term in the free part ME_i^F and a non-empty boolean part.

Rule: Identification.

$$MS \quad ==> \{MS'_j \mid 1 \le j \le m\},$$

if set NEW is not empty.

There is one MS'_j for every partition j of NEW $\cup$ OLD, of which every set in the partition j has at most one member of OLD.

MS'_j is constructed from MS by joining the multi-equations, which are in the same set of partition j.

Rule: Constant Abstraction.

$$\{ME_i^V = ME_i^F = ME_i^B \mid 1 \le i \le n\} ==> \{ME_i^V = ME_i^F = ME'^B_i \mid 1 \le i \le n\},$$

if there is a $ME_i \in$ NEW, which has no abstraction constant in ME_i^B.

Let c_i, c_j be different abstraction constants, if $i \neq j$.

If $ME_i \notin$ NEW or $\exists\, c_i \in ME_i^B$, which is an abstraction constant,

then $ME'^B_i := ME_i^B$,

else $ME'^B_i := \{c_i\} \cup ME_i^B$.

Besides the four rules mentioned above the reduction rules are applied whenever possible. This would be after the application of the unification rules.

We may restrict sets NEW and OLD to multi-equations with a non-empty boolean part, as those with one empty theory part and a compound term in the other theory part no longer influence the algorithm. They only remain in the system to enable postprocessing to construct a complete solution, as these multi-equations have at least one significant variable in the variable part.

The identification rule handles also recursive identification, where the set OLD is not empty. This recursive identification is necessary, because the unification rules may transform some multi-equations into a form with a free compound term and a non-empty boolean part. In recursive identification we do not have to create partitions, which join two multi-equations that already have been subjected to

prior identification. Those non-joinable multi-equations, which still can influence the algorithm, have an abstraction constant in the boolean part and are stored in set OLD. Thereby we restrict the resulting partitions in a recursive identification to those, which do not join multi-equations of set OLD.

Joining of the multi-equations makes rule F-Unification applicable, as there are multi-equations with more than one term in the free component. The application of the free unifier on the system might transform some multi-equations so that they have a free compound term and a boolean compound term. This will induce the application of the identification rule after the rule sequence of this recursion level has been concluded.

The constant abstraction inserts auxiliary constants into boolean parts of multi-equations, which have been previously joined by the identification step. During a recursion we do not have to insert constants for every set in a partition. If there is a member of set OLD in such a set of the partition, then the joined multi-equation has already the abstraction constant of this member of set OLD in the Boolean part.

After constant abstraction we start a unification cycle of BR-Unification and F-Unification just as in preprocessing. We begin with BR-Unification, as all free parts of the multi-equations have at most one member, which is provided by performance of F-Unification after identification. There are more than one member in the boolean parts caused by identification or merging induced by unifier application of F-Unification or constant abstraction. This unification cycle stops, when both theory parts have at most one member. If both theory parts contain a compound term, then we reenter the identification cycle with identification . Note that the rule BR-Unification is already prepared to perform unification with the now introduced auxiliary constants. The auxiliary constants are not shifted into the free part as the significant constant are, because the auxiliary constants are interpreted as variables on toplevel of the multi-equations.

As clarifying example we continue the first example for preprocessing (Section 4.1.2).

Example: $MS_0 = \langle f(x) = x + f(y) \rangle$

Let BR = <+, *, 0, 1> be a Boolean ring, f a unary free function symbol, x, y are significant variables and z, x_F, y_F, x_B are auxiliary variables and c_1, c_2 are auxiliary constants.

Preprocessing:
$$\langle \{\}^V = \{ f(x_F) \}^F = \{ x_B + z \}^B$$
$$\{\}^V = \{ f(y_F) \}^F = \{ z \}^B$$
$$\{ y \}^V = \{ y_F \}^F = \{\}^B$$
$$\{ x \}^V = \{ x_F \}^F = \{ x_B \}^B \rangle$$

Identification: MS': none joined MS'': ME_1 & ME_2 joined

MS': $\langle \{\}^V = \{ f(x_F) \}^F = \{ x_B + z \}^B$
$\{\}^V = \{ f(y_F) \}^F = \{ z \}^B$
$\{ y \}^V = \{ y_F \}^F = \{\}^B$
$\{ x \}^V = \{ x_F \}^F = \{ x_B \}^B \rangle$

MS'': $\langle \{\}^V = \{ \underline{f(x_F), f(y_F)} \}^F = \{ z, x_B + z \}^B$
$\{ y \}^V = \{ y_F \}^F = \{\}^B$
$\{ x \}^V = \{ x_F \}^F = \{ x_B \}^B \rangle$

Treatment of MS':

Constant Abstraction: $\langle \{\}^V = \{ f(x_F) \}^F = \{ \underline{c_1, x_B + z} \}^B$
$\{\}^V = \{ f(y_F) \}^F = \{ \underline{c_2, z} \}^B$
$\{ y \}^V = \{ y_F \}^F = \{\}^B$
$\{ x \}^V = \{ x_F \}^F = \{ x_B \}^B \rangle$

BR-Unification: $\sigma_B = \{z \leftarrow c_2, x_B \leftarrow c_1 + c_2\}$
$\langle \{\}^V = \{ f(x_F) \}^F = \{ c_1 \}^B$
$\{\}^V = \{ f(y_F) \}^F = \{ c_2 \}^B$
$\{ y \}^V = \{ y_F \}^F = \{\}^B$
$\{ x \}^V = \{ x_F \}^F = \{ c_1 + c_2 \}^B \rangle$

Treatment of MS'':

F-Unification: $\sigma_F = \{y \leftarrow x\}$
ME_2, ME_3 are merged after application of σ_F.
$\langle \{\}^V = \{ f(x_F) \}^F = \{ z, x_B + z \}^B$
$\{ x, y \}^V = \{ x_F \}^F = \{ x_B \}^B \rangle$

Constant Abstraction: $\langle \{\}^V = \{ f(x_F) \}^F = \{ \underline{c_1, z, x_B + z} \}^B$
$\{ x, y \}^V = \{ x_F \}^F = \{ x_B \}^B \rangle$

BR-Unification: $\sigma_B = \{z \leftarrow c_1, x_B \leftarrow 0\}$
After application of σ_B c_1 is no longer used and thereby removed.
This leads to removal of ME_1 by rule 'Trivial Multi-equations'.
$\langle \{ x, y \}^V = \{ x_F, 0 \}^F = \{ \}^B \rangle$

F-Unification: $\langle \{ x, y \}^V = \{ 0 \}^F = \{ \}^B \rangle$

The complexity of this identification step is mainly determined by the identification rule, where all partitions of a set NEW or of set NEW $\cup$ OLD minus a restriction are computed. As the complexity of computing all partitions is exponential, we decided not to compute them all. This is possible, if we combine the Identification rule with the F-Unification rule. We will create only such partitions, which lead to a system with a unifiable free part. Yet it is not sufficient to unify the free parts of the

multi-equations to be joined before construction of the new multi-equation system.

Regarding the free unification problems of the different partitions, we see that many unification subproblems exist, which are shared by partitions (Example: partition$_i$ $\{\{ME_1ME_2ME_3\}\{ME_4ME_5\}\}$ and partition$_j$ $\{\{ME_1ME_2ME_3\}\{ME_4\}\{ME_5\}\}$ share the subproblem of unifying the free terms of $\{ME_1ME_2ME_3\}$).Thereby we proceed as follows: We compute the free unifiers of every pairs of free terms in the multi-equations to be joined. Note that a pair cannot originate in two multi-equations of set OLD. Then we start with the first partition, the one which joins no multi-equations. For this case we do not even need F-Unification. Now we check partitions by incrementally increasing the set of multi-equations to be joined from the empty to the whole set. The free unifiability check is not executed by unification of the problems, but by merging already computed free unifiers with unifiers of a pair containing the new element. Another significant effect is that a new identification is only tested, if the free parts of multi-equations of the inner set and the free part of the set extending pair of multi-equations were unifiable. By this method a lot of time is spared in F-Unification by merging unifiers and in identification by not constructing all partitions. Our method originates in an approach by M. Tepp [Tep 88]. Yet our version allows to interrupt and restart the identification. Thereby we can compute one unifier now and the rest of the unifiers on a later demand (see Chapter 5).

Example: Four multi-equations have to be identified, i.e. |NEW| = 4.

Part a) describes MS = ⟨f(x) + f(y) = f(a) + f(z)⟩.

Part b) describes MS = ⟨f(x) + f(y) = f(a) + f(b)⟩.

These abbreviations are used for the free compound terms:

a) 1 = f(z), 2 = f(b), 3 = f(x), 4 = f(y)

b) 1 = f(a), 2 = f(b), 3 = f(x), 4 = f(y)

Unifiable single equations:

a) 2=1; 3=1; 3=2; 4=1; 4=2; 4=3

b) ---; 3=1; 3=2; 4=1; 4=2; 4=3

Unification tests for Partitions, where more than a single free equation has to be solved.

a) 3=2=1 by merging results of 3=2 and 3=1; 4=1,3=2 by merging results of 4=1 and 3=2;
4=2=1 by merging results of 4=2 and 4=1; 4=2,3=1 by merging results of 4=2 and 3=1;
4=3=2=1 by merging results of 4=3 and 4=2=1; 4=3=2 by merging results of 4=3 and 4=2;
4=3=1 by merging results of 4=3 and 4=1; 4=3,2=1 by merging results of 4=3 and 2=1

b) 3=2=1 non-unifiability detected; 4=1,3=2 by merging results of 4=1 and 3=2;
4=2=1 non-unifiability detected; 4=2,3=1 by merging results of 4=2 and 3=1;
4=3=2=1 not tested; 4=3=2 by merging results of 4=3 and 4=2;
4=3=1 by merging results of 4=3 and 4=1; 4=2,3=1 not tested

Besides the combination of the identification rule with free unification, there is another difference to the approach by M. Schmidt-Schauß. He performs the constant abstraction after all identification and unification induced by that identification has been concluded:

MSS: $(I\ (F\ B)^*)^*\ C$ <===> BCR: $(I\ F\ C\ (B\ F)^*)^*$

Our approach has mainly two advantages:

1. We reduce the search space faster by simplifying BR-Unification (B) with the insertion of the abstraction constant in the boolean part. The boolean compound term collapses into the abstraction constant.
2. If recursive identification is necessary, we use results of already executed abstractions. Free compound terms are abstracted on the recursion level, where they are involved in the identification, i.e. member of set NEW. M. Schmidt-Schauß will abstract free compound terms involved in a prior identification more often, as he has to perform abstraction for all free compound terms in the resulting branch and not as we do just the new ones.

4.1.4 Cycle Elimination

The task of cycle elimination is to bring multi-equation systems into sequentially solved form, where every multi-equation contains at most one compound term of either theory and no cycles. Prior to this step the first premise has been fulfilled. Therefore this step is solemnly responsible for removing the cycles occurring in the multi-equations system. At this stages all cycles are **theory overlapping** cycles, which alternate through two types of multi-equations:

a) auxiliary variable in free part and compound term in boolean part
b) auxiliary constant in boolean part and compound term in free part

Example: $\langle \{ z \}^F = \{ y_B * c \}^B, \{ f(z) \}^F = \{ c \}^B \rangle$

Accordingly one could assume that there are two approaches to resolve those cycles:

α) Elimination of an auxiliary constant in a boolean compound term of multi-equation type a
β) Elimination of an auxiliary variable in a free compound term of multi-equation type b

Yet as the free theory is a simple theory, the auxiliary variables cannot be eliminated in the free compound terms. Therefore only approach α can resolve cycles (Example: term $y_B * c$ with constant c to be eliminated, but not variable z in term f(z)).

Below a detailed description of the cycle elimination step is given. The basic rules of this step are the cycle elimination rule, as defined below, and the already defined F-Unification rule. Beside these two rules the reduction rules are applied whenever possible to reduce the systems' complexity.

Rule: Cycle Elimination.

MS_C ==> $\{MS'_s \mid 1 \le s \le S\}$, if there is a cycle in MS.

To compute one solution s:

- Select one constant elimination problem C_{new}.
- If MS_C is a multi-equation system before application of the Cycle Elimination rule,
  then $MS_s := MS_C$ and $C_s := C_{new}$
  else determine MS_I, the multi-equation system before application of the Cycle Elimination rule,
  C_{old}, the constant elimination problem solved by application of the Cycle Elimination rule,
  and select one constant elimination problem C'_{new}, which represents one possibility to track the constant elimination problem C_{new} back to MS_I
  then $MS_s := MS_I$ and $C_s := C'_{new} \cup C_{old}$.
- Solve the constant elimination problem C_s related to a $MS_s = \{ME_{si} \mid 1 \le i \le n\}$
- Let σ_s be a solution of C_s

  If $t = \sigma_B ME_{si}^B$ is a (significant) constant, then
  $ME'^V_{si} := ME^V_{si}$, $ME'^F_{si} := ME^F_{si} \cup t$ and $ME'^B_{si} := \{\}$,
  else $ME'^V_{si} := ME^V_{si}$, $ME'^F_{si} := ME^F_{si}$ and $ME'^B_{si} := t$.

The first important subroutine in cycle elimination is **cycle detection**. It determines if the cycle elimination rule has to be applied. Also we need the set of cycles to compute the constant elimination problems, i.e. the constants to be eliminated to resolve the cycles. For explanation of this routine, which is based on an algorithm by H. Weinblatt [Wei 72] we need the following definitions:

A directed graph may be described as a set of points (vertices) together with a set of directed line segments (arcs) such that each arc connects precisely two vertices, "originating" on one vertex and "terminating" on the other. A path connecting one vertex, v_0, to another, v_n, is an ordered collection of vertices $v_0v_1...v_n$ such that for every v_iv_{i+1} there exists an arc a_i, which originates on v_i and terminates on v_{i+1} (Note that this arcless representation is unambiguous only for graphs, which contain no parallel arcs). A path is simple if it encounters no vertex twice, and cyclic if it originates and terminates on the same vertex. The vertices of a cyclic path without regard to its endpoints form a cycle. A cyclic path is simple-cyclic, if it encounters one vertex twice (the one it originates and terminates), and no other vertex more than once. A cycle is simple if it corresponds to a simple-cyclic path.

For our problem, the breaking of all cycles, it is sufficient to compute the list of **simple cycles**, as obviously every other cycle is composed of simple cycles. Those simple cycles are computed by the algorithm of H. Weinblatt [Wei 72]. The algorithm begins with a preprocessing of the graph, which reduces the size of the graph by eliminating those vertices, which cannot belong to any cycle. Preprocessing starts with the removal of any vertices, on which no arc terminates, together with the arcs originating on these vertices. This step is recursively applied. Then all vertices, on which no arc originate, are removed together with the arcs terminating on these vertices. This step is also recursively applied. After this preprocessing the algorithm selects one of the remaining vertices as a startpoint and begins to examine a path emanating from the vertex. A path is explored, until a vertex is encountered, which has been previously examined. Then the algorithm tracks back to the last branch point (a point, where more than one arc originates) and chooses another path. If no branching point exists then another starting point is selected. The algorithm terminates, if all vertices of the graph have been examined. The cycles are determined in two fashions. The signal for existence of a cycle is, when we encounter a vertex, which we have reached before. If the currently examined path still contains this vertex, then there is one cyclic path: the one, which is a subpath of the examined path (Example: current path = $v_2v_1v_5v_1$ with cycle = $v_1v_5v_1$). The other possibility is the construction of the cycle from subpaths of previously discovered cycles with a subpath of the currently processed path, which terminates on the last vertex.

Example: $cycle_1 = v_1v_2v_1$; $cycle_2 = v_2v_3v_2$; current path = v_1v_3 reencounters v_3
==> new cycle $v_1v_3v_2v_1$

The advantage of this algorithm is that every individual arc is examined just once and only once. Sometimes we will have to examine parts of these cycles (but not the individual arcs) a number of times. For the validity of the algorithm see Weinblatt's paper [Wei 72].

This algorithm is used to compute all simple cycles of the multi-equation system. The vertices of the graph are the auxiliaries of the system. An arc originates on a vertex v_1 and terminates on another vertex v_2, if the auxiliary corresponding to v_1 is member of one theory component of the multi-equation ME_j and the auxiliary corresponding to v_2 occurs in a compound term, which is also member of ME_j. If we determine at least one simple cycle, then the rule **cycle elimination** has to be applied. From the simple cycles we can easily compute the set of possibilities, which make this graph cycle free, i.e. break all cycles. Cycles are broken by removing arcs from the graph, i.e. by eliminating auxiliaries. As only the auxiliary constants in the boolean part can be eliminated, we can only break cycles after every second vertex.

Our **constant elimination** problems C have the following structure:

$$C := \{c_i \notin \text{Constants}(t_{ij}) \mid 1 \le i \le n;\ 1 \le j \le m\},$$

where c_i are different free constants, in fact auxiliary constants, and t_{ij} are boolean compound terms. The set of solutions for C is the set

$$U_{BR}(C) := \{\sigma \mid \exists\ t'_{ij} =_{BR} \sigma t_{ij} \text{ and } c_i \notin \text{Constants}(t'_{ij}) \text{ for } 1 \le i \le n;\ 1 \le j \le m\}.$$

A **complete set of constant eliminators** $cU_{BR}(C)$ is a set of substitutions such that

(1) $cU_{BR}(C) \subseteq U_{BR}(C)$

(2) for every $\theta \in U_{BR}(C)$ there exists $\sigma \in cU_{BR}(C)$ such that $\sigma \le_{BR} \theta[\text{Variables}(C)]$.

A complete set is called **minimal** or **most general**, iff additionally

(3) $\forall\ \sigma,\tau \in cU_{BR}(C)\ \ \tau \le_{BR} \sigma[\text{Variables}(C)] \Rightarrow \tau = \sigma$ (minimality).

