CENTRE de
RECHERCHE en
INFORMATIQUE de
NANCY

C. N. R. S. (laboratoire associé n° 262)
Université de Nancy I
Université de Nancy II
Institut National Polytechnique de Lorraine

Rapport Interne n° 87 R 34

First Workshop on Unification

VAL D' AJOL
18 au 20 mars 1987

Campus scientifique - Boite Postale n° 239 - 54506 VANDOEUVRE LES NANCY CEDEX

FOREWORD

The ***First Workshop on Unification*** held in Val d'Ajol, a small village in the Vosges in France, March 18-20, 1987 and has been organized by

Alexander Herold, Kaiserslautern University,
Jean-Pierre Jouannaud, LRI Orsay,
Claude Kirchner, CRIN LORIA Nancy,
Jörg Siekmann, Kaiserslautern University,
Gert Smolka, Kaiserslautern University.

Unification or equation solving is a field of computer science and symbolic computation that knows these last few years a huge development and a surge of interest. Both are related to the better understanding of the formal foundations of computer science and in particular to the increasing interest in programming languages based on formal logical concepts. In that context there was a great interest to organize a workshop in order to share the current knowledge in the field and to join together the unification community.

This report regroups the abstracts and the copies of the transparencies of the talks given during the workshop.

During the workshop it has been decided to organize an electronic forum chaired by Gert Smolka. This forum will allow to share information in the field, like open problems and abstracts of the relevant literature. People interested to be in the mailing list or who want to make contributions can write to Gert Smolka (smolka@uklirb.uucp).

The next workshop will be organized by Claude Kirchner (ckirchner@crin.uucp) and Gert Smolka (smolka@uklirb.uucp) and will probably hold in the same place during the first week of June 1988.

We gratefully acknowledge the financial support of the University of NANCYI and the logistic support of the University of Kaiserslautern and the Centre de Recherche en Informatique de Nancy.

Claude Kirchner: Nancy, April 87.

A LIST OF OPEN PROBLEMS

Here is a list of open problems in unification that have been collected by Pierre Lescanne during the workshop. The name of the person who posed the problem is specified when this person was identified, otherwise it is the full group who raised the problem.

1. Unification, i.e., the existence of a unifier, in permutative theories is decidable. A permutative theory is a theory with axioms of the form $s = t$, where s and t have exactly the same number of occurrences of operators and variables.

2. There exists a finite and complete unification algorithm for skeletal permutative theories. A skeletal permutative theory is a theory with axioms of the form $s = \sigma(t)$, where σ is a permutation over the variables of t (Jouannaud).

3. Is Rety's sufficiently large normalizing narrowing optimal? If not does an optimal normalizing narrowing exist? (Smolka, Lescanne).

4. In order sorted theories is the unification decidable if the theory has a presentation with linear signature? (Schmidt-Schauss)

5. Using Plaisted's test set method one can decide inductive reducibility. Is it true that if the normal form of any term in the test set is a unique term u, then the normal form of any ground term is u? (Comon, Jouannaud)

6. What can we say about the rationality of the narrowing tree?

7. Give counter-examples to
$$\begin{gathered} U_1 \subseteq M_1 \\ U_\infty \subseteq M_\infty \\ M_\infty \subseteq U_\infty \\ M_0 \subseteq U_0 \end{gathered}$$
U_1 (resp. M_1) is the class of theories with a unique most general unifier (resp. most general matcher) for each equation, U_∞ (resp. M_∞) is the class of theories with a possible infinite complete and minimal generator set of unifiers (resp. matchers) for an equation. $\subseteq$ is for "subclass of".

8. Under which conditions will the above inclusions hold? Almost collapse-freeness is such a condition, but is there a more general one?

9. A proof or counter-example for:
$$E \text{ almost collapse} - \text{free } \Rightarrow M_\infty \subseteq U_\infty$$
(Schmidt-Schauss)

10. The direct sum of two normalizing or weakly terminating term rewriting systems is normalizing. (Nipkow after Toyama).