We have to compute the most general set for the elimination rule. M Schmidt-Schauß [Schm 88] offered the following method to solve constant elimination problems in a Boolean ring. Let the constant elimination problem $C = \{c_i \notin \text{Constants}(t_{ij}) \mid 1 \le i \le n;\ 1 \le j \le m\}$. Let C_0 be the set of all constants to be eliminated $C_0 = \{c_i \mid 1 \le i \le n\}$ and let V_0 be the set of all variables occurring in the terms of the problems $V_0 = \text{Variables}\{t_{ij} \mid 1 \le i \le n;\ 1 \le j \le m\} = \{z_k \mid k = 1, \ldots, K\}$. Let D be the set of all possible products of elements in C_0, $D = \{c_{i1} * c_{i2} * \ldots * c_{ij} \mid \{i1, \ldots, ij\} \subseteq \{1, \ldots, n\}$. D contains the element "1" as empty product and hence set D generated by C_0 has 2^n elements. We try a 'general' substitution with $\text{Dom}(\sigma) = V_0$. A general representation is $\sigma z_k = \Sigma\{y_{k,d} * d \mid d \in D\}$, where $y_{k,d}$ are different new variables and stand for terms not containing constants from C_0. If we apply σ to C then we get $\sigma t_{ij} = \Sigma\ \{t_{i,j,d} * d \mid d \in D\}$, where $t_{i,j,d}$ is a term not containing constants from C_0. The unification problem Γ_C corresponds to C as follows:

$$\Gamma_C := \{t_{i,j,d} = 0 \mid d \in D, \text{ where } c_i \text{ is a factor of } d,\ 1 \le i \le n;\ 1 \le j \le m\}.$$

This unification problem does not contain constants from C_0 and is to be solved without these constants. The obtained most general unifier can be transformed into a solution of the constant elimination problem C. Since the Boolean ring is unitary unifying, there is at most one general constant eliminator necessary.

Example: $MS_0 = \langle x = f(x * y) \rangle$.

Let BR = <+, *, 0, 1> be a Boolean ring, f a unary free function symbol, x, y are significant variables, c is an auxiliary constant and z, y_B, y_{B1}, y_{B2} are auxiliary variables.

MS after identification: $\langle \{ y \}^V = \{\}^F = \{ y_B \}^B$

$\{\}^V = \{ z \}^F = \{ y_B * c \}^B$

$\{ x \}^V = \{ f(z) \}^F = \{ c \}^B \rangle$

cycle: $z\ y_B\ z$

elimination problem: $\{c \notin y_B * c\}$

Let $y_B := y_{B1} + y_{B2} * c$, where y_{B1} and y_{B2} are variables that stand for terms not containing c. The problem to be solved is $c \notin y_{B1} * c + y_{B2} * c$, which is equivalent to $y_{B1} + y_{B2} = 0$, as c is a free constant and y_{B1}, y_{B2} do not contain c. The unique solution is $y_{B1} = y_{B2}$. So the constant eliminator is $\{y_B \leftarrow y_{B1} + y_{B1} * c\}$.

Constant elimination: $\langle \{ y \}^V = \{\}^F = \{ y_{B1} + y_{B1} * c \}^B$
$\{\}^V = \{ z = 0\}^F = \{\}^B$
$\{ x \}^V = \{ f(z) \}^F = \{ c \}^B \rangle$

F-Unification: $\langle \{ y \}^V = \{\}^F = \{ y_{B1} + y_{B1} * c \}^B$
$\{ x \}^V = \{ f(0) \}^F = \{ c \}^B \rangle$

We improved M. Schmidt-Schauß method for constant elimination mainly in two ways:

1) Is there a subproblem $c_a \notin t_{ij}$ with $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$ in the constant elimination problem C with $\text{Variables}(t_{ij}) = \emptyset$, then the constant elimination problem is not solvable. No substitution σ can change the structure of t_{ij}, as no variables, especially no variables of the domain of σ, occur in t_{ij}.

2) We split the constant elimination problem into independent subproblems, i.e. variable disjoint subproblems: $C = \{C^1, \ldots, C^N\}$. Let $(c_{i'} \notin t_{i'j'}) \in C^r$ and $(c_{i''} \notin t_{i''j''}) \in C^s$ with $r, s \in \{1, \ldots, N\}$. If $\text{Variables}(t_{i'j'}) \cap \text{Variables}(t_{i''j''}) \neq \emptyset \Rightarrow r = s$.

This method is advantageous for such cases, where subproblems have different sets of constants to be eliminated $C_0^1, \ldots, C_0^N$. As we have smaller sets of possible products of elements of C_0^i, we introduce less new variables to our systems. Also we have to solve simpler unification problems.

Example: $C = \{a \notin x + aby, c \notin av + adc\}$

Let BR = <+, *, 0, 1> be a Boolean ring, a, d, c are constants and x, y, v, x_1, x_2, x_3, x_4, y_1, y_2, y_3, y_4, v_1, v_2, v_3, v_4 are variables.

MSS) $\sigma = \{x \leftarrow x_1 + ax_2 + cx_3 + acx_4, y \leftarrow y_1 + ay_2 + cy_3 + acy_4, v \leftarrow v_1 + av_2 + cv_3 + acv_4\}$
problems: $x_2 + by_1 + by_2 = 0$; $x_4 + by_3 + by_4 = 0$; $v_3 + v_4 + d = 0$
solution: $\{x_2 \leftarrow by_1 + by_2, x_4 \leftarrow by_3 + by_4, v_3 \leftarrow v_4 + d\}$
transformed terms: $x_1 + cx_3$ and $av_1 + av_2$

BCR) $C^1 = \{a \notin x + aby\}$ and $C^2 = \{c \notin av + adc\}$
$\sigma = \{x \leftarrow x_1 + ax_2, y \leftarrow y_1 + ay_2, v \leftarrow v_1 + cv_2\}$
problems: $x_2 + by_1 + by_2 = 0$; $v_2 + d = 0$
solution: $\{x_2 \leftarrow by_1 + by_2, v_2 \leftarrow d\}$
transformed terms: x_1 and av_1

Sometimes new cycles are introduced to a multi-equation system, when the cycles already existing in the system are removed by application of the cycle elimination rule. Therefore recursive cycle elimination is necessary:

Definitions:

Let MS_I be the multi-equation system after identification, i.e. before any application of a constant eliminator computed by the cycle elimination rule. Let MS_C be the multi-equation system to be examined and C_{OLD} is the already solved constant elimination problem of MS_I, which lead to the creation of MS_C. $C_{OLD} = \{c_i \notin \text{Constants}(t_{ij}) \mid 1 \le i \le n;\ 1 \le j \le m\}$ with $t_{ij} \in MS_I^B$. The reference list R connects the currently valid multi-equation system MS_C with MS_I:

$R = \{(t'_r, t_r) \mid 1 \le r \le n$ with $t_r \in MS_I^B$, $t'_r \in MS_C^B$ and $t'_r = \sigma t_r$ with σ the solution of $C_{OLD}\}$.

The function Refer determines the list of possible original terms for a term x' occurring in the currently valid system MS_C: $\text{Refer}(x') = \{x \mid (x', x) \in R\}$.

The main idea for the solution of recursive cycle elimination problems is **backtracking**:

We have a multi-equation system MS_C, which is cyclic. Then we can determine the constant elimination problem $C_{NEW} = \{c_i \notin \text{Constants}(t'_{ij}) \mid 1 \le i \le n;\ 1 \le j \le m\}$ with $t'_{ij} \in MS_C^B$, whose solution applied to MS_C would remove all determined cycles. Yet we do not remove the cycles in MS_C, but in the original system MS_I. As we use MS_I in the cycle elimination rule, we have to backtrack the elimination problem C_{NEW}, which eliminates constants in compound terms of MS_C, to an elimination problem C'_{NEW}, which eliminates constants in the corresponding original compound terms of MS_I. This backtracking is achieved with the function Refer:

$C'_{NEW} = \{c_i \notin \text{Constants}(t_{rij}) \mid t_{rij} \in \text{Refer}(t'_{ij});\ 1 \le i \le n;\ 1 \le j \le m\}$

Note that more than one back tracked elimination problem may exist for an elimination problem C_{NEW}, as the application of the solution of the elimination problem C_{OLD} to MS_I might have merged several original compound terms of MS_I to a single compound term of MS_C. The backtracked constant elimination problem has to be combined with the previously solved constant elimination problem C_{OLD}: $C_S := C_{OLD} \cup C'_{NEW}$.

Now having computed the problem C_S for the cycle elimination rule we can solve it and apply the solution to the system MS_I. If another cyclic system is computed, then the recursive cycle elimination process is reentered.

We use this backtracking method out of termination reasons. What would happen, if we had decided to solve only the constant elimination problems directly determined by the system? In that case the algorithm might not terminate, as every application of the solution of an elimination problem to the system creates a new system, which then again might have a new cycle. Our method on the other hand

terminates, as we always use the same initial system MS_I and only increase the constant elimination problem (cf. [BJS 88]). We add subproblems of the form "$c_{i'} \notin$ Constants $(t_{i'j})$" to the already existing problem C_{OLD}. This obviously terminates, as the set of compound term in the boolean part of MS_I and set of the constants used in $MS_I{}^B$ are finite.

4.2 Improved Unification Algorithm

There are several possibilities to improve the performance of the combination algorithm. Two of the implemented improvements, which are not based on some technical tricks, are presented in this section. Further interesting possibilities, which I have not used in the algorithm, are discussed with their advantages and outweighing disadvantages in Section 4.3.

The improvements presented here are employed in the preprocessing part of the combination algorithm; more exactly after renaming and before identification. It is the procedure, where we alternate syntactic (free) and Boolean ring unification, until the free part and the boolean part of a multi-equation consists of not more than one element.

We know that Boolean ring unification requires more computation time than free unification. Therefore it is our aim to reduce the search space of Boolean ring unification problems. Previously this was achieved by executing free unification (F) before Boolean ring unification (B) in the alternating application of unification: (FB)*

The first of our upgradings consists of the construction of a **preunification** step (B^P) for Boolean ring unification problems. This preunification step is executed in alternation with free unification, until neither of them can be applied. Then the alternation of free and Boolean ring unification is executed. So the new sequence is $(FB^{P*})^*(BF)^*$. In the preunification step we solve Boolean ring unification problems, which involve only constants and variables on toplevel (e.g.: x = a, x = y). For such problems it is not necessary to employ Boolean ring unification as presented in Chapter 3. Those problems are handled, as if they were problems in a empty theory without function symbols. Of course the computed unifiers are applied to the multi-equation system and our preunification step is performed, until no more problems of this kind occur. Just as in the BR-Unification step we have to take into account that the free constants are stored in the free part in a multi-equation.

Let $ME_i^P = \{v \mid v \in ME_i^B \text{ is a variable}\} \cup \{c \mid c \in ME_i^F \text{ is a constant}\}$.

Rule: Preunification.

$\{ME_i^V = ME_i^F = ME_i^B \mid 1 \le i \le n\} \Longrightarrow \{ME_i^V = ME'^F_i = ME'^B_i \mid 1 \le i \le n\}$,

if there is a $|ME_i^P| > 1$.

Let σ be a free mgu of $\langle\{ME_i^P \mid 1 \le i \le n\}\rangle_F$.

If there is a $t \in \sigma ME_i^B$ that is a (significant) constant, then

$ME'^F_i := ME_i^F \cup \{t\}$ and $ME'^B_i := \sigma ME_i^B \setminus \{t\}$,

else $ME'^F_i := ME_i^F$ and $ME'^B_i := \sigma ME_i^B$.

Example for preunification:

a, b are free constants, x, y variables, f unary free function symbol, B = <+, *, 0 , 1> Boolean ring $MS_0 = \langle f(x) = f(y) = f(b), x + y = a \rangle$ After application of the rules Partitioning, Unfolding, Renaming, F-Unification we get the following system:

$\langle \{ a \}^F = \{ x_B + y_B \}^B \quad {}^{B^P} \quad \langle \{ a, 0 \}^F \quad {}^{F}$

$\{ x, y \}^V = \{ b \}^F = \{ x_B, y_B \}^B \rangle \Rightarrow \quad \{ x, y \}^V = \{ b \}^F \rangle \Rightarrow$ ↯

The important advantage of preunification is, that we can narrow down the search space of Boolean ring unification. The number of variables, which effects exponentially the complexity of Boolean ring unification, is reduced: Variables are eliminated either by substitution with another variable already occurring in the system or even better with a constant.

The second improvement is connected with the identification process. In the identification process we get a specific identification by abstracting compound free terms of multi-equations to be joined with the same abstraction constant, while all other compound free terms are abstracted with pairwise different constants. Some identifications of compound free terms are not possible, especially those which cannot be unified. Definite members of this set are free terms, which are different ground terms. Already during preprocessing we use this property to perform **preabstraction** of ground terms. This means different ground free terms are abstracted with pairwise different constants. These abstractions are executed, provided they have not been performed earlier, before the Boolean ring unification step of preprocessing. Later in the combination algorithm, when we consider the possible identifications, we have to remember to combine the multi-equations containing ground free terms with the multi-equations containing other compound free terms (Example: Let ME_1, ME_2 be multi-equations with free ground terms, ME_3, ME_4 be multi-equations with other compound free

terms ⇒ Identifications to be tested join these multi-equations: none, ME_1 with ME_3, ME_1 with ME_4, ME_2 with ME_3, ME_2 with ME_4, ME_3 with ME_4, ME_1 with ME_3 and ME_4, ME_2 with ME_3 and ME_4, ME_1 with ME_3 and ME_2 with ME_4, ME_1 with ME_4 and ME_2 with ME_3).

Let NEW be set of multi-equations ME_i of MS, which have a not yet abstracted compound ground term in the free part ME_i^F and a non-empty boolean part.

Rule: Preabstraction.

$$\{ME_i^V = ME_i^F = ME_i^B \mid 1 \le i \le n\} \Longrightarrow \{ME_i^V = ME_i^F = ME'^B_i \mid 1 \le i \le n\},$$

if set NEW is not empty.

Let c_i, c_j be different abstraction constants, if $i \neq j$.

If $ME_i \in$ NEW then $ME'^B_i := \{c_i\} \cup ME_i^B$ else $ME'^B_i := ME_i^B$.

Example for preabstraction:

a, b are free constants, x, y variables, f, g unary free function symbols, B = <+, *, 0 , 1> Boolean ring, v_B an abstraction variables, c, d abstraction constants

$MS_0 = \langle g(x) = g(a), f(b) = y + f(x) \rangle$ After application of the rules Partitioning, Unfolding, Renaming and F-Unification we get the following system:

$\langle \{ f(b) \}^F = \{ y_B + v_B \}^B$ (Preabstraction) $\langle \{ f(b) \}^F = \{ c, y_B + v_B \}^B$ (BR-Unification, $\langle x = a$

$\{ f(a) \}^F = \{ v_B \}^B$ ⇒ $\{ f(a) \}^F = \{ d, v_B \}^B$ Postprocessing) $y = f(b)+f(a) \rangle$

$\{ x \}^V = \{ a \}^F$ $\{ x \}^V = \{ a \}^F$ ⇒

$\{ y \}^V = \{ y_B \}^B \rangle$ $\{ y \}^V = \{ y_B \}^B \rangle$

Just like in the previous improvement we reduce Boolean ring problems by decreasing the variable set or we get smaller problems by unifying the abstraction constant with complex Boolean ring term. In the course of this upgrading we have also introduced a constraint for identification into the combination algorithm: Free ground terms cannot be identified.

Summarizing Section 4.2 we state that our improvements can make unification problems solvable in the combination of Boolean ring theory with free function symbols without Boolean ring unification. Sometimes the problems can be solved before the identification and cycle elimination processes.

4.3 Theoretical Possibilities

4.3.1 Pure Boolean Ring Multi-equations

When we solve a multi-equation $ME = \langle t_1 = \ldots = t_n \rangle_{BR}$ in a Boolean ring BR, we would normally proceed as follows. We interpret the multi-equation as a set of equations $\langle t_1 = t_2, \ldots, t_{n-1} = t_n \rangle_{BR}$. Then we solve one equation after another having applied the composed previously acquired unifiers in advance; always provided that the equation system was solvable so far. This procedure involves several applications of unifiers to equations as well as composition of unifiers. Therefore it would be favourable, if we can transform the multi-equation into a single equation $\langle 0 = t \rangle_{BR}$. How such a term is structured, is explained by the next Theorem 4.3.1. In fact the term is a sum of products, which are the elements of the powerset of the set of equation elements without this set itself and the empty set.

Theorem 4.3.1:
Let $ME = \langle t_1 = \ldots = t_n \rangle_{BR}$ be a multi-equation in a Boolean ring BR.
Then $\langle 0 = \Sigma_{S \subset N} \Pi_{s \in S} t_s \rangle_{BR}$, where $N = \{1, \ldots, n\}$, is an equivalent equation.