Summaries of the talks given at the

FIRST WORKSHOP ON UNIFICATION VAL D'AJOL

18 AU 20 MARS 1987

Edited by Claude KIRCHNER

SCHEDULE OF THE FIRST WORKSHOP ON UNIFICATION

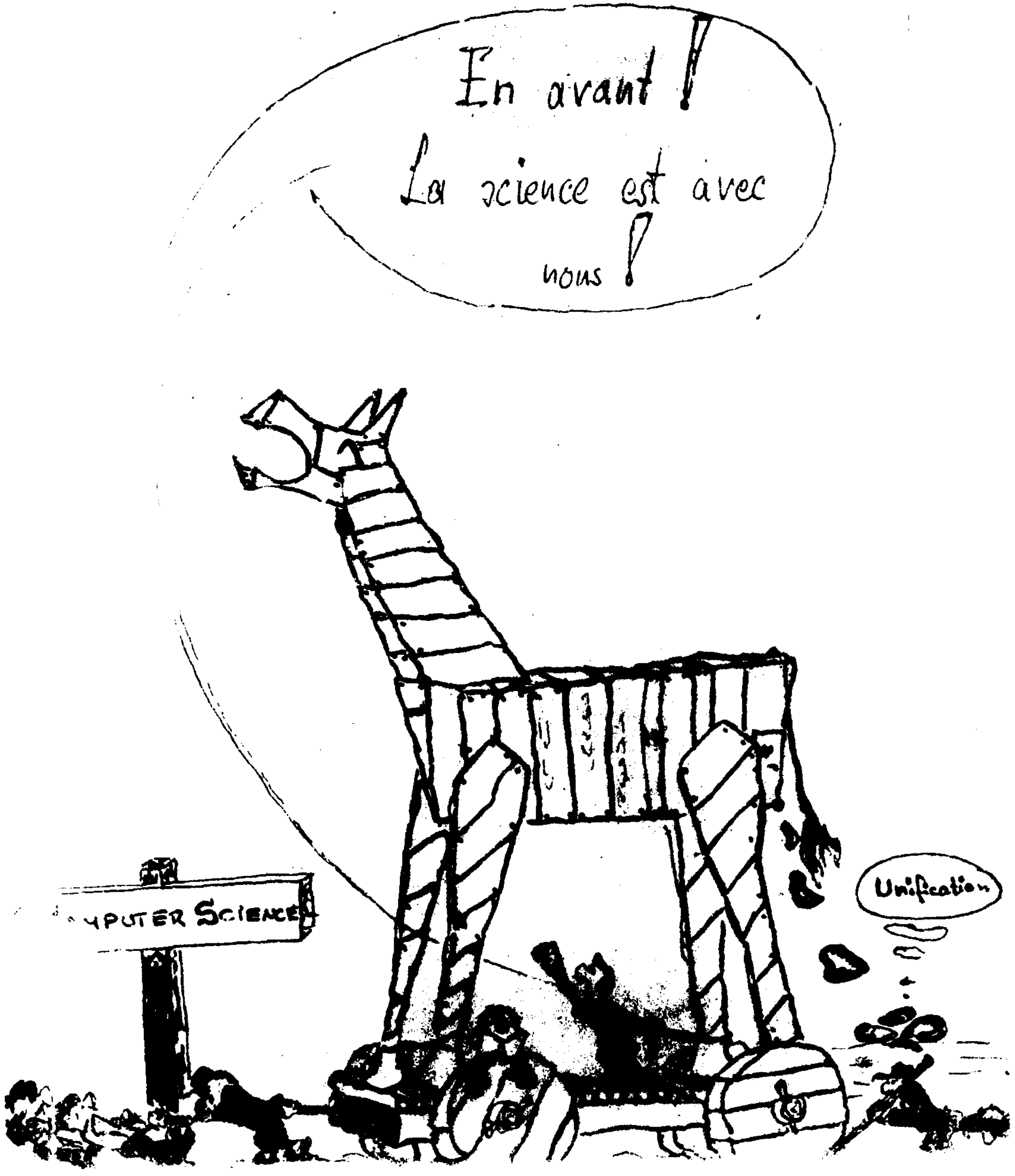
Wednesday, March 18, 1987 :