Proof by induction:
Let $ME = \langle t_1 = t_2 \rangle_{BR}$. $t_1 = t_2 \Leftrightarrow 0 = t_1 + t_2$.
Now $ME' = \langle t_1 = \ldots = t_n = t_{n+1} \rangle_{BR}$ with $N' = N \cup \{n+1\}$.

$$
\begin{aligned}
& t_1 = \ldots = t_{n+1} \\
\Leftrightarrow\ & t_1 = \ldots = t_n \quad \text{and} \quad t_1 = t_{n+1} \\
\Leftrightarrow\ & 0 = \Sigma_{S \subset N} \Pi_{s \in S} t_s \quad \text{and} \quad 0 = t_1 + t_{n+1} \\
\Leftrightarrow\ & 0 = (t_1 + t_{n+1}) + \Sigma_{S \subset N} \Pi_{s \in S} t_s + ((t_1 + t_{n+1}) * \Sigma_{S \subset N} \Pi_{s \in S} t_s) \\
\Leftrightarrow\ & 0 = t_1 + t_{n+1} + t_1 * \Sigma_{S \subset N} \Pi_{s \in S} t_s + (1 + t_{n+1}) * \Sigma_{S \subset N} \Pi_{s \in S} t_s \\
\Leftrightarrow\ & 0 = t_1 + t_{n+1} + \Sigma_{S \subset N} t_1 \Pi_{s \in S} t_s + \Sigma_{S \subset N', S \neq \{n+1\}, S \neq N} \Pi_{s \in S} t_s \\
\Leftrightarrow\ & 0 = t_1 + t_{n+1} + \Sigma_{S \subset N, 1 \in S} t_1 \Pi_{s \in S} t_s + \Sigma_{S \subset N, 1 \notin S} t_1 \Pi_{s \in S} t_s + \Sigma_{S \subset N', S \neq \{n+1\}, S \neq N} \Pi_{s \in S} t_s \\
\Leftrightarrow\ & 0 = t_1 + t_{n+1} + \Sigma_{S \subset N, 1 \in S} \Pi_{s \in S} t_s + \Sigma_{S \subset N, 1 \in S, S \neq \{1\}} \Pi_{s \in S} t_s + \Pi_{s \in N} t_s + \Sigma_{S \subset N', S \neq \{n+1\}, S \neq N} \Pi_{s \in S} t_s \\
\Leftrightarrow\ & 0 = t_1 + t_{n+1} + t_1 + \Pi_{s \in N} t_s + \Sigma_{S \subset N', S \neq \{n+1\}, S \neq N} \Pi_{s \in S} t_s \\
\Leftrightarrow\ & 0 = \Sigma_{S \subset N'} \Pi_{s \in S} t_s \qquad \blacksquare
\end{aligned}
$$

There are at least two special cases for this theorem, which seem interesting. The first one restricts the solution enormously, while the second one shows a remarkable resemblance to the theorem itself. This, of course, allows an easy insertion of those two constraints into the algorithm for the theorem above.

Lemma 4.3.2:

Let $ME = \langle t_1 = \ldots = t_n \rangle_{BR}$ be a multi-equation in a Boolean ring BR. $N = \{1, \ldots, n\}$, $N' = N \setminus \{i\}$.

a) If there is a t_i, which equal to 1, in the multi-equation ME, then
$\langle 1 = \prod_{s \in N'} t_s \rangle_{BR}$ is equivalent to ME.

b) If there is a t_i, which is equal to 0, in the multi-equation ME, then
$\langle 0 = \sum_{S \subseteq N'} \prod_{s \in S} t_s \rangle_{BR}$ is equivalent to ME.

Proof:

a) $t_1 = \ldots = t_n \Leftrightarrow 0 = \sum_{S \subset N} \prod_{s \in S} t_s \Leftrightarrow 0 = \sum_{S \subset N, i \in S} \prod_{s \in S} t_s + \sum_{S \subset N, i \notin S} \prod_{s \in S} t_s$
$\Leftrightarrow 0 = 1 + \sum_{S \subseteq N'} \prod_{s \in S} t_s + \sum_{S \subset N'} \prod_{s \in S} t_s$
$\Leftrightarrow 0 = 1 + \prod_{s \in N'} t_s + \sum_{S \subset N'} \prod_{s \in S} t_s + \sum_{S \subset N'} \prod_{s \in S} t_s$
$\Leftrightarrow 1 = \prod_{s \in N'} t_s$ ■

b) $t_1 = \ldots = t_n \Leftrightarrow 0 = \sum_{S \subset N} \prod_{s \in S} t_s \Leftrightarrow 0 = \sum_{S \subset N, i \in S} \prod_{s \in S} t_s + \sum_{S \subset N, i \notin S} \prod_{s \in S} t_s$
$\Leftrightarrow 0 = 0 + \sum_{S \subset N'} \prod_{s \in S} t_s$
$\Leftrightarrow 0 = \sum_{S \subset N'} \prod_{s \in S} t_s$ ■

Of course, we would like to transform a whole multi-equation system into an equation as well. This can be achieved by transforming the multi-equation system into an equivalent set of equations. Then this set of equations can be combined to a single multi-equation, which again can be transformed into an equivalent equation. This procedure is expressed by the following theorem.

Theorem 4.3.3:

Let $\Gamma = \{ME_1, \ldots, ME_m\}$ be a multi-equation system in a Boolean ring BR
with $ME_j = \langle t_{j1} = \ldots = t_{jn} \rangle_{BR}$.
Then $\langle 0 = \sum_{R \subset M} \prod_{r \in R} \tau_r \rangle_{BR}$ with $M = \{1, \ldots, m\}$ and $\tau_r = \sum_{S \subset N} \prod_{s \in S} t_{rs}$
is an equivalent equation.

Proof:

By Theorem 4.3.1: $ME_j = \langle t_{j1} = \ldots = t_{jn} \rangle_{BR} \Leftrightarrow \langle 0 = \sum_{S \subset N} \prod_{s \in S} t_{js} \rangle_{BR}$
As every multi-equation of the system Γ can be expressed by a term equal to 0, we can transform these equations to the multi-equation $\langle 0 = \sum_{S \subset N} \prod_{s \in S} t_{1s} = \ldots = \sum_{S \subset N} \prod_{s \in S} t_{ms} \rangle_{BR}$. Here we apply Lemma 4.3.2.b and transform it to $\langle 0 = \sum_{R \subset M} \prod_{r \in R} \tau_r \rangle_{BR}$ with $\tau_r = \sum_{S \subset N} \prod_{s \in S} t_{rs}$. ■

Having discovered these possibilities to transform multi-equations and even multi-equation systems into a single equation, we had to discuss the possible applications. Although no composition of

unifiers and no application of unifiers to equations are necessary, there are disadvantages:

1) The construction of term t involves a large simplification process.
2) The term t to be constructed might have an enormous size, which has an exponential effect on the complexity of the final problem $\langle 0 = t \rangle_{BR}$.
3) We loose the information, that two terms are equal. Such an equation $\langle t_i = t_j \rangle_{BR}$ might restrict the search space enormously, especially if one is a constant.

Therefore we came to the conclusion to use the transformations, only if terms of the equations have a similar constant and variable set. This causes the simplification to be more effective, as there is higher probability that equal (removable) subterms occurs. Thereby we may obtain a smaller term t for the final equivalent equation.

Although we do not employ the transformation methods, we make use of another constraint. We sort our multi-equation such that we start with simple problems. Thereby we can restrict the search space for the following sequel of equations of this multi-equation. Simple problems are mainly those involving constants.

4.3.2 Identification

Let us recapture the main ideas of the identification process for the combination algorithm. During identification we join some multi-equations containing compound free terms. This is achieved in our version of the algorithm by abstracting the compound free terms of multi-equations to be joined with the same abstraction constant, while all other compound free terms are abstracted with pairwise different constants. From the view of Boolean ring unification, one can regard the different multi-equation systems based on the different identification possibilities as a set of Boolean unification problems (problem elements are like $\langle \text{abstractionconstant}_i = \text{booleanringterm}_j \rangle_{BR}$), which differ only in the used abstraction constants. Therefore we can use multi-equation system MS1 to compute multi-equation system MS2, which identifies the same compound free terms as MS1 plus the one from multi-equation ME_i with the one from multi-equation ME_j. This achieved by using almost all abstraction constants of MS1 but the abstraction constant of ME_i for ME_j. In other word we could apply a transformation $\lambda = \{c_i \leftarrow c_j\}$ to MS1 to obtain MS2.

Here we see a possibility to save some time in the Boolean ring unification branch. Provided we store the Boolean ring unifiers of the identification problems and we now have an identification problem, which joins a superset of multi-equations of an stored identification problem, we could apply the transformation to the unifier instead of to the system. Thereby we would save considerable time, as we do not need to solve the Boolean ring problem again. Yet we have to prove, that

performing the transformation $\lambda = \{c_i \leftarrow c_j\}$, where c_i, c_j have to be elements of the constants of the unification problem, to the problem itself is equivalent to performing the transformation to the unifiers of the problem; of course provided the problem was previously solvable. This is accomplished by the following theorem.

Theorem 4.3.4:

Let t be a term of a Boolean ring BR, a, b $\in$ constants (t), $\lambda = \{a \leftarrow b\}$ be a transformation, $t' = \lambda(t)$ and $\tau = \lambda(\sigma)$ [Variables (t')].

σ is a most general unifier of $\langle t = 0 \rangle_{BR}$, iff τ is a most general unifier of $\langle t' = 0 \rangle_{BR}$.

Proof:

First we have to prove that, if σ is a unifier of $\langle t = 0 \rangle_{BR}$, then $\tau = \lambda(\sigma)$ is a unifier of $\langle t' = 0 \rangle_{BR}$.

$\sigma t =_{BR} 0 \Rightarrow \sigma t =_{BR \cup \{a=b\}} 0$

$\Rightarrow^{1)} \tau t' =_{BR \cup \{a=b\}} 0$ | $^{1)}$ $\sigma =_{\{a=b\}} \tau$ and $t =_{\{a=b\}} t'$

$\Rightarrow^{2)} \tau t' =_{BR} 0$ | $^{2)}$ a does not occur in $\tau t'$

Now by Löwenheim [Löw 08] the most general unifier of $\langle t = 0 \rangle_{BR}$ is

$\sigma = \{x \leftarrow x' + \gamma t\ (x' + s_x) \mid x \in$ Variables (t)$\}$ and the most general unifier of $\langle t' = 0 \rangle_{BR}$ is

$\tau = \{x \leftarrow x' + \gamma t'\ (x' + s'_x) \mid x \in$ Variables (t')$\}$.

$\lambda(\sigma)$ [Variables (λt)] = $\{x \leftarrow \lambda(x' + \gamma t\ (x' + s_x)) \mid x \in$ Variables (t)$\}$ [Variables (λ(t)]

$=_{BR} \{x \leftarrow x' + \gamma\lambda(t\ (x' + s_x))) \mid x \in$ Variables (λ(t))$\}$

$=_{BR} \{x \leftarrow x' + \gamma t'(x' + \lambda(s_x))) \mid x \in$ Variables (t')$\}$

$=_{BR} \tau$ | (as $\lambda(s_x) =_{BR} s'_x$ for every $x \in$ Variables (t'))

■

As by Theorem 4.3.3 any Boolean ring multi-equation system can be transformed in a single equation, we could used Theorem 4.3.4 in the following way:

Start the identification process with smallest identification possible (none joined) and continue up to the biggest identification (all joined together). For each identification ensure, that any subset of identification, whose solution could be used, has already been computed.

Regretfully during the course of this thesis several reasons were discovered not to employ Theorem 4.3.4:

1) Joining of multi-equations involves also unification in the free part. This unification is faster than the Boolean ring unification. Thereby we start the test of the identifications by unifying the free part. This means that our improvement is not fully applicable, as not all identified multi-equation systems reach the Boolean ring unification step.

2) The theorem restricts the substitution to the variables occurring in the transformed term. Therefore we have to apply the transformation also to the term to determine the variables necessary for the domain.

3) The recursive character of identification complicates the procedure, as the several levels of identifications lead to several levels of transformations.

4) A great amount of storage is needed to store the unifiers and the accessors to the unifiers.

Because of these reasons we did not employ Theorem 4.3.4 in the identification process.

4.4 Summary

The combination algorithm with its improvements is divided into four phases:

Phase 1: Preprocessing

Task: Transformation of multi-equation system into separated unfolded normal form
Rules: Partitioning, Unfolding, Renaming,
F-Unification, Preunification, BR-Unification, Preabstraction,
Reduction Rules.

Phase 2: Identification

Task: Transformation of multi-equation system into sequentially solved form (allowing cycles)
Rules: Identification, F-Unification, BR-Unification, Constant Abstraction,
Reduction Rules.

Phase 3: Cycle Elimination

Task: Transformation of multi-equation system into sequentially solved form (no cycles)
Rules: Cycle Elimination, F-Unification,
Reduction Rules.

Phase 4: Postprocessing

Task: Transformation of multi-equation system into solved form
Rules: $\text{Elimination}_{B.c}$, $\text{Elimination}_{B.v}$, Elimination_{F},
Reduction Rules.

The Reduction Rules are Trivial Multi-equations, Auxiliaries, Merge and Equal Terms.

4.5 History and Related Algorithms

There are several investigations covering the field "Combination Of Unification Algorithms" for disjoint theories. Most of these approaches have also some restrictions on theories they can be applied to. K. Yelick's [Yel 85] algorithm is based upon the variable abstraction method, which was introduced first by M.E. Stickel [Sti 81] and F. Fages [Fag 84] for problems of the AC theory. For her approach both of the theories have to fulfill the premises of regularity and collapse freeness. C. Kirchner [Kir 85] tackles the problem computing decomposition schemes (called mutations) from the axioms for the theories. We could say, that he does not combine the unification algorithms of the two theories themselves, but modified versions of them. His work is restricted to theories, which are simple (cycle free). A. Herold's algorithm [Hero 86] is based upon the constant abstraction method, which was employed by M. Livesey and J. Siekmann for their AC unification algorithm [LS 78]. Herold's approach is limited to regular and collapse free theories. E. Tidén [Tid 86] again uses variable abstraction for his algorithm, which is restricted to collapse free theories. He was the first one to introduce eliminators. A. Boudet, J.-P. Jouannaud and M. Schmidt-Schauß [BJS 88] have developed an algorithm, which employs variable abstraction. In addition they need a variable elimination algorithm for one of the theories and a constant matching algorithm. Also this theory has to be cycle free. M. Schmidt-Schauß [Schm 88] has created an algorithm, which has no restrictions at all on the theories. The main procedures he uses are unfolding, constant abstraction, identification and theory labelling. This algorithm is currently being implemented by M. Tepp at the University of Kaiserslautern [Tep 88]. Our algorithm of Chapter 4 is, as mentioned earlier, based on another approach by Schmidt-Schauß. There Schmidt-Schauß restricts one theory to be simple. This allows a creation of an improved algorithm, because the properties of a simple theory include, that cyclic systems of equations are not solvable in such a theory. The algorithms differ mainly in that we do not need theory labelling and that we can use a simpler identification step. For detailed information see [Schm 88]. As pointed out, there are three different basic approaches: **variable abstraction,**

constant abstraction and **decomposition**. In the table below we try to pinpoint this and the restrictions on two combined theories E and F.

Table 4.5.1: Comparison of unification algorithms in the combination of theories

AUTHORS	PRIMARY METHODS	THEORY RESTRICTIONS
Schmidt-Schauß	Variable & Constant Abstraction Theory Labelling	No Restrictions
Schmidt-Schauß	Variable & Constant Abstraction	E arbitrary & F cycle free
Boudet et al	Variable Abstraction	E arbitrary & F cycle free
Tidén	Variable Abstraction	E & F collapse free
Herold	Constant Abstraction	E & F regular E & F collapse free
Yelick	Variable Abstraction	E & F regular E & F collapse free
Kirchner	Decomposition	E & F cycle free

Earlier we have mentioned, that the Boolean ring theory is neither regular nor cycle free nor collapse free. Therefore at the moment only the first three different algorithms can be used. We have already disclosed, that for our specific application the use of the more specialized algorithm of Schmidt-Schauß is superior, as one can readily make use of the cycle free properties of the free theory. This is supported by data, which has been exchanged with Tepp, who is implementing this other method. Therefore the only real competition is Boudet et al. In both algorithms several similar problems have to be solved. They both need an algorithm for unifying pure free terms as well as for unifying pure Boolean ring terms. At the moment Boudet et al employ the Martin/Nipkov method for the Boolean rings. In opposition to them we tend to the Büttner/Simonis approach for the reasons mentioned in Chapter 3 (smaller unifiers). Also both combination algorithms need similar elimination algorithms to break theory overlapping cycles. But the most important difference is in the abstraction method: Variable & Constant Abstraction <===> Variable Abstraction. We argue that our usage of constant abstraction is superior, as in a Boolean ring variables increase the solution space decisively. Therefore we assume, that our algorithm would be the better choice, what hopefully will be supported by the runtime data, which we expect to exchange with the Boudet et al.

5. DATA STRUCTURES AND SPECIAL FEATURES

In this chapter we give some information about the realization of the unification algorithms. In particular we explain the datastructures selected for the implementation. Also we describe the top-level functions available to the user as well as some lower-level functions crucial to the course of the algorithms. Finally we present a test environment, which has been used to compare and test the three pure unification algorithms and test and trace the combination algorithm. This environment can also be used to develop and run test examples in any instantiation of Boolean ring theory with or without free function symbols.

The algorithms have been implemented Common Lisp and run on Symbolics Machines 36xxx. The language chosen is the Lisp dialect **KK**-Lisp, which has been developed at the universities of **K**aiserslautern and **K**arlsruhe as an AI-oriented extension of Common-Lisp. One of the bigger features is a more powerful defstruct macro [BC 85] with incorporated flavour system [Bol 87]. This defstruct is used as basic datastructure in this thesis as well as in the theorem proving environment HADES [Ohl 88] to which the algorithms are connected to.

Defstruct is similar to the "records" in Pascal or "structures" in PL/1. However it is more than a datatype. One could describe it as a program environment, in which a definition of a datatype initiates automatically the construction of several functions and macros applicable to objects of this datatype. For more detailed information see the KK-Lisp manual [KKL 88].

5.1 Structures of Terms and Substitutions

We chose the datastructures suggested and provided by the theorem proving environment HADES developed at the University of Kaiserslautern. It has a structure sharing graph approach, which is extremely suitable for the Boolean ring theory, as its unification algorithms construct exponentially growing terms.

This approach is realized by storing the terms and substitutions as defstruct objects in a large array, where each element occurs exactly once. If an object a occurs as a direct subelement of an other object b, then a counter for object a is increased instead of storing object a twice. Object b receives a pointer to object a and its other direct subelements.

Example:

$\sigma = \{ \; x \leftarrow av + acw$
$\quad\quad y \leftarrow av + c\}$

table: objects with their reference counter

objects	counter	objects	counter
σ	+ 1	av	+ 2
x	+ 1	acw	+ 1
y	+ 1	a	+ 3
av + acw	+ 1	c	+ 2
av + c	+ 1	v	+ 2
		w	+ 1

From the concrete definition of terms and substitution we just describe the excerpt, which has been used during implementation. The term class is splitted into several subclasses. The ones we are concerned with are variables, constants, function symbols and compound terms. In the following we list the classes with a short description of their important slots.

1) Variable and Constant:

reference.counter: number of times the object occurs in other objects,
e.g. direct subterm, unifier, ...
this information is used for garbage collection
terms with counter 0 are deleted

type: type of the object, either :constant or :variable
this information is used to distinguish between variables, constants and compound terms

name: pretty-print name

2) Function symbol:

reference.counter:	number of times the object occurs in other objects
type:	type of the object (:functional)
name:	pretty-print name
axiom.names:	axioms concerned with function symbol ex.: associativity for addition allows identification of function symbols

3) Compound term:

reference.counter:	number of times the object occurs in other objects
type:	type of the object (:c.term)
topsymbol:	toplevel function symbol
arguments:	a list of the direct subterms
free.variables:	variable set of the term
depth:	depth of the term

4) Substitution:

reference.counter:	number of times the object occurs in other objects
domain:	a list of variables
codomain:	a list of the corresponding terms

5.2 Simplification of Terms

Simplification is very important, as it keeps the termsize as small as possible. Therefore every term operation initiates simplification, unless it is explicitly suppressed. In our case simplification is a rewriting of a term into a sorted disjunctive normal form. We will present the rewrite rules, which are deduced from the Boolean ring axioms and properties. Let BR = <B, +, *, 0, 1>.

1) Simp-no.argument

for:	+, 0
rule:	(+) → 0
origin:	addition with no summands is equal to 0
effect:	an internal operation, which is necessary because of recursive simplification calls

2) Simp-remove.one

for: +, 0
rule: $(t + 0) \rightarrow t$
origin: zero element axiom
effect: all zeros are removed

3) Simp-replace.duplicates

for: +, 0
rule: $t + (t_1 + t_1) \rightarrow t,\ t_1 + t_1 \rightarrow 0$
origin: inverse axiom
effect: even times occurring terms are removed,
odd times occurring terms are reduced to one term

4) Simp-remove.zero

for: *, 0
rule: $(t * 0) \rightarrow 0$
origin: Boolean ring property iii)
effect: one zero in a multiplication makes it zero

5) Simp-remove.one

for: *, 1
rule: $(t * 1) \rightarrow t$
origin: unit element axiom
effect: all ones are removed

6) Simp-remove.duplicates

for: *
rule: $(t * t_1 * t_1) \rightarrow t * t_1$
origin: unit element axiom
effect: all more than one occurrences of an element are removed

7) Simp-ac.flatten

for: +

example: $(t_1 + (t_4 + (t_3 + t_2)) \rightarrow (t_1 + t_2 + t_3 + t_4)$

origin: associativity and commutativity

effect: term is flattened and sorted

8) Simp-ac.flatten

for: *

example: $(t_1 * (t_4 * (t_3 * t_2)) \rightarrow (t_1 * t_2 * t_3 * t_4)$

origin: associativity and commutativity

effect: term is flattened and sorted

9) Simp-distributivity

for: +, *

rule: $t_1 * (t_2 + t_3) \rightarrow t_1t_2 + t_1t_3$, $(t_1 + t_2) * t_3 \rightarrow t_1t_3 + t_2t_3$

origin: distributivity axiom

effect: the distributivity is exploited

These simplifiers are applied in the given order on every newly created term. Simplifier 3 and 6 are applied again after the flattening to eliminate new double occurrences. If a simplifier itself creates a new term, then all these simplifier are applied to this new term (For example during an application of simplifier 9 all simplifiers will be applied to the new subterm t_1t_3). Thereby we prevent the creation of large intermediate terms.

5.3 Procedures for Pure Boolean Ring Unification

For the three different unification algorithms (description with improvements in Chapter 3) we took care that all algorithms have the same input pattern of two terms to be unified and <+, *, 0, 1> from the signature. The latter ensures, that the algorithms can be applied to problems in different instantiations of Boolean ring theory. Finally we implemented a main pure unification algorithm, which allows the selection of one specific algorithm by flag, some unifiability tests and equivalence tests for Boolean ring unifiers.

1) br-boolean.ring.unify1
This is the "Variable Elimination" method. The variable selection is determined by *br=select.variable*. For this function an alteration is prepared, which allows your own selection by a menu.

Example: xa + yad + yae + yac + zab + zc + zd | Explanations: term

x 1 | variable and occurrences in subterms

y 2 | If nothing is selected, program chooses lowest

z 3

2) br-boolean.ring.unify2
This is the "Basis of Constants" method.

3) br-boolean.ring.unify3
This is the "Basis of Coefficients" method.

4) br-boolean.ring.unify
This algorithm starts with tests of Section 3.3, which enable a direct generation of a most general or a particular solution. Then an additional input parameter determines, which of the previous three algorithm should be used; of course without reexecuting the tests. This function's versatility makes it optimal for user application, even if it is slightly slower than the others.

1-4) The aim of these methods is to have the smallest possible unifier cardinality. Yet sometimes we want to have all variables in the domain (e.g. renaming purposes). A menu versatility is provided to switch this effect on and off *(br-determine.br.unifier.cardinality)*. It is also possible to switch this effect by an internal function.

5) br-unifiable

It tests, if two Boolean ring terms are unifiable by applying Boole's test.

6) br-boole.s.test

It tests unifiability by Boole's test. Yet this function has three possible result forms determinable by the user:

as.flag	t, if unifiable
as.term	0, if unifiable, else the term computed by Boole's test.
as.list	nil, if unifiable, else the term computed by Boole's test in list representation without function symbols. An advantage to as.term is, that it does not create unnecessary terms.

7) br-s.u.same.unifier.p

It compares two br-unifiers to detect equivalence. Both unifiers should be most general unifiers of the same problem. Otherwise you have to call this function a second time with switched arguments. It operates in that fashion, as it is primarily used to compare the result of the three different unification algorithms.

There are two ways of deduction possible:

- explicit unification
- conversion of unification problem into one term and application of Boole's test.

8) br-s.u.same.unifier.withsamedomain.p

It works exactly as *br-s.u.same.unifier.p* with an additional condition: set-equalness of domains.

5.4 Structures and Procedures for the Combination Algorithm

The combination algorithm problems are presented to us in a different form as the pure unification problems. There we had an equation, two single terms, and we computed a unifier for this equation. Here we have a multi-equation system (MS), represented as list of termlists, and we transform it into another, a solved, multi-equation system. Besides an internal representation for a multi-equation (ME) we require datastructures for the identification and the elimination problem. This leads to the definition of three different classes with increasing descriptive power. All classes are defstruct objects, which are stored together with terms and substitutions of HADES [Ohl 88]. In the following we present the classes with the slots, which are of interest to the user.

1) theory.termlist:	represents one multi-equation splitted into theory parts.
variables:	the variable terms; only significant variables from input MS
free:	compound terms, auxiliary variables used in free part
	and all constants (free and Boolean ring)
boolean:	compound terms and auxiliaries used in Boolean ring part
2) iddata	represents identification problems of a theory.termlists (MS)
theory.termlists:	the theory.termlists
abstractionsdone:	a list of theory.termlist with constants used to abstract their free terms
abstractionsnew:	a list of theory.termlist with constants proposed to abstract their free terms
pointer:	a pointer on resultlist determining the identification stage
pairlist:	working memory of simple identification solutions
	(unifier of free part of two theory.termlists)
addlist:	buffer of identification solutions
resultlist:	memory of identification solutions
3) umdata	
eliminationlist:	a list of elimination problems
	one problem is a list of terms with to be eliminated constants
theorytermlists:	the cyclic theorytermlists
iddatalist:	a list of identification problems <objects of iddata>

In the following we will describe the functions, which are either available to the user or important to the algorithm itself. Let be B = <B, +, *, 0, 1> be a Boolean ring and t_i be terms of the Boolean ring. We start with two function used in pure Boolean ring unification.

br-boolean.ring.unify.termlist:

It computes a unifier for a list of Boolean ring terms representing a multi-equation. We can choose between two methods:

a) Transformation of the multi-equation into the corresponding equation system and solving this equation system $t_1 = \ldots = t_n \rightarrow t_1 = t_2, \ldots, t_1 = t_n$
b) Transformation of the multi-equation into one term t_{ME} representing the multi-equation and solving the equation $t_{ME} = 0$

The second method should be used, if the terms in the multi-equation are similar.

br-boolean.ring.unify.termlists:

It computes a unifier for a list of lists of Boolean ring terms representing a multi-equation system. We can choose between three methods:

a) Transformation of the multi-equation system into the corresponding equation system and solving this equation system:
   $t_{11} = \ldots = t_{1n}, \ldots, t_{m1} = \ldots = t_{mn} \rightarrow t_{11} = t_{12}, \ldots, t_{11} = t_{1n}, t_{21} = t_{22}, \ldots, t_{m1} = t_{mn}$
b) Transformation of each multi-equation ME_i of the multi-equation system into a term t_{MEi} representing the multi-equation and solving the equation system $t_{ME1} = 0, \ldots, t_{MEn} = 0$
c) Transformation of the multi-equation system into one term t_{MS} representing the system and solving the equation $t_{MS} = 0$

The second method should be used, if the terms are similar in the multi-equations, and the third, if the terms are similar in the whole multi-equation system. Tests have shown that the applications for the third method are very limited.

There are four functions, which handle unification in the free theory. They compute unifiers for two terms *br-robinson.unify.twoterms,* for a termlist representing a multi-equation *br-robinson.unify.termlist*, for a list of termlists representing a multi-equation system *br-robinson.unify.termlists*, or they merge two unifiers into one *br-robinson.merge.two.unifier*. The basic concept used is Robinson unification [Rob 65], but these functions work on multi-equations making use of their properties (multiple clash check, etc.).

The cycle elimination process of the combination algorithm needs some functions of graph theory. In particular we employ a function to detect simple cycles *(br=ce.find.cycles)* and one to check the connections in a graph *(br=ce.connectiveness)*. The function *br=ce.find.cycles* is based on an algorithm of H. Weinblatt [Wei 72]. This algorithm computes all simple cycles of a graph. A simple cycle is a cycle, where just one vertex is encountered twice. We have used a variation of Weinblatt's algorithm, where cycles are represented only by their vertices (without arcs). This function is used by the function *br=ce.find.cycle.elimination.points.from.graph*, which computes the possibilities to break all cycles. Usually cycles can be broken by removing any arc of the graph. Yet sometimes there are constraints for breaking cycles. For example let there be two types of vertices and all paths in the graph should alternate between the two types (F1 → E2 → F3 → E4 → F1). Assuming that cycles can only be broken by removing arcs of a certain type (F → E or E → F), then *br=ce.find.cycle.elimination.points.from.graph* will calculate this smaller set of possibilities to break all cycles, if we supply the function with a predicate capable of identifying this type. In our application the two types are Boolean ring and free terms and cycles can be only be broken after Boolean ring terms. The function *br=ce.connectiveness* computes a list of all subgraphs, which are not connected to each other. The function *br-ce.eliminate.aliens*, which computes a unifier solving elimination problems in Boolean rings as offered by *br=ce.find.cycle.elimination.points.from.graph,* uses this function to divide the elimination problems into smaller independant subproblems. The function *br-ce.eliminate.aliens* is independant from our specific datastructure for multi-equation systems and therefore it may be used separately [Tep 88].

The most important function for identification is *br=um.get.one.solution.identification*. The task of this function is to perform the identifications and to compute a theorytermlists representing a multi-equation system, where the identification process has been successfully concluded. It operates like a finite automat. Its stages corrolates to the identification stages, which are determined by the slots pointer, pairlist, addlist and resultlist of iddata. For more detailed information see program documentation.

The combination algorithm of Boolean rings with free function symbols is accessable with the name *br-unify.mixed.termlists*. The input of this function is a unification problem, represented as a

multi-equation system consisting of mixed terms of a specific Boolean ring B' and an arbitrary set of free functions, and the signature of the Boolean ring B' = <+, *, 0, 1>. It computes the set of most general unifiers for the unification problem and transforms the multi-equation system into a set of solved multi-equation systems by application of the unifiers. There is an additional option, which allows to interrupt the computation of unifiers after having computed the first one. The computed value in this case is one solved multi-equation and an object of *um.data,* where the data about the current stage of the algorithm is stored. We can reenter the combination algorithm by calling the function *br-unify.mixed.termlists.from.data* with the *um.data.* This function may be interrupted in the same fashion as the previous one. Thus the unifier of a unification problem can be computed one by one.

Our implementation is independant from the datastructure chosen by HADES for multi-equations and multi-equation systems, as it uses its own internal representations. Nevertheless our combination algorithm processes a multi-equation system to a set of multi-equation systems. The processed multi-equation systems are lists of termlists and a multi-equation a termlist. If the implementors of HADES or other users of our system decide to employ a different datastructure for multi-equations and systems of multi-equations, then the following three functions have to be changed: *br=calc.variables.of.termlists*, which computes all variables inside of the multi-equation system's terms; *br=um.split.in.var.free.and.boolean.part.termlists*, which divides each multi-equation of the system into its theory parts (variable, Boolean ring and Free theory); and *br=um.convert.to.multiequation*, which converts our internal representation of a multi-equation system, the theorytermlists, to the external representation of list of termlists. The first two functions receive the external representation as input. By alteration of these three functions our combination algorithm can be adapted to any external datastructure for multi-equations and and systems of multi-equations.

5.5 Test Environment

The pure unification algorithms and the combination algorithm are primarily designed for application as subprocedures in HADES [Ohl 88]. Therefore the test environment has been mainly developed for internal use like debugging, tracing and comparison of runtime behaviour. Its second task is to enable a direct use of our system. For this purpose we created procedures, which allow to perform unification in a user-defined environment. In regard to those two different concepts this chapter is divided into two parts: one aimed at a programmer, who either intends to connect our algorithms to his

system or wants to enhance the performance of our algorithms; the other one aimed at a user trying to solve particular unification problems.

5.5.1 Test Environment as a Debugging Tool

The test environment is defined by selecting names for variables, free constants, the signature of one Boolean ring <+, *, 0, 1> and, if necessary, a set of free function symbols with their arities. This can be done in two ways, either interactively with a menu *(br-testing.of.boolean.unifier)*, or by an internal function *(br=testing.of.boolean.unifier)* selecting standard default values: f,g,h as free function symbols, x,y,z as variables, a,b,c as constants and <+, *, 0, 1> as signature for Boolean ring. Terms are created with *(br-create)*, which transfers a term from a list representation into our internal representation. Sometimes, especially during debugging, you require the internal address of a symbol, which you have defined while initializing the environment. For this purpose you can use *(br-show.addresses)*, which will print all self-defined constants, variables and functions with their addresses. On the other hand, if you need to know, what the meaning of a symbol in your environment is, you may call *(br-print.test.environment)*. This function pretty-prints the environment, as you have defined it.

There are two test functions: one for unification in free Boolean rings and the other one for unification in a combination of a free Boolean ring with free function symbols. The main task of *br-test* is to compare the three pure unification algorithms. It computes the unifiers of an input term with the boolean constant 0 and measures the computation time needed for the three algorithms. There are two additional parameters. The parameter "apply" controls the application of the unifier computed by the "Basis of Coefficients" method to the input term. This option is used to check, if the computed most general unifier really unifies the input term with boolean constant 0. Its default is nil, as in general application of a unifier needs more time than computing the unifier itself. The second parameter "unifier.print" determines the presentation of the computed unifiers. Usually (unifier.print = T) the unifiers are printed with domain variables and the codomain terms in a list representation. Yet sometimes the unifier is too big. We are able to print just the cardinality of the unifiers, if we switch this option to NIL.*br-test.unify.mix* is the test algorithm for the combination algorithm. It calculates the computation time needed to solve such a unification problem consisting of a multi-equation system represented as list of termlists and pretty-prints the resulting multi-equation systems. The complexity of this algorithm lead to the introduction of several additional parameters. A "time"-flag decides, if the algorithm should be timed. A "pprint"-number, if given, enables a trace

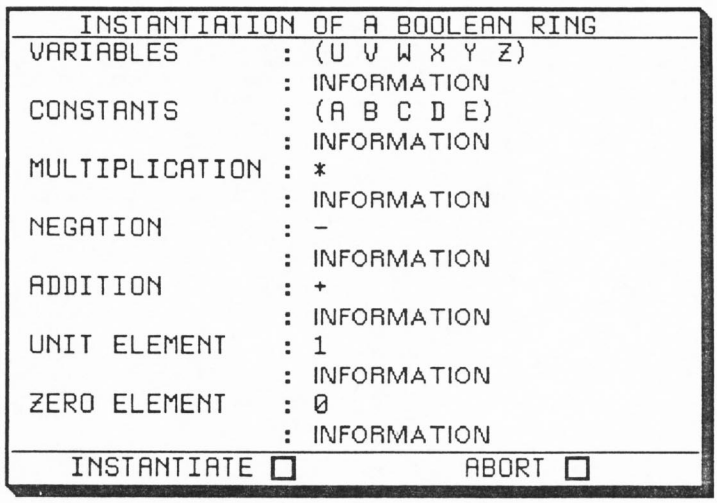
pretty-printing the transformed multi-equation system after each major step of the algorithm, which is executed before the identification process. The higher the number (1 - 4), the more detailed is the information. The last parameter gets his importance from internal processes. With "same.address" we have the choice between :old, :new and :environment, where :old means that no garbage collection is performed in the term area, :environment that garbage collection is performed in term area, :new reinitializes the environment in advance. We choose :old, if we just want to check, if the algorithm computes the right values. Assuming the garbage collection of HADES works correctly, we may also select :environment. It is slightly slower, but then we can also check, if the reference counter of objects created by our algorithms is correct. If we select :new, then the storage, where terms, substitutions and other objects of the HADES and of our algorithms are stored, is cleared and then reinitialized as defined with the initialization function for our environment (see *br-testing.of.boolean.unify*). All other data is lost. The advantage of this selection is, that it ensures, that before calling our algorithms everything is in the correct state. A disadvantage is, that a definition of a Boolean ring needs comperatively too much time.