Thursday, March 19, 1987

Friday, March 20, 1987

WELLCOME TO

FIRST WORKSHOP ON UNIFICATION



EARLY HISTORY

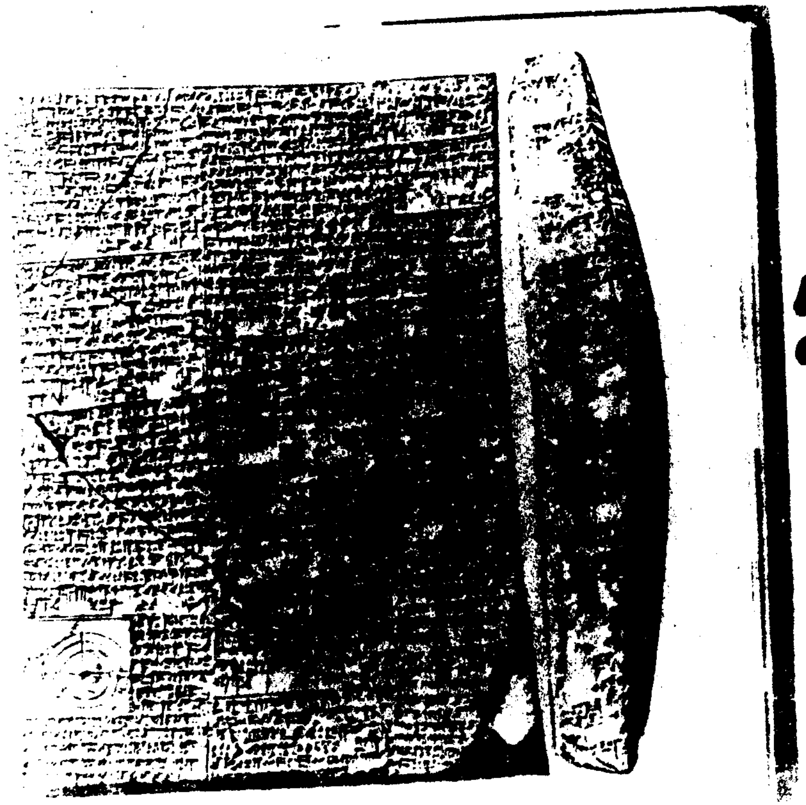
1920	EMIL POST, UNIFICATION ALGORI
1930	J. HERBRAND, UNIFICATION ALGORITH
1960	D. PRAWITZ, MOST GENERAL UNIFI
1963	M. DAVIS, 'LINKED CONJUNCTS'
1964	J. GUARD et al. T-UNIFICATION,
1965	A. ROBINSON, M.G.U., ALGORITH
1970	D. KNUTH, UNIFICATION IN T.F
1967	A. ROBINSON, T-UNIFICATION
1972	G. PLOTKIN, T-UNIFICATION
1975	J. SIEKMANN, T-UNIFICATION, HIER
1971	P. ANDREWS, HOL
1976	G. HUET, HOL, ∞-TERMS, ALGEB T-

Solving of Equations:

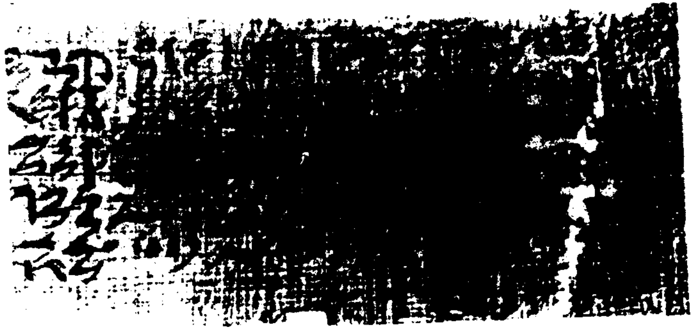
▷ [illegible] ANTOS OF ALEXANDRIA (~ [illegible] B

▷ [illegible] MATHEMATI[illegible]

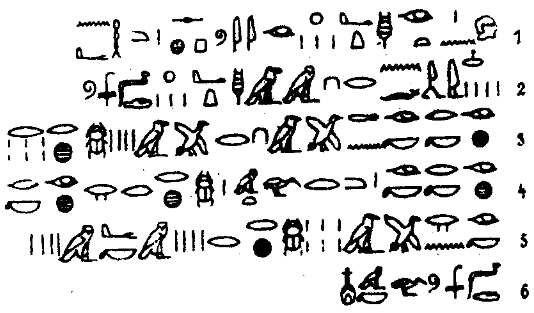
STONE PLATE with CUNEIFORM [illegible]
about 2000 to 3000 B.C.:



250 B.C.



PAPYRUS TEXT, ~ 1800
(demotian language)



TRA[illegible]

[illegible]

HIER[illegible]