5.5.2 Test Environment as a Tool for Solving Boolean Unification Problems

To enable the user to solve unification problems in a self-defined environment, we offer him the function *br-select.example*. Some of its interna are based on features mentioned in Section 5.5.1. It is totally menu controlled, which makes it perfect for presentation purposes. The problems to be solved, the multi-equation systems, can be input in two fashions: either interactively on demand of this function or in advance as an argumentlist to *br-select.example*. This versality allows to prepare certain unification problems in advance. Of course there are no restrictions on the definition of the Boolean ring and the free theory. The test configuration (timing, pprint, ...) can be changed after each run. In the following we will explain, how this function proceeds.

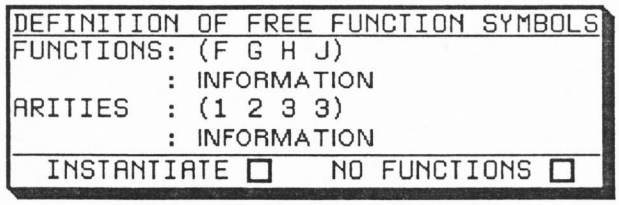
At the start a menu appears on the screen, by which the test environment for the following tests can be defined. With this menu one selects the names for variables, free constants and the signature of a Boolean ring necessary to create unification problems in a free Boolean ring.

Menu 1



Then we are confronted with a menu, which defines a set of free function symbols. If we choose to mark "Instantiate", then the function symbols are defined. In this case we can create unification problems in the combination of a Boolean ring with free function symbols. (Remark: Even if we choose to define none, the combination algorithm is executed. This is done because of internal reasons.).

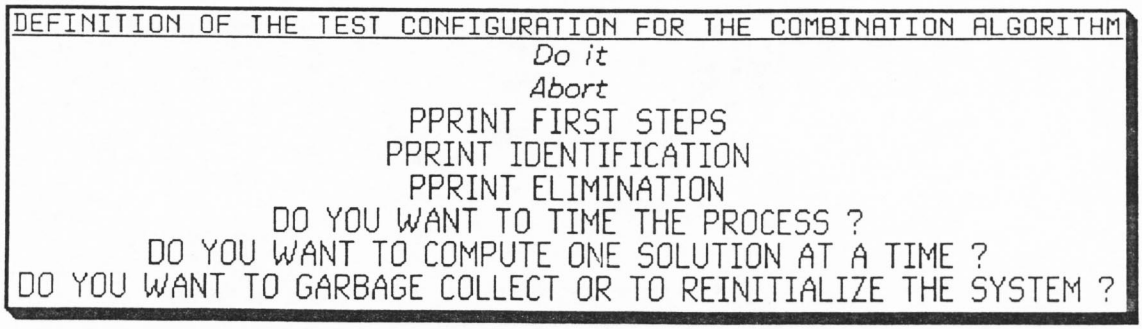
Menu 2



Of course, if we choose to input some problems in advance, we have to define our terms using the provided default names for the symbols.

The next menu defines, how the tests shall be executed. We can insert pretty-prints in different parts of the algorithm, measure the algorithm, compute all solutions at once or one after another and decide on garbage handling.

Menu 3

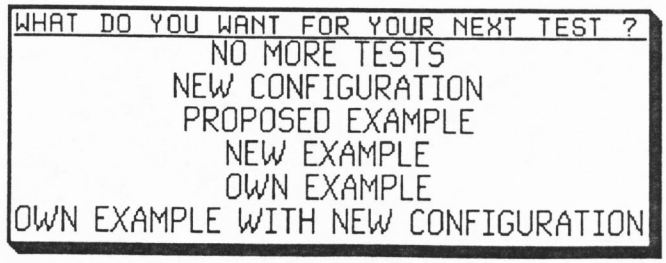


The first three selectors determine, if pretty-prints should be executed. "Pprint First Steps" activates the pretty-prints of the multi-equation system after each major step executed before identification. "Pprint Identification" prints the system before identification with the possible identification pairs. During identification it informs, if identification is succesful or if it leads to recursive identifications. "Pprint Elimination" describes the cycle elimination process. It prints cycles of the multi-equation system with constant elimination problems to be solved. It also describes recursive cycle checks. With the other selectors we can decide to measure the combination algorithm or to compute one solution at a time. If we choose to measure and compute one solution at a time, then each computation of a solution is timed seperately. The last selector controls the garbage collection in the term area. If we click the left mouse button (Garbage Collect), then the computed objects are deleted. If we click the middle button (Reintialize), then the area, where terms, substitutions, etc. are stored is reinitialized. We keep computed results, if the right button (Results Are Not Deleted) is clicked.

Now having determined a test configuration, we may start the test runs. With the help of the following menu we select our example.

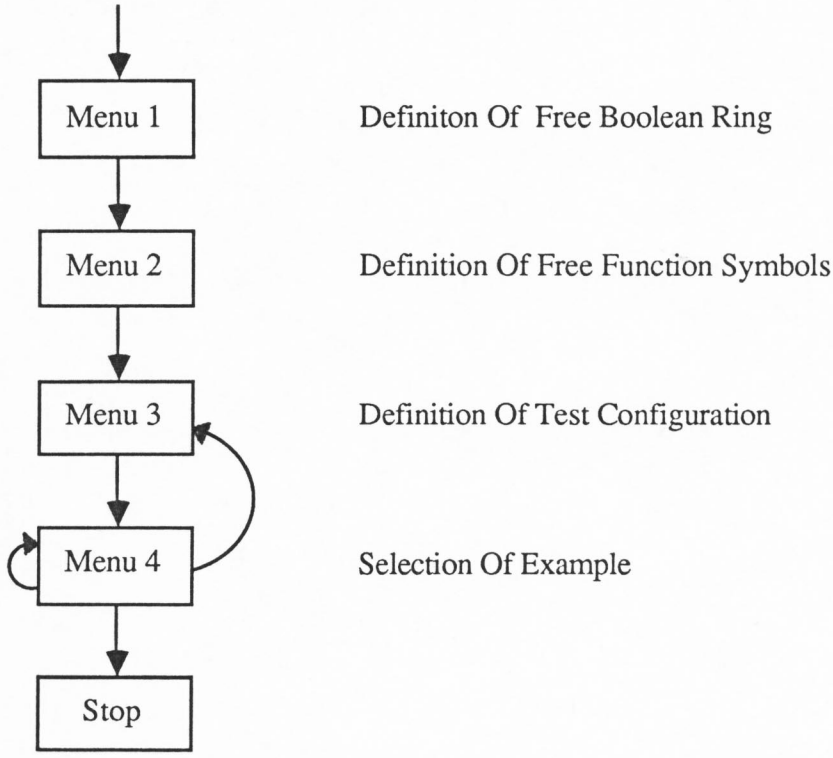
Menu 4 with example proposal

```
THE TESTSYSTEM PROPOSES THIS MULTI-EQUATION SYSTEM AS NEXT EXAMPLE
(+ (F X) (F Y)) == (+ (F A) (F B))
```



The first predefined multi-equation system, which is displayed on terminal, can be selected by "Proposed Example". If there are more predefined examples, we can switch to the next one selecting "New Example". If we want to define an example on our own, we select "Own Example". In this case we are provided with a special editor, in which we may create a new multi-equation system. If we want to execute our test in a new configuration, we select "New Configuration".

The last two menus occur alternatively, allowing either to change the configuration or the example. The process is stopped by selecting "No More Tests" in Menu 4. The course of function *br-select.example* is summarized in the next diagram.



Some test runs of important examples, which were introduced Martin/Nipkov, are included in the appendix. Those test runs have been executed with all possible pretty-prints activated. This enables a good understanding of the examples as well as of the algorithm itself.

6. APPLICATIONS

6.1 Application Fields for Boolean Ring Unification Algorithms

There are three major application fields for pure unification algorithms: **digital hardware design**, **set theory** and **propositional logic**.

As previously stated, the "Variable Elimination" method has been rediscovered by W. Büttner and H. Simonis [BüSi 87]. They intend to improve logic programming (using Prolog as an example) by embedding data types. For integration of the data type "Boolean expression", they need the Boolean ring unification. The embedding of this data type is comperatively easy because of the unitarity of Boolean ring unification. Now with this enhanced Prolog they can comfortably describe digital circuits, as they can represent boolean expressions (and, xor) at term level. This opens their system to a wide range of application in **digital hardware design** consisting of verification, simulation, synthesis, simplification, specialization and debugging. In his report of the unification workshop at Val d'Ajol 1987 Simonis conferred in great detail a verification of a complete sixteen bit adder described at the logic gate level [Sim 87]. Other examples are simulation of executable specifications, synthezing terms or equations from truth tables, simplification induced by design rule changes (ECL -> MOS) or specialization of circuits like from an adder to increment by augmenting the description. Most of these different applications may also be realized directly with systems capable of solving pure Boolean ring equation systems. The foundation of this application possibility is the one-to-one correspondence between the two-element Boolean ring and the truth-functional calculus in digital hardware ($+ \leftrightarrow \oplus$, $* \leftrightarrow \wedge$, $0 \leftrightarrow F$, $1 \leftrightarrow T$).

Example: Specialization of a four bit adder to a four bit increment [Sim 87]

Let be B be a Boolean ring $<\emptyset, \oplus, \wedge, F, T>$ and x, y, c_{in}, c_{out}, x_1, ..., x_4, y_1, ..., y_4, c_0, ..., c_4, s_1, ..., s_4 are variables.

An adder computes the sum s and the carry c_{out} of two binary digits x and y and a carry c_{in}. It is described as $s = x \oplus y \oplus c_{in}$ and $c_{out} = x \wedge y \oplus x \wedge c_{in} \oplus y \wedge c_{in}$.

A four bit adder computes the sum $S = s_4s_3s_2s_1$ and out-carry c_4 of two four bit numbers $X = x_4x_3x_2x_1$, $Y = y_4y_3y_2y_1$ and an in-carry c_0. A four bit adder is constructed out four adders. Thereby it is described by the following equation system:

$\forall_i\, 1 \leq i \leq 4 \quad s_i = x_i \oplus y_i \oplus c_{i-1}$ and $c_i = x_i \wedge y_i \oplus x_i \wedge c_{i-1} \oplus y_i \wedge c_{i-1}$

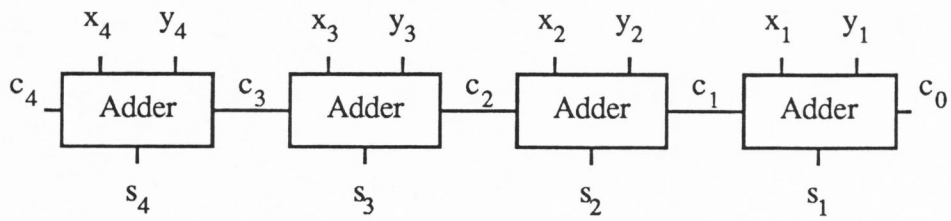


Figure 6.1.1: A four bit adder

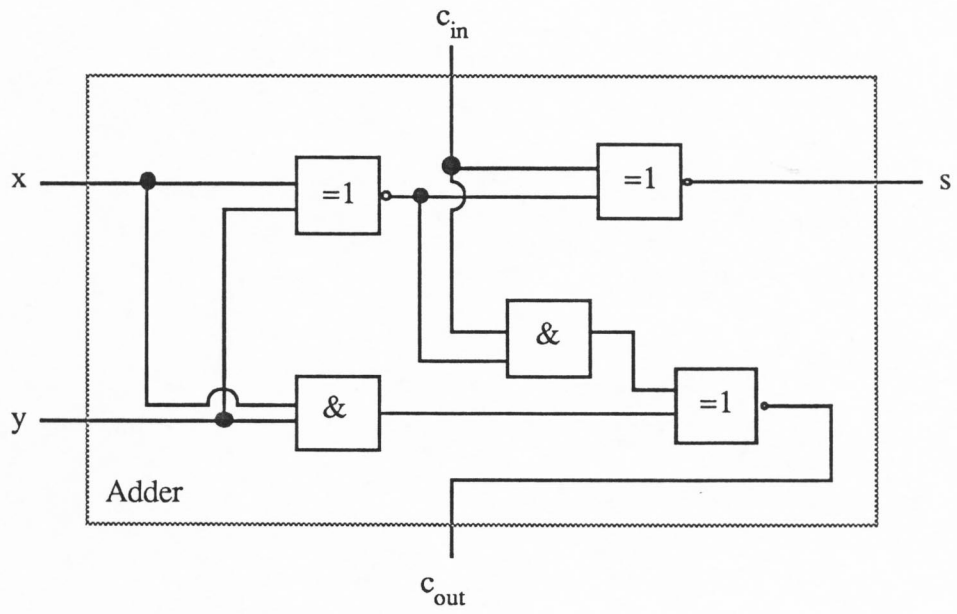


Figure 6.1.2: An adder.

A four bit increment adds 1 to the binary number $X = x_4x_3x_2x_1$, computing results $S = s_4s_3s_2s_1$ and out-carry c_4. This means we can add the following equations to the equation system: $y_1 = T$, $y_2 = F$, $y_3 = F$, $y_4 = F$, $c_0 = F$. After unification and simplification we get the following equation system describing the four bit increment:

$$s_1 = x_1 \oplus T \text{ and } c_1 = x_1$$

$$\forall_i 2 \leq i \leq 4 \quad s_i = x_i \oplus c_{i-1} \text{ and } c_i = x_i \wedge c_{i-1}$$

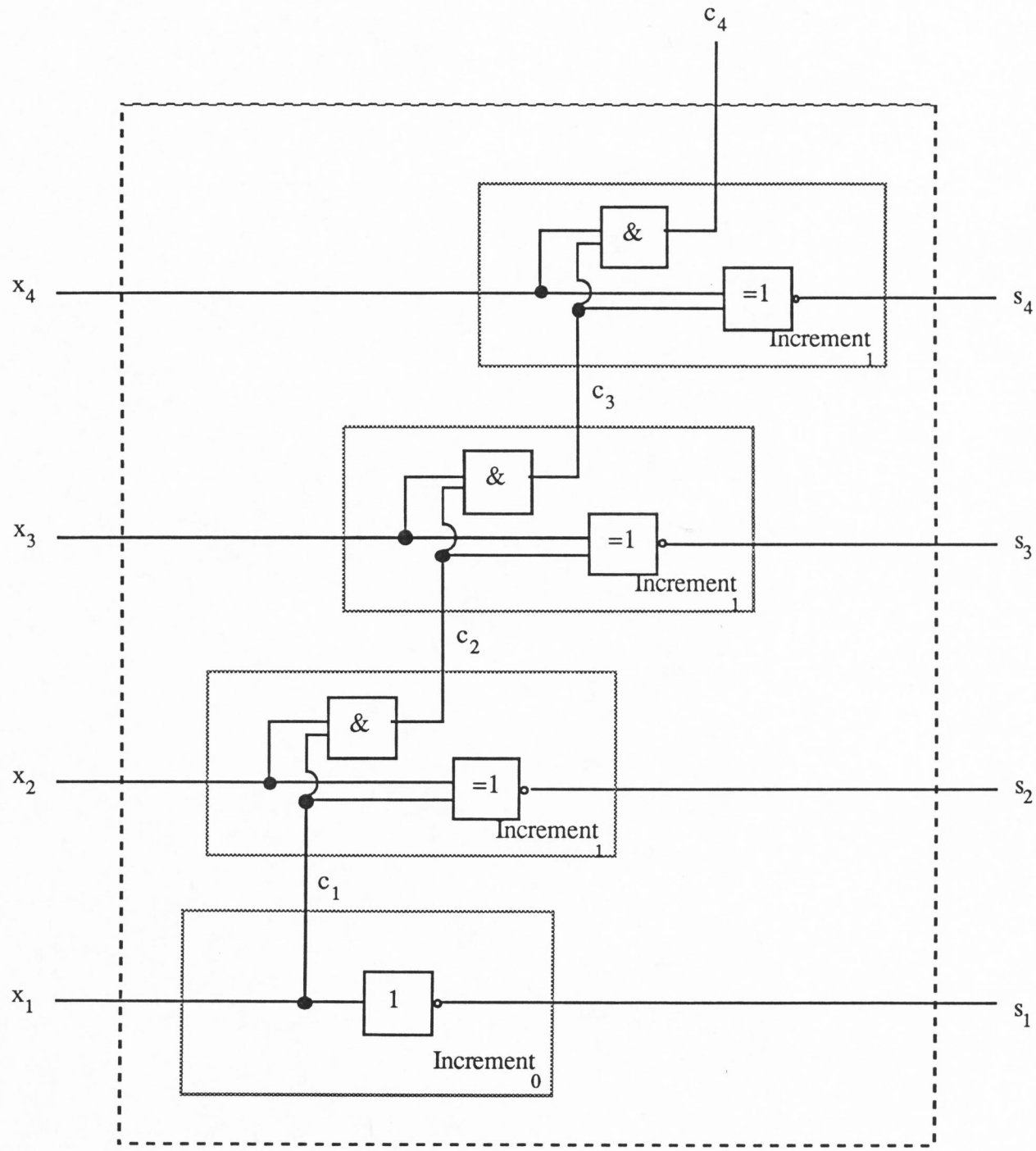


Figure 6.1.3: A four bit increment.

For more detailed information see Simonis' paper on digital hardware design applications [Sim 87].

Another application field is **propositional logic**. The set of well formed formulae of the propositional calculus on propositional symbols $P = \{p_1, \ldots, p_n\}$ forms a Boolean ring, which is isomorphic to $T(P, \Sigma)_E$. Thereby propositions with their operators ($\wedge$, $\vee$, $\neg$, $\rightarrow$, ...) can be transformed into equivalent Boolean ring terms. Unification is not necessary to determine, if a proposition is a tautology or unsatisfiable. It is sufficient to simplify the corresponding Boolean term and to check, if the term is equal to 1 or 0. Unifying a term involving variables with 1 or 0 relates to finding the most general values of these variables, which make the corresponding proposition a tautology or unsatisfiable.

We recall an application presented by U. Martin and T. Nipkov: Construction of Derived Rules.

Example: [MN 86]

Determine the most general value of x, which will make the following rule into a derived rule.

$$\frac{(p \rightarrow q) \wedge x}{q \vee r}$$

This means we compute the most general solution of $(p \rightarrow q) \wedge x \rightarrow (q \vee r) = 1$. Applying transfer rules $(a \vee b) \Rightarrow (a + b + ab)$, $(a \wedge b) \Rightarrow ab$, $(a \rightarrow b) \Rightarrow (1 + a + ab)$, we can translate the equation into a Boolean ring equation :

$$1 + x\,(1 + p + pq) + x\,(1 + p + pq)\,(q + r + qr) = 1$$
$$\Leftrightarrow \quad x\,(1 + p + pq)\,(1 + q + r + qr) = 0$$
$$\Leftrightarrow \quad x\,((1 + p) + pq)\,(1 + q)\,(1 + r) = 0$$
$$\Leftrightarrow \quad x\,(1 + p)\,(1 + q)\,(1 + r) = 0.$$

The most general solution is $\tau = \{x \leftarrow x'\,(1 + (1 + p)\,(1 + q)\,(1 + r))\}$, which is equivalent to $\tau_p = \{x \leftarrow x' \wedge \neg(\neg p \wedge \neg q \wedge \neg r)\} = \{x \leftarrow x' \wedge (p \vee q \vee r)\}$. Thus we have shown that for any x,

$$\frac{(p \rightarrow q) \wedge (p \vee q \vee r) \wedge x'}{q \vee r}$$

is a derived rule.

Boolean ring unification can also be employed in **set theory**. In particular the powerset of any set $S = \{s_1, \ldots, s_n\}$ forms a Boolean ring under the operations symmetric difference (Δ) and intersection ($\cap$). The Boolean ring unit element 0 relates to the empty set, the unit element 1 to set S and the boolean functions +, * to set operators Δ, $\cap$. In general the operators intersection ($\cap$), union ($\cup$) and complement ($\setminus$) are used in set theory. Terms using such operators are also transferable, just as we have explained for Boolean algebra in introductory Chapter 2 (Let t_1, t_2 be terms. $t_1 \cup t_2 = t_1 \Delta t_2 \Delta t_1 t_2$, $\setminus t_1 = S \Delta t_1$). Therefore it is sufficient to solve unification problems in a Boolean ring $B = < S^{\{\}}, \Delta, \cap, \emptyset, S >$, where $S = \{s_1, \ldots, s_n\}$ and $S^{\{\}}$ is a set of one element sets $\{\{s_1\}, \ldots, \{s_n\}\}$. Nevertheless we **can not directly use** the unification algorithms of Chapter 3. They have be modified, as the operators (Δ, $\cap$) are also defined on the elements of generating set $S^{\{\}}$.

Chapter 3:	$s_1 \cap s_2 = s_1 \cap s_2$	$s_1 \Delta s_2 = s_1 \Delta s_2$	$= s_1 \neg s_2 \cup \neg s_1 s_2$
set theory:	$\{s_1\} \cap \{s_2\} = \emptyset$	$\{s_1\} \Delta \{s_2\} = \{s_1\} \Delta \{s_2\}$	$= \{s_1\} \cup \{s_2\}$

In the Büttner/Simonis method we have to adjust the operators, while for the Martin/Nipkov method we also compute a different orthogonal basis and thereby a different specific solution. Those changes are demonstrated by the following example, where we employ both methods to compute the most general unifier for an equation in Boolean ring $B' = < S^{\{\}}, \Delta, \cap, \emptyset, S >$, where $S = \{a, b, c, \ldots\}$, with Variables = {x, y} (To abbreviate we will write for the one element set like {c} the expression c and for $a \cap b$ the expression ab).

Example: Compute the mgu for the equation $\langle cxy \Delta ax \Delta by \Delta c = 0 \rangle_{B'}$.

a) "Variable Elimination" method

$t = cxy \Delta ax \Delta ay \Delta by \Delta c =_{B'} (cy \Delta a)\, x \Delta ay \Delta by \Delta c$ | eliminate variable x

$t = (cy \Delta a \Delta S)\,(ay \Delta by \Delta c) =_{B'} by \Delta cy \Delta c =_{B'} (b \Delta c)\, y \Delta c$ | eliminate variable y

$t = (b \Delta c \Delta S)\, c =_{B'} c \Delta c =_{B'} 0$

Resubstitution of solutions

$\tau = \{\}$

$\tau = \{\} \circ \{y \leftarrow c \Delta y' (b \Delta c \Delta S)\} = \{y \leftarrow c \Delta y' (b \Delta c \Delta S)\}$

$\tau = \{y \leftarrow c \Delta y' (b \Delta c \Delta S)\} \circ \{x \leftarrow ay \Delta by \Delta c \Delta x' (cy \Delta a \Delta S)\}$

$=_{B'} \{y \leftarrow c \Delta by' \Delta cy' \Delta y'$

$x \leftarrow (a \Delta b)\,(c \Delta by' \Delta cy' \Delta y') \Delta c \Delta x' (c\,(c \Delta by' \Delta cy' \Delta y') \Delta a \Delta S)\}$

$=_{B'} \{y \leftarrow c \Delta y' \Delta by' \Delta cy'$

$x \leftarrow c \Delta x' \Delta ay' \Delta ax' \Delta cx'\}$

b) "Basis of Constants" method (as an example for the "Particular Solution" approach)

1) Conversion of term t = cxy Δ ax Δ ay Δ by Δ c into polynomial form.
   t = cxy Δ ax Δ (a Δ b)y Δ c
2) Splitting t into its homogeneous part t_{hom} and its inhomogeneous part t_{inhom}.
   t_{hom} = cxy Δ (a Δ b)y Δ by
   t_{inhom} = c
3) Sort the homogeneous part with the criterion "smallest variable part".
   t_{hom} = (a Δ b)y Δ by Δ cxy
4) Compute the part D' of orthogonal basis necessary to represent to t_{inhom}.
   The orthogonal basis for ring B' is the set S, as every element of ring B'(the power set of S) can be written as a linear combination of elements of S and the elements of S are trivially orthogonal. Therefore D' = {c}.
5) The solution we collect for c is N(c) = {xy}, as ca $=_{B'}$ 0, c(a Δ b) $=_{B'}$ 0 and cc $\neq_{B'}$ 0. Therefore the particular solution is σ = {y ← c, x ← c}
6) The most general solution is

$$\begin{aligned}\tau = \{&y \leftarrow y' \Delta (cx'y' \Delta ax' \Delta ay' \Delta by' \Delta c)\ (y' \Delta c)\\ &x \leftarrow x' \Delta (cx'y' \Delta ax' \Delta ay' \Delta by' \Delta c)\ (x' \Delta c)\}\\ =_{B'} \{&y \leftarrow c \Delta y' \Delta ay' \Delta by' \Delta cy' \Delta ax'y'\\ &x \leftarrow c \Delta x' \Delta ax' \Delta cx' \Delta ax'y' \Delta bx'y'\}.\end{aligned}$$

Summarizing the presentation of those three different applications, we come to the following conclusions. The greatest possible impact lies in the field of digital hardware design. In this wide field we find several practical applications for our algorithms, which directly support the developer of digital circuits. The other two fields, propositional logic and set theory, are of a more theoretical relevance.

6.2 Extending the Algorithms to Another Theory

Quite often boolean problems are not formulated in terms of a Boolean ring, but in a signature Γ of different theory B. As long as the signature of theory B is expressive as the Boolean ring theory, i.e. theory B is weak isomorphic to Boolean ring theory, there is no problem, because we can translate in either direction. In this chapter it will be discussed, if and how our unification algorithms can be made applicable to such a theory B. As an example we have selected the **Boolean algebra** with

signature $\Gamma = \{\cup, \cap, \neg, 0, 1\}$.

The first naive approach would be to translate the unification problem from the Boolean algebra into the Boolean ring, to solve it there, and then to transform the solution back into Boolean algebra. The transformations from Boolean ring (BR) to Boolean algebra (BA) and back are defined by T and T^{-1}. They are equivalent to the $^{\otimes}$ operator of Stone's theorem (Theorem 2.1.1)[Sto 36].

T: $BA \rightarrow BR$ $\quad t_{BA} \rightarrow t_{BR} = T(t_{BA})$

T^{-1}: $BR \rightarrow BA$ $\quad t_{BR} \rightarrow t_{BA} = T^{-1}(t_{BR})$

The transformations have these properties, which are used in the sequence:

$T^{-1}(T(t_{BA})) = t_{BA}$ and $T(T^{-1}(t_{BR})) = t_{BR}$

From Stone's theorem we can obviously deduce the following facts also used sequently:

Let σ, γ be substitutions and s, t be terms of either theory.

Fact 1) $T(\sigma s) =_{BR} T(\sigma)T(s)$ and $T^{-1}(\sigma s) =_{BA} T^{-1}(\sigma)T^{-1}(s)$

Fact 2) $s =_{BA} t \Leftrightarrow T(s) =_{BR} T(t)$ and $s =_{BR} t \Leftrightarrow T^{-1}(s) =_{BA} T^{-1}(t)$

Fact 3) $T(\gamma\sigma) =_{BR} T(\gamma)T(\sigma)$ and $T^{-1}(\gamma\sigma) =_{BA} T^{-1}(\gamma)T^{-1}(\sigma)$

Fact 4) Let F be the set of free function symbols

$\forall f \in F \quad T(f(s_1, \ldots, s_n)) =_{BR} f(T(s_1), \ldots, T(s_n)))$ and

$\forall f \in F \quad T^{-1}(f(s_1, \ldots, s_n)) =_{BA} f(T^{-1}(s_1), \ldots, T^{-1}(s_n)))$

Lemma 6.2.1 states, that the most general unifier of an equation, which has been converted from Boolean algebra to Boolean ring, is in his reconverted form a most general unifier of the original equation.

Lemma 6.2.1: $\mu U_{BA}(s_{BA} = t_{BA}) = \{\sigma_{BA} = T^{-1}(\sigma_{BR}) \mid \sigma_{BR} \in \mu U_{BR}(T(s_{BA}) = T(t_{BA}))\}$

Proof:

First we prove that σ_{BA} is a unifier:

$\sigma_{BA}s_{BA} =_{BA} \sigma_{BA}T^{-1}(T(s_{BA}))$ | by fact 1

$=_{BA} T^{-1}(\sigma_{BR}(T(s_{BA}))$ | by fact 2

$=_{BA} T^{-1}(\sigma_{BR}(T(t_{BA}))$ | by fact 1

$=_{BA} \sigma_{BA}T^{-1}(T(t_{BA}))$

$=_{BA} \sigma_{BA}t_{BA}$

Now let $\tau_{BA} \in U_{BA}(s_{BA} = t_{BA})$. Then there is a λ_{BA} with $\lambda_{BA}\tau_{BA} =_{BA} \sigma_{BA}$ [Variables (s_{BA}, t_{BA})].
$T(\tau_{BA}) \in U_{BR}(T(s_{BA}) = T(t_{BA}))$, so there is a $T(\lambda_{BA})$ with
$T(\lambda_{BA})\, T(\tau_{BA}) \;=_{BR}\; T(\sigma_{BA})$ [Variables (s_{BA}, t_{BA})] | by fact 3
$\Rightarrow\; T(\lambda_{BA}\tau_{BA}) \;=_{BR}\; T(\sigma_{BA})$
$\Rightarrow\quad \lambda_{BA}\tau_{BA} \;=_{BA}\; \sigma_{BA}$ ■

Yet this method has a decisive disadvantage: The translation of the unification problems as well as of the resulting codomain terms will exponentially blow up the size of the formulae. This can be suppressed, if we leave the formulae untouched and **transform the algorithms** instead. In their converted forms we present the lemma necessary for the "Variable Elimination" method and the theorem for the "Particular Solution" method.

Lemma 6.2.2:
Let x, x' be variables and c, d, e terms of Boolean Algebra BA and $x \notin$ Variables (c, d, e).
$\sigma := \tau \circ \{x \leftarrow (\neg c \cap \neg d \cap \neg e) \cap x' \cup (d \cup e) \cap \neg x' \cup (\neg c \cap d \cap \neg e)\}$ is mgu of
$\langle (c \cap x) \cup (d \cap \neg x) \cup e = 0 \rangle_{BA}$
$\Leftrightarrow \tau$ is mgu of $\langle (c \cap d) \cup e = 0 \rangle_{BA}$

Proof:
The original lemma for the "Variable Elimination" method is Lemma 3.1.1:
$\sigma = \tau \circ \{x \leftarrow b + x'(1 + a)\}$ is mgu of $\langle ax + b = 0 \rangle_{BR} \Leftrightarrow \tau$ is mgu of $\langle ab + b = 0 \rangle_{BR}$.
It is either possible to prove Lemma 6.2.2 analogously to Lemma 3.1.1 or to use the transformations of Stone's theorem (Theorem 2.1.1). We apply the second method:

$$
\begin{aligned}
(c \cap x) \cup (d \cap \neg x) \cup e &= (cx + d\neg x) \cup e = cx + d\neg x + e + cxe + d\neg xe \\
&= cx + dx + d + e + cxe + dxe + de \\
&= (c + d + ce + de)\, x + d + e + de \\
&= ((\neg c \cap d) \cup (c \cap \neg d) \cap \neg e)\, x + (d \cup e)
\end{aligned}
$$

$\Rightarrow a = (\neg c \cap d) \cup (c \cap \neg d) \cap \neg e)$ and $b = (d \cup e)$

$$
\begin{aligned}
ab + b = \neg a \cap b &= \neg((\neg c \cap d) \cup (c \cap \neg d) \cap \neg e) \cap (d \cup e) \\
&= ((c \cap d) \cup (\neg c \cap \neg d) \cup e) \cap (d \cup e) \\
&= (c \cap d) \cup (d \cap e) \cup (c \cap d \cap e) \cup (\neg c \cap \neg d \cap e) \cup e \\
&= (c \cap d) \cup e
\end{aligned}
$$

$$
\begin{aligned}
b + x'(1 + a) &= b + (x' \cap \neg a) = (\neg b \cap x' \cap \neg a) \cup (b \cap \neg(x' \cap \neg a)) \\
&= (\neg a \cap \neg b \cap x') \cup (b \cap \neg x') \cup (a \cap b) \\
&= (((c \cap d) \cup (\neg c \cap \neg d) \cup e) \cap (\neg d \cap \neg e) \cap x') \cup ((d \cup e) \cap \neg x') \cup \\
&\quad (((\neg c \cap d) \cup (c \cap \neg d) \cap \neg e) \cap (d \cup e)) \\
&= (\neg c \cap \neg d \cap \neg e) \cap x' \cup (d \cup e) \cap \neg x' \cup (\neg c \cap d \cap \neg e) \quad \blacksquare
\end{aligned}
$$

Theorem 6.2.3:

Let t_1 and t_2 be two terms of free Boolean algebra BR over the Boolean algebra B and let $b_1, \ldots, b_n$ be elements of B and $Var(t_1) \cup Var(t_2) = \{x_1, \ldots, x_n\}$ the used variables.

When $\sigma t_1 =_{BA} \sigma t_2$ with $\sigma = \{x_i \leftarrow b_i \mid 1 \le i \le n\}$

then the substitution

$$\tau = \{x_i \leftarrow (b_i \cup \gamma(\neg t_1)) \cap x'_i \cup (b_i \cap \gamma(t) \cap \neg x'_i) \mid 1 \le i \le n\}$$

with $\gamma = \{x_i \leftarrow x'_i \mid 1 \le i \le n\}$

and $t = (\neg t_1 \cap t_2) \cup (t_1 \cap \neg t_2)$

is the most general unifier of t_1 and t_2.