FORM [illegible] BERECHNU[illegible] EINES HAUFEN, GERECHNET [illegible] ZU-SAMMEN MIT 4. ER [illegible] BIS 10. DER HA[illegible]

[illegible] = 10

THE FIELD TODAY:

(i) THE SPECIAL THEORY

- Special Unification and Matching Algorithms
- Order Sorted Unification
- Combination of Theories
- Universal Unification Algorithms
  Narrowing
  Decomposition
- Disunification, Antiunification, Paramo
- HIGHER ORDER UNIFICATION

(ii) THE GENERAL THEORY

- Unification Hierarchy
- Classification of Equational Theories
- Relation to Mathematical Structures
- Formal Foundations, Abstract Theory

[illegible] EVERY EQUATION:

$$a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0 = 0$$

A SOLUTION?

▲ DESCARTES (1596-165

▲ [illegible]

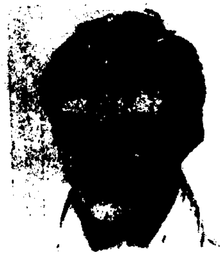
▲ GAUSS (177

MAIN THEOREM OF ALGEBRA

▶ 1799

▷ CAN EVERY SOLUTION BE EXPRESSED AS A RADI

NIELS HENRIK ABEL



E. GAL
(1811-

ART IV:

CURRENT ISSUES

4.1. SPECIAL THEORIES

- THE NEXT 700 UNIFICATION ALGORITHMS

- Complexity Results
- Hardware Realisations
- Unification For Logic Program Languages
- Disunification
- Universal Unification

4.2. Combination Of Algorithms

- variable abstraction
- constant abstraction
- multiequations

4.3 Unification In Sorted Lc

- basic order sorted
- polymorphic functions
- declarations

4.4. General Theory

- classification / hierarchy
- Ehrenfeucht sets

COMPUTER SCIENCE APPLICATIONS

- DATA BASE THEORY
- INFORMATION RETRIEVAL
- COMPUTER ALGEBRA
- FORMAL LANGUAGES: PARSING
- PROGRAMMING LANGUAGES
- STRING MATCHING: SNOBOL

AI-APPLICATIONS:

- AUTOMATED THEOREM PROVING "BUILT-IN EQUATIONS"
- LOGIC PROGRAMMING
- PATTERN INVOCATED PROCEDURES
- NATURAL LANGUAGE PROCESSING PATR-II
- VISION ×
- EXPERT SYSTEM

Foundations of Equational Deduction: A Categorical Treatment of Equational Proofs, Unification Algorithms and Critical Pair Completion

D.E. Rydeheard and J.G. Stell

December 1986

Abstract

Equational deduction is the process of replacing like for like using substitutivity and the equivalence properties of equality. It has a simple compositional structure which allows us to introduce ideas from category theory: categorical concepts correspond to those in equational deduction whilst constructions in category theory, such as colimits and free algebras, correspond to decision procedures and algorithms for solving equations. In particular, we

- show how equational deduction has a 2-category structure,
- derive algorithms for the unification of terms from general constructions of colimits,
- provide an abstract framework for solving equations in equational theories (equational unification),
- relate critical pair completions to constructions of free algebras.

A good deal of this can be realized as computer programs either as algorithms, in which case we provide an abstract analysis of their compositional structure, or as proof support systems based upon the primitive of composition of morphisms.

This is a preliminary announcement of results; much of it is work in progress.

CATEGORICAL FOUNDATIONS OF EQUATIONAL DEDUCTION:

Equational Proofs and Unification Algorithms.

David Rydeheard
John Stell
Rod Burstall

KEY IDEAS

Constructivity of category theory as tool in program design

Deductive systems as 2-categories

- Compositional structure of
  - Proofs
  - Substitutions
- Localization of variables - variable handling by limits and colimits
- Categorical constructions specialize to algorithms for equational deduction e.g. unification algorithms

TOPICS

Equational Deduction and 2-categories

Unification Algorithms derived from constructions of colimits

Equational unification and confluence

Categorical setting for combining unification algorith

Critical pair completion as free algebra construction

THE BASIC CATEGORY

-operator domain

- Objects: Sets (of variables)
- Morphisms: Term substitutions
  - + $X \Rightarrow Y$ in T_Ω is a function
  - + $X \Rightarrow T_\Omega(Y)$ ..... set of Ω-terms with variables in Y