Proof:

The original theorem for the "Particular Solution" method is Theorem 3.2.1. Again it is possible to prove Theorem 6.2.3 analogously to Theorem 3.2.1. Yet we will transform the theorem:

$$
\begin{aligned}
x'_i + \gamma(t) * (x'_i + b_i) &= (x'_i \cap \neg(\gamma(t) \cap (x'_i + b_i)) \cup (\neg x'_i \cap (\gamma(t) \cap (x'_i + b_i)) \\
&= (x'_i \cap (\neg\gamma(t) \cup \neg(x'_i + b_i)) \cup (\neg x'_i \cap b_i \cap \gamma(t)) \\
&= (x'_i \cap (\neg\gamma(t) \cup (x'_i \cap b_i) \cup (\neg x'_i \cap \neg b_i))) \cup (\neg x'_i \cap b_i \cap \gamma(t)) \\
&= (x'_i \cap (\gamma(\neg t) \cup (x'_i \cap b_i))) \cup (b_i \cap \gamma(t_1) \cap x'_i) \\
&= (b_i \cup \gamma(\neg t_1)) \cap x'_i \cup (b_i \cap \gamma(t) \cap \neg x'_i)
\end{aligned}
$$

$t = t_1 + t_2 = (\neg t_1 \cap t_2) \cup (t_1 \cap \neg t_2)$ ■

The "Variable Elimination" method can be constructed directly from the lemma. Yet for the "Particular Solution" methods, which are based on the theorem, other approaches have to be selected to determine a special solution. The "Basis of Constants" and "Basis of Coefficients" methods use properties of the polynomial form, which is not available in Boolean algebra. Therefore we present the full conversion of the "Variable Elimination" method.

Algorithm: **"Variable Elimination" Method** for Boolean Algebra

Input: term $t \in$ Boolean algebra BA.

Output: most general BA-unifier σ of term t and 0.

Step 1)	If $t =_{BA} 0$, then $\sigma := \{\}$ is mgu of $\langle t = 0 \rangle_{BA}$.
Step 2)	If Variables (t) = $\emptyset$, then stop with **fail**.
Step 3)	Select a variable $x \in$ Variables (t) with $t = (c \cap x) \cup (d \cap \neg x) \cup e$ and $x \notin$ Variables (c, d, e)
Step 4)	Compute the most general unifier τ of the problem $\langle (c \cap d) \cup e = 0 \rangle_{BA}$.
Step 5)	$\sigma := \tau \circ \{x \leftarrow (\neg c \cap \neg d \cap \neg e) \cap x' \cup (d \cup e) \cap \neg x' \cup (\neg c \cap d \cap \neg e)\}$ is mgu of $\langle t = 0 \rangle_{BA}$.

Besides the unification algorithms for pure Boolean rings, we would also like to adapt the combination algorithm. To modify the algorithm, that it can handle problems in the combination of Boolean algebra and free theory, we need the following lemma:

Lemma 6.2.4:

Let BA be a Boolean algebra, BR a Boolean ring and F the free theory.

Let + indicate the combination of theories.

$$\mu U_{BA+F}(s_{BA+F} = t_{BA+F}) = \{\sigma_{BA+F} = T^{-1}(\sigma_{BR+F}) \mid \sigma_{BR+F} \in \mu U_{BR+F}(T(s_{BR+F}) = T(t_{BR+F}))\}$$

Proof:

Lemma 6.2.1 has established the assertion above for pure problems. In this lemma we have additionally the free function symbols. Fact 4 deduced from Stone 's theorem allows to extend the transformation to a combination with the free function symbols.

By this lemma it is sufficient to replace the Boolean ring part of the combination algorithm with Boolean algebra part. Then the algorithm is able to handle problems in the combination of Boolean algebra and free theory.

In this chapter we have shown, that it is possible to extend the unification algorithms from Boolean ring to another theory, the Boolean algebra. An imperative condition is, that the expressiveness of the new theory is equivalent to the expressiveness of Boolean ring theory. A second condition, which would improve the performance of the new algorithm, is the availablity of a deterministic normal form. Then we can detect term equality very quickly. This has no effect on the adaption of the algorithm itself, as seen in the Boolean algebra case. Yet it influences internal processes like simplifcation and term equality and thereby it will slow down the performance of the algorithm.

Remark:

The unification algorithms can also be adapted to Boolean algebras with other operators (ex.: the one with just the operator $\neg\cap$, i.e. nand), as long as the premises mentioned are fulfilled. This makes the procedures ideal for applications in digital hardware design.

7. CONCLUSION

In this thesis we accomplished to confer a thorough presentation of the two major approaches to unification in free Boolean rings. By rigorous scrutinization and comparison of the methods we revealed their weaknesses, which sequently were removed or at least diminished. Also we successfully suppressed some of the exponential term growth related to the unification algorithms. Yet under some circumstances, especially while handling bigger problems (terms with more than five variables and five constants), we still get a too high computational complexity. Our research here disclosed some tendencies, which should be promising for the future. Regrettably we were only to start off in those directions, as a complete coverage would have gone beyond the scope of this thesis. We preferred to equip the reader with sufficient data, that he is able to select the best algorithm for his applications.

The second part of this thesis, the combination algorithm, is of significant relevance, as previous combination algorithms were not able to solve the open problem of combining a simple theory (free theory) and arbitrary one (Boolean ring theory). The foundation of our method is an algorithm by M.Schmidt-Schauß. After an examination of his ideas and propositions we developed their realization. Furthermore we enhanced the performance of the algorithm by introducing some of our ideas, which are partially due to properties of the Boolean ring. The basic components of the algorithm are E-unification algorithms for the two involved theories and a method solving constant-elimination problems in the Boolean ring. For unification of pure Boolean ring terms we selected algorithms from our improved set of algorithms. Mainly we employed the "Variable Elimination" method, because its computed unifiers are often smaller.

With our current system we are able to solve the test examples known from the literature with a satisfactory run time. For more complex examples we offer a feature, which allows to compute the most general unifiers one at a time instead of the standard procedure of computing the set of most general unifiers all at once.

Finally we explained, how to extend our algorithms from Boolean rings to Boolean algebras. This enlarges their application fields decisively.

8. REFERENCES

[BB 87] K.H.Bläsius, H.-J.Bürckert: *Deduktionssysteme*. Oldenbourg, (1987)

[BC 85] A.Bold, B.Crone-Rawe: Extension of Lisp by Record Structures. *Projektarbeit,* University of Kaiserslautern, (1985)

[BHS 87] H.-J.Bürckert, A.Herold, M.Schmidt-Schauß: On Equational Theories, Unification and Decidability. *Proceedings Rewriting Techniques and Applications,* (also *Journal of Symbolic Computation),* LNCS 256, pp.204-215, (1987)

[Bol 87] A.Bold: An Object-Orientated Extension of Common Lisp. *Diplomarbeit,* University of Kaiserslautern, (1987)

[Boo 47] G.Boole: *The Mathematical Analysis of Logic*. Macmillan, (1847), reprinted 1948

[Bou 87] A.Boudet: Unification dans les Anneaux Booléens en Présence de Symboles Libres. *Rapport de Stage D.E.A*, (September 1987)

[Bry 86] R.E.Bryant: Graph-Based Algorithms For Boolean Function Manipulation. *IEEE Transactions on Computers*, Vol. C-35, No. 8, pp. 677-691, (August 1986)

[BJS 88] A.Boudet, J.-P.Jouannaud, M.Schmidt-Schauß: Unification in Boolean Rings and Abelian Groups. *Proceedings Logic In Computer Science*, (1988)

[BS 81] S.Burris, H.P.Sankappanavar: *A Course in Universal Algebra*. Springer, (1981)

[Bür 86] H.-J.Bürckert: Some Relationships between Unification, Restricted Unification and Matching. *Proceedings 8th International Conference on Automated Deduction,* LNCS 230, pp. 514-524, (1986)

[BüSi 87] W.Büttner, H.Simonis: Embedding Boolean Expressions into Logic Programming. *Journal of Symbolic Computation*, (1987)

[Büt 88] W.Büttner: Unification in Post Algebras. *Technical Report*, (1988)

[CM 81] W.F.Clocksin, C.S.Mellish: *Programming in Prolog*. Springer, (1981)

[Fag 84] F.Fages: Associative Commutative Unification. *Proceedings 7th International Conference on Automated Deduction*, LNCS 170, pp. 194-208, (1984)

[FH 83, 86] F.Fages, G.Huet: Complete Set of Unifiers and Matchers in Equational Theories. *Proceedings of Colloquium on Trees in Algebra and Programming '83,* LNCS 159, pp. 205-220, (1983);
Journal of Theoretical Computer Science, pp. 189-200, (1986)

[Grä 79] G.Grätzer: *Universal Algebra*. Springer, (1979)

[HD 80] G. Huet and D.C. Oppen: Equations and Rewrite Rules: A Survey. In *Formal Languages: Perspectives and Open Problems,* Academic Press, (1980)

[Her 30] J.Herbrand: Recherches sur La Théorie de la Démonstration. *Travaux de la Society des Sciences et des Lettres de Varsovie*, No. 33, 128, (1930)

[Hero 86] A.Herold: Combination Of Unfication Algorithms. *Proceedings 8th International Conference on Automated Deduction,* pp. 450-469, (1986), also *Memo-Seki-85-3-KL,* Universität Kaiserslautern, (1985)

[Hers 68] I.N.Herstein: Non Commutative Rings. *Carus Mathematical Monograph,* Wiley, (1968)

[HsDe 85] J.Hsiang, N.Dershowitz: Rewrite Methods for Clausal and Nonclausal Theorem Proving. *Proceedings 5th Conference on Rewriting Techniques and Applications,* (1985)

[Hue 80] G.Huet: Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems. *Journal of the Association for Computing Machinery,* Vol.27, No.4, pp. 797-821, (1980)

[Kir 85] C.Kirchner: Méthodes et Outils de Conception Systématique d'Algorithmes d'Unification dans le Théories Équationelles. *Thése de Doctorat d'Etat en Informatique*, Université de Nancy, (1985)

[Kir 87] C.Kirchner: First Workshop on Unification Val d'Ajol. *Rapport Interne 87R34,* Université de Nancy, (1987).

[KKL 88] The Staff of Project KK-Lisp: The KK-Lisp Manual. *Technical Paper,* University of Kaiserslautern, (1988)

[Löw 08] L.Löwenheim: Über das Auflösungsproblem im Logischen Klassenkalkül. *Sitzungsbericht Berliner Mathematische Gesellschaft,* Nr. 7, pp. 89-94, (1908)

[LS 78] M.Livesey, J.Siekmann: Unification of Sets and Multisets. *SEKI technical report*, Universität Kaiserslautern, (1978)

[MM 79] A.Martelli, U.Montanari: An Efficient Unification Algorithm. *Technical Report,* University of Pisa, (1979)

[MM 82] A.Martelli, U.Montanari: An Efficient Unification Algorithm. *Association for Computing Machinery Transcript Programming Languages and Systems,* Vol.4, No. 2, pp. 258-282, (1982)

[MN 86] U.Martin, T.Nipkov: Unification in Boolean Rings. *Proceedings 8th International Conference on Automated Deduction*, LNCS 230, pp. 506-513, (1986)

[Ohl 88] H.-J.Ohlbach: The HADES Manual. *Technical Paper,* University of Kaiserslautern, (to appear)

[Plo 72] G.Plotkin: Building-in Equational Theories. *Machine Intelligence*, Vol.7,

pp. 73-90, (1972)

[Rob 65] J.A.Robinson: A Machine Oriented Logic Based on the Resolution Principle. *Journal of the Association for Computing Machinery,* Vol.12, No.1, pp. 23-41, (1965)

[Rud 74] S.Rudeanu: *Boolean Functions and Equations.* North Holland, (1974)

[Sch 90] E.Schröder: Vorlesungen über die Algebra der Logik. Leipzig, Vol. 1, (1890), Vol. 2, (1891, 1905), Vol. 3 (1895), reprint Chelsea, (1966)

[Schm 88] M.Schmidt-Schauß: Unification in a Combination of Arbitary Disjoint Equational Theories, *Proceedings 9th International Conference on Automated Deduction,* LNCS 310, pp. 378 - 396, (1988)

[Sie 78] J.Siekmann: Unification and Matching Problems. *PH.D. Thesis,* Essex University, (1978)

[Sie 88] J.Siekmann: Universal Unification. *Proceedings 7th International Conference on Automated Deduction,* USA, pp. 1-42, (1984)

[Sie 87] J.Siekmann: Universal Unification. *Journal of Symbolic Computation,* (1987)

[Sik 67] R.Sikorski: *Boolean Algebra.* Springer, (1967)

[Sim 87] H.Simonis: Application of Boolean Unification in Logic Programming. *Proceedings of 1st Workshop on Unification,* Rapport Interne 87R34, Université de Nancy, pp. 136-140, (1987)

[Sti 75] M.E.Stickel: A Complete Unification Algorithm for Associative-Commutative Functions. *Proceedings of 4th International Joint Conferences on Artificial Intelligence* , pp. 71-82, (1975)

[Sti 81] M.E.Stickel: A Complete Unification Algorithm for Associative-Commutative Functions. *Journal of the Association for Computing Machinery*, Vol.28, No. 3, pp. 423-444, (1981)

[Sto 35] M.H.Stone: Subsumption of the Theory of Boolean Algebras under the Theory of Rings. *Proceedings National Academy of Science,* USA, pp. 103-105, (1935)

[Sto 36] M.H.Stone: The Theory of Representations for Boolean Algebras. *Transcript American Mathematical Society,* Vol. 40, pp. 37-111, (1936)

[Tep 88] M.Tepp: Unification in a Combination of Arbitary Disjoint Equational Theories. *Diplomarbeit,* Universität Kaiserslautern, (to appear)

[Tid 86] E.Tidén: Unification in Combinations of Collapse-free Theories with Disjoin Sets of Function Symbols. *Proceedings 8th International Conference on Automated Deduction*, LNCS 230, pp. 431-449, (1986)

[Wei 72] H.Weinblatt: A New Search Algorithm for Finding the Simple Cycles of a Finite Directed Graph. *Journal of the Association for Computing Machinery,* Vol.19, No.1, pp. 43-56, (January 1972)

[Yel 85] K.Yelick: Combining Unification Algorithms for Confined Regular Equational Theories. *Proceedings 1th Conference on Rewriting Techniques and Applications,* pp. 365-380, (1985)

[Yel 87] K.Yelick: Unification in Combinations of Collapse-free Regular Theories. *Journal of Symbolic Computation,* Vol. 3, pp. 153-181, (1987)

9. APPENDIX

In the appendix three testsequences for the unification algorithms for pure Boolean Rings are presented. Method 1 denotes the "Variable Elimination" method, Method 2 the "Basis of Constants" method, Method 3 the "Basis of Coefficients" method. Method 2 and Method 3 are combined to Method 2/3, when the performance of the "Particular Solution" methods is identical.

We conclude the appendix with the results of four test runs of the unification algorithm for the combination of Boolean ring and free theory.

Testsequence 1:

Term		Method 1		Method 2		Method 3	
abbrev.		time (sec.)	cons cells	time (sec.)	cons cells	time (sec.)	cons cells
t	(+ (* x a) (* a b) (* a c))	0.032	231	0.086	565	0.092	506
t'	(+ t (* x y c) (* y b))	0.215	1341	0.293	1770	0.297	1821
t''	(+ t' (* x y z a) (* x z b))	1.745	8887	0.502	2942	0.564	3150
t'''	(+ t'' (* w x y a c))	13.155	49901	0.659	3904	0.713	4056

Testsequence 2:

Term		Method 1		Method 2		Method 3	
abbrev.		time (sec.)	cons cells	time (sec.)	cons cells	time (sec.)	cons cells
t	(+ (* x a b c) (* a d))	0.042	241	0.005	58	0.004	34
t'	(+ t (* y b e))	0.133	719	0.008	98	0.009	69
t''	(+ t' (* z c d e))	0.435	2035	0.012	121	0.026	140
t'''	(+ t'' (* v a c d f))	0.944	4045	0.027	228	0.077	208
t''''	(+ t''' (* w b f))	1.877	7792	0.030	240	0.116	476

Testsequence 3:

Term	Method 1			Method 2/3		
	time (sec.)	cons cells	|σ|	time (sec.)	cons cells	|σ|
(+ x a)	0.005	43	1	0.005	44	1
(+ x a (* x y))	0.121	824	2	0.085	581	2
(+ x a (* x y z) (* y z))	0.110	650	2	0.144	1037	3
(+ y a (* v x) (* v x y z) (* x y z) (* x y))	0.386	2312	3	0.266	2175	4
(+ y a (* v w x) (* v w x y z) (* v x y z) (* x y))	1.508	7340	4	0.391	2390	5
(+ y a (* u v w x y z) (* u v x z) (* v w x) (* v w x y z) (* x y))	2.985	13516	4	0.593	3474	6
(+ x a b)	0.009	76	1	0.007	50	1
(+ x a b (* x y))	0.172	1117	2	0.136	810	2
(+ x a b (* x y z) (* y z))	0.165	892	2	0.205	1397	3
(+ y a b (* v x) (* v x y z) (* x y z) (* x y))	0.747	3888	3	0.335	2099	4
(+ y a b (* v w x) (* v w x y z) (* v x y z) (* x y))	5.129	22168	4	0.503	3057	5
(+ y a b (* u v w x y z) (* u v x z) (* v w x) (* v w x y z) (* x y))	11.729	44070	4	0.805	8199	6
(+ x a b c)	0.011	91	1	0.008	56	1
(+ x a b c (* x y))	0.465	1704	2	0.168	1001	2
(+ x a b c (* x y z) (* y z))	0.215	2118	2	0.255	1659	3
(+ y a b c (* v x) (* v x y z) (* x y z) (* x y))	1.111	5736	3	0.390	2529	4
(+ y a b c (* v w x) (* v w x y z) (* v x y z) (* x y))	19.136	73020	4	0.624	3607	5
(+ y a b c (* u v w x y z) (* u v x z) (* v w x) (* v w x y z) (* x y))	33.351	115422	4	0.833	4925	6
(+ x a b c d)	0.013	106	1	0.009	62	1
(+ x a b c d (* x y))	0.317	1874	2	0.196	1192	2
(+ x a b c d (* x y z) (* y z))	0.234	1290	2	0.340	1973	3
(+ y a b c d (* v x) (* v x y z) (* x y z) (* x y))	1.616	8080	3	0.488	2963	4
(+ y a b c d (* v w x) (* v w x y z) (* v x y z) (* x y))	43.108	147067	4	0.723	4144	5
(+ y a b c d (* u v w x y z) (* u v x z) (* v w x) (* v w x y z) (* x y))	69.765	231864	4	0.991	5635	6

TEST COMBINATION ALGORITHM:

MACHINE: MYSTIC
DATE: 13.10.1988

TEST ENVIRONMENT:

BOOLEAN THEORY:
  function ADDITION --> +
  function MULTIPLICATION --> *
  constant UNIT ELEMENT --> 1
  constant ZERO ELEMENT --> 0

FREE THEORY:
  function F with arity 1

VARIABLES: U, V, W, X, Y, Z

FREE CONSTANTS: A, B, C, D, E

THE PROBLEM SPLITTED IN THE THEORYPARTS
ME 411
BOOLEAN | (+ (F A) (F B)) == (+ (F X) (F Y))

UNFOLDING OF ALIENS OF INITIAL SYSTEM
ME 411
BOOLEAN | (+ |V_418| |V_435|) == (+ |V_464| |V_481|)
ME 491
FREE | (F Y)
BOOLEAN | |V_481|
ME 474
FREE | (F X)
BOOLEAN | |V_464|
ME 445
FREE | (F B)
BOOLEAN | |V_435|
ME 428
FREE | (F A)
BOOLEAN | |V_418|

RENAMING OF INITIAL SYSTEM

```
ME 561
VARIABLES   | Y
FREE        | |V_520|
ME 554
VARIABLES   | X
FREE        | |V_510|
ME 411
BOOLEAN     | (+ |V_418| |V_435|) == (+ |V_464| |V_481|)
ME 491
FREE        | (F |V_520|)
BOOLEAN     | |V_481|
ME 474
FREE        | (F |V_510|)
BOOLEAN     | |V_464|
ME 445
FREE        | (F B)
BOOLEAN     | |V_435|
ME 428
FREE        | (F A)
BOOLEAN     | |V_418|
```

UNIFICATION OF FREE PART AND PREUNIFICATION OF BOOLEAN RING PART

```
ME 561
VARIABLES   | Y
FREE        | |V_520|
ME 554
VARIABLES   | X
FREE        | |V_510|
ME 411
BOOLEAN     | (+ |V_418| |V_435|) == (+ |V_464| |V_481|)
ME 491
FREE        | (F |V_520|)
BOOLEAN     | |V_481|
ME 474
FREE        | (F |V_510|)
BOOLEAN     | |V_464|
ME 445
FREE        | (F B)
BOOLEAN     | |V_435|
ME 428
FREE        | (F A)
BOOLEAN     | |V_418|
```

UNIFICATION OF BOOLEAN RING AND FREE PART
ME 561
VARIABLES | Y
FREE | |V_520|
ME 554
VARIABLES | X
FREE | |V_510|
ME 491
FREE | (F |V_520|)
BOOLEAN | |V_659|
ME 474
FREE | (F |V_510|)
BOOLEAN | (+ |V_659| |C_582| |C_592|)
ME 445
FREE | (F B)
BOOLEAN | |C_582|
ME 428
FREE | (F A)
BOOLEAN | |C_592|

identification is necessary
those identifications of two free terms are possible
IDENTIFICATIONPAIR: <(F |V_520|) (F B)>
IDENTIFICATIONPAIR: <(F |V_520|) (F A)>
IDENTIFICATIONPAIR: <(F |V_510|) (F B)>
IDENTIFICATIONPAIR: <(F |V_510|) (F A)>
IDENTIFICATIONPAIR: <(F |V_510|) (F |V_520|)>

the following identification of free terms IS NO SOLUTION <no cycle check>
IDENTIFICATIONLIST: < >
the following identification of free terms IS NO SOLUTION <no cycle check>
IDENTIFICATIONLIST: < (F |V_520|) (F B) >
the following identification of free terms IS NO SOLUTION <no cycle check>
IDENTIFICATIONLIST: < (F |V_520|) (F A) >
this identification of free terms is not possible:
IDENTIFICATIONLIST: < (F |V_520|) (F A) (F B) >
the following identification of free terms IS NO SOLUTION <no cycle check>
IDENTIFICATIONLIST: < (F |V_510|) (F B) >
the following identification of free terms IS A SOLUTION <no cycle check>
IDENTIFICATIONLIST: < (F |V_520|) (F A) > < (F |V_510|) (F B) >

TOPLEVEL CYCLE CHECK
NO CYCLES DETECTED

THIS IS A SOLUTION WITHOUT CYCLES
ME 805
VARIABLES | Y
FREE | A
ME 770
VARIABLES | X
FREE | B

the following identification of free terms IS NO SOLUTION <no cycle check>
IDENTIFICATIONLIST: < (F |V_510|) (F A) >
this identification of free terms is not possible:
IDENTIFICATIONLIST: < (F |V_510|) (F A) (F B) >
the following identification of free terms IS A SOLUTION <no cycle check>
IDENTIFICATIONLIST: < (F |V_520|) (F B) > < (F |V_510|) (F A) >

TOPLEVEL CYCLE CHECK
NO CYCLES DETECTED

THIS IS A SOLUTION WITHOUT CYCLES
ME 812
VARIABLES | Y
FREE | B
ME 770
VARIABLES | X
FREE | A

the following identification of free terms IS NO SOLUTION <no cycle check>
IDENTIFICATIONLIST: < (F |V_510|) (F |V_520|) >
the following identification of free terms IS NO SOLUTION <no cycle check>
IDENTIFICATIONLIST: < (F |V_510|) (F |V_520|) (F A) >
the following identification of free terms IS NO SOLUTION <no cycle check>
IDENTIFICATIONLIST: < (F |V_510|) (F |V_520|) (F B) >

the problem:
one multiequation:
(+ (F X) (F Y)) == (+ (F A) (F B))

one solution:
one multiequation:
Y == B
one multiequation:
X == A

one solution:
one multiequation:
Y == A
one multiequation:
X == B

THE PROBLEM SPLITTED IN THE THEORYPARTS

```
ME 411
BOOLEAN     | (* (F A) (F B)) == (* (F X) (F Y))
```

UNFOLDING OF ALIENS OF INITIAL SYSTEM

```
ME 411
BOOLEAN     | (* |V_418| |V_435|) == (* |V_464| |V_481|)
ME 491
FREE        | (F Y)
BOOLEAN     | |V_481|
ME 474
FREE        | (F X)
BOOLEAN     | |V_464|
ME 445
FREE        | (F B)
BOOLEAN     | |V_435|
ME 428
FREE        | (F A)
BOOLEAN     | |V_418|
```

RENAMING OF INITIAL SYSTEM

```
ME 561
VARIABLES   | Y
FREE        | |V_520|
ME 554
VARIABLES   | X
FREE        | |V_510|
ME 411
BOOLEAN     | (* |V_418| |V_435|) == (* |V_464| |V_481|)
ME 491
FREE        | (F |V_520|)
BOOLEAN     | |V_481|
ME 474
FREE        | (F |V_510|)
BOOLEAN     | |V_464|
ME 445
FREE        | (F B)
BOOLEAN     | |V_435|
ME 428
FREE        | (F A)
BOOLEAN     | |V_418|
```

UNIFICATION OF FREE PART AND PREUNIFICATION OF BOOLEAN RING PART

```
ME 561
VARIABLES   | Y
FREE        | |V_520|
ME 554
VARIABLES   | X
FREE        | |V_510|
ME 411
BOOLEAN     | (* |V_418| |V_435|) == (* |V_464| |V_481|)
ME 491
FREE        | (F |V_520|)
BOOLEAN     | |V_481|
ME 474
FREE        | (F |V_510|)
BOOLEAN     | |V_464|
ME 445
FREE        | (F B)
BOOLEAN     | |V_435|
ME 428
FREE        | (F A)
BOOLEAN     | |V_418|
```

UNIFICATION OF BOOLEAN RING AND FREE PART

```
ME 561
VARIABLES   | Y
FREE        | |V_520|
ME 554
VARIABLES   | X
FREE        | |V_510|
ME 491
FREE        | (F |V_520|)
BOOLEAN     | (+ |V_707| (* |V_707| |C_582| |C_592|) (* |C_582| |C_592|))
ME 474
FREE        | (F |V_510|)
BOOLEAN     | (+ |V_741|
                 (* |V_707| |V_741|)
                 (* |V_707| |V_741| |C_582| |C_592|)
                 (* |V_741| |C_582| |C_592|)
                 (* |C_582| |C_592|))
ME 445
FREE        | (F B)
BOOLEAN     | |C_582|
ME 428
FREE        | (F A)
BOOLEAN     | |C_592|
```

identification is necessary
those identifications of two free terms are possible
IDENTIFICATIONPAIR: <(F |V_520|) (F B)>
IDENTIFICATIONPAIR: <(F |V_520|) (F A)>
IDENTIFICATIONPAIR: <(F |V_510|) (F B)>
IDENTIFICATIONPAIR: <(F |V_510|) (F A)>
IDENTIFICATIONPAIR: <(F |V_510|) (F |V_520|)>

the following identification of free terms IS NO SOLUTION <no cycle check>
IDENTIFICATIONLIST: < >
the following identification of free terms IS NO SOLUTION <no cycle check>
IDENTIFICATIONLIST: < (F |V_520|) (F B) >
the following identification of free terms IS NO SOLUTION <no cycle check>
IDENTIFICATIONLIST: < (F |V_520|) (F A) >
this identification of free terms is not possible:
IDENTIFICATIONLIST: < (F |V_520|) (F A) (F B) >
the following identification of free terms IS NO SOLUTION <no cycle check>
IDENTIFICATIONLIST: < (F |V_510|) (F B) >
the following identification of free terms IS A SOLUTION <no cycle check>
IDENTIFICATIONLIST: < (F |V_520|) (F A) > < (F |V_510|) (F B) >

TOPLEVEL CYCLE CHECK
NO CYCLES DETECTED

THIS IS A SOLUTION WITHOUT CYCLES
ME 922
VARIABLES | Y
FREE | A
ME 950
VARIABLES | X
FREE | B

the following identification of free terms IS NO SOLUTION <no cycle check>
IDENTIFICATIONLIST: < (F |V_510|) (F A) >
this identification of free terms is not possible:
IDENTIFICATIONLIST: < (F |V_510|) (F A) (F B) >
the following identification of free terms IS A SOLUTION <no cycle check>
IDENTIFICATIONLIST: < (F |V_520|) (F B) > < (F |V_510|) (F A) >

TOPLEVEL CYCLE CHECK
NO CYCLES DETECTED

THIS IS A SOLUTION WITHOUT CYCLES
ME 950
VARIABLES | Y
FREE | B
ME 1065
VARIABLES | X
FREE | A

the following identification of free terms IS NO SOLUTION <no cycle check>

IDENTIFICATIONLIST: < (F |V_510|) (F |V_520|) >
the following identification of free terms IS NO SOLUTION <no cycle check>
IDENTIFICATIONLIST: < (F |V_510|) (F |V_520|) (F A) >
the following identification of free terms IS NO SOLUTION <no cycle check>
IDENTIFICATIONLIST: < (F |V_510|) (F |V_520|) (F B) >

the problem:
 one multiequation:
  (* (F X) (F Y)) == (* (F A) (F B))

one solution:
 one multiequation:
 Y == B
 one multiequation:
 X == A

one solution:
 one multiequation:
 Y == A
 one multiequation:
 X == B

THE PROBLEM SPLITTED IN THE THEORYPARTS
ME 363
VARIABLES | X
FREE | (F (* X Y))

UNFOLDING OF ALIENS OF INITIAL SYSTEM
ME 380
FREE | |V_370|
BOOLEAN | (* X Y)
ME 363
VARIABLES | X
FREE | (F |V_370|)

RENAMING OF INITIAL SYSTEM
ME 431
VARIABLES | Y
BOOLEAN | |V_409|
ME 380
FREE | |V_370|
BOOLEAN | (* |V_399| |V_409|)
ME 363
VARIABLES | X
FREE | (F |V_370|)
BOOLEAN | |V_399|

UNIFICATION OF FREE PART AND PREUNIFICATION OF BOOLEAN RING PART
ME 431
VARIABLES | Y
BOOLEAN | |V_409|
ME 380
FREE | |V_370|
BOOLEAN | (* |V_399| |V_409|)
ME 363
VARIABLES | X
FREE | (F |V_370|)
BOOLEAN | |V_399|

UNIFICATION OF BOOLEAN RING AND FREE PART
ME 431
VARIABLES | Y
BOOLEAN | |V_409|
ME 380
FREE | |V_370|
BOOLEAN | (* |V_399| |V_409|)
ME 363
VARIABLES | X
FREE | (F |V_370|)
BOOLEAN | |V_399|

identification is necessary

the following identification of free terms IS A SOLUTION <no cycle check>
IDENTIFICATIONLIST: < >

TOPLEVEL CYCLE CHECK
CYCLES ARE DETECTED IN FOLLOWING SYSTEM:
CYCLES: (|V_370| |C_459| |V_370|)

ME 496
VARIABLES | Y
BOOLEAN | |V_409|
ME 503
FREE | |V_370|
BOOLEAN | (* |V_409| |C_459|)
ME 510
VARIABLES | X
FREE | (F |V_370|)
BOOLEAN | |C_459|

WE TRY TO BREAK CYCLES AT FOLLOWING POINTS
TERM (* |V_409| |C_459|) ALIENS |C_459|
THE CYCLES ARE BREAKABLE

RECURSIVE CYCLE CHECK
NO CYCLES DETECTED

THIS IS A SOLUTION WITHOUT CYCLES
ME 647
VARIABLES | Y
BOOLEAN | (+ |V_637| (* |V_637| |C_459|))
ME 692
VARIABLES | X
FREE | (F 0)
BOOLEAN | |C_459|

the problem:
one multiequation:
X == (F (* X Y))

one solution:
one multiequation:
Y == (+ |V_637| (* |V_637| (F 0)))
one multiequation:
X == (F 0)

THE PROBLEM SPLITTED IN THE THEORYPARTS

```
ME 375
FREE        | (F X)
BOOLEAN     | (+ X (F Y))
```

UNFOLDING OF ALIENS OF INITIAL SYSTEM

```
ME 375
FREE        | (F X)
BOOLEAN     | (+ X |V_382|)
ME 392
FREE        | (F Y)
BOOLEAN     | |V_382|
```

RENAMING OF INITIAL SYSTEM

```
ME 484
VARIABLES   | X
FREE        | |V_421|
BOOLEAN     | |V_455|
ME 477
VARIABLES   | Y
FREE        | |V_411|
ME 375
FREE        | (F |V_421|)
BOOLEAN     | (+ |V_382| |V_455|)
ME 392
FREE        | (F |V_411|)
BOOLEAN     | |V_382|
```

UNIFICATION OF FREE PART AND PREUNIFICATION OF BOOLEAN RING PART

```
ME 484
VARIABLES   | X
FREE        | |V_421|
BOOLEAN     | |V_455|
ME 477
VARIABLES   | Y
FREE        | |V_411|
ME 375
FREE        | (F |V_421|)
BOOLEAN     | (+ |V_382| |V_455|)
ME 392
FREE        | (F |V_411|)
BOOLEAN     | |V_382|
```

UNIFICATION OF BOOLEAN RING AND FREE PART
ME 484
VARIABLES | X
FREE | |V_421|
BOOLEAN | |V_455|
ME 477
VARIABLES | Y
FREE | |V_411|
ME 375
FREE | (F |V_421|)
BOOLEAN | (+ |V_382| |V_455|)
ME 392
FREE | (F |V_411|)
BOOLEAN | |V_382|

identification is necessary
those identifications of two free terms are possible
IDENTIFICATIONPAIR: <(F |V_411|) (F |V_421|)>

the following identification of free terms IS A SOLUTION <no cycle check>
IDENTIFICATIONLIST: < >

TOPLEVEL CYCLE CHECK
CYCLES ARE DETECTED IN FOLLOWING SYSTEM:
CYCLES: (|V_421| |C_522| |V_421|)

ME 566
VARIABLES | X
FREE | |V_421|
BOOLEAN | (+ |C_512| |C_522|)
ME 573
VARIABLES | Y
FREE | |V_411|
ME 580
FREE | (F |V_411|)
BOOLEAN | |C_512|
ME 587
FREE | (F |V_421|)
BOOLEAN | |C_522|

WE TRY TO BREAK CYCLES AT FOLLOWING POINTS
TERM (+ |C_512| |C_522|) ALIENS |C_522|
THE CYCLES CAN'T BE BROKEN

the following identification of free terms IS A SOLUTION <no cycle check>
IDENTIFICATIONLIST: < (F |V_411|) (F |V_421|) >

TOPLEVEL CYCLE CHECK
NO CYCLES DETECTED

THIS IS A SOLUTION WITHOUT CYCLES
ME 573
VARIABLES | Y == X
FREE | 0

the problem:
one multiequation:
(+ X (F Y)) == (F X)

one solution:
one multiequation:
Y == X == 0