Example

$$\begin{tikzcd}\{u, v\} \arrow[r, "{\begin{array}{l} u \mapsto [illegible](x,y) \\ v \mapsto n(q(h(x,y))) \end{array}}"'] & \{x, y\}\end{tikzcd}$$

[Kleisli 1965]

EQUATIONAL DEDUCTION AND 2-CATEGORIES

Set of rewrite rules $\mathcal{R}$ induce 2-category structure on $\mathbb{T}$

Example

$$\begin{tikzcd}[row sep=large] {} \arrow[r, bend left=40, "z \mapsto x \times y", ""{name=U, below}] \arrow[r, bend right=40, "z \mapsto y \times x"', ""{name=D, above}] & \{x, y\} \arrow[Rightarrow, from=U, to=D, "\text{commute}"]\end{tikzcd}$$

Rich compositional structure: captures that of equational deduction.

UNIFICATION ALGORITHMS AND CONSTRUCTIONS OF COLIMITS

FACT Coequalizers in are unifiers of sets of equations.

General categorical constructions of coequalizers yield the recursive part of unification algorithms:

Theorem 1. If a morphism $q: b \to c$ is the coequalizer of $f, g: a \to b$ and $c \to d$ is coequalizer of

$$\begin{tikzcd} & b \arrow[dr, "q"] & \\ {} \arrow[ur] \arrow[dr] & & c \\ & b \arrow[ur, "q"'] & \end{tikzcd}$$

then $b \to d$ is coequalizer of

$$\begin{tikzcd} a + a' \arrow[r, shift left, "{[6F]}"] \arrow[r, shift right, "{[g, g']}"'] & b. \end{tikzcd}$$

Theorem 2. For all epis $h: a'' \to a$, the morphism $q: b \to c$ is the coequali of $a \to b$ iff it is the coequaliz of

Also Considered

Solving equations in confluent theories

Combing unification algorithms (e.g Yelick 1985) as constructions of colimits in colimit categories

Free algebras and the iterative structure of critical pair completions

To Be Considered

Other extensions of unification, efficient algorithms, special purpose unification algorithms.

Manfred Schmidt-Schauß

On the Definition of the Unification Type of an Equational Theory

In this talk we propose and justify a definition of unification type of an equational theory as the property of the set of unifiers of a system of equations instead of a single equation. Most equational theories have the same unification type for both definitions. In particular, for theories of unification type unitary, finitary or nullary this is always true. For infinitary theories it makes a difference : We give an example of a theory T that has unification type infinitary if we consider only single equations and unification type nullary if we consider the unifiers of more than one equation.

On the Definition of Unification Type

M. Schmidt - Schauß

Usual Definition.:

The unification type of $\mathcal{E}$ is

i) 0 , iff there are terms s,t such that $\mu U_{\mathcal{E}}(s,t)$ does not exist

ii) 1,ω,∞ depending on the cardinality of the sets $\mu U_{\mathcal{E}}(s,t)$ for terms s,t.

Problems: The signature is not explicit.
Merge of substitutions.
Systems of equations ?

Boolean rings are unitary, but not unitary with free function symbols

AC1 is unitary, but finitary with free constants

Unification may become undecidable after addition of free constants

Proposal for a new Definition:

$\mathcal{E} = (\Sigma, E)$ (signature & axioms)
default for $\Sigma = \{\text{symbols in } E\}$

equation system
$\Gamma = \langle s_1 = t_1, \ldots, s_n = t_n \rangle_{\mathcal{E}}$

instead of a single pair
$s = t$

Def: The <u>unification type</u> of $\mathcal{E}$ is

i) 0 , iff there is an equation system Γ such that $\mu U_{\mathcal{E}}(\Gamma)$ does not exist

ii) $1, \omega, \infty$ depending on the cardinality of the sets $\mu U_{\mathcal{E}}(\Gamma)$ for equation systems Γ

The definitions above are equivalent

i) For type $1, \omega$ - theories

ii) For finite equational theories $1, \omega, \infty$

iii) If a free (or decomposable) function symbol of arity ≥ 2 is available.

iv) If an Ω-free function symbol of arity ≥ 2 is available.

old type 0 $\Rightarrow$ new type 0
new type $\infty \Rightarrow$ old type ∞.

! The definitions above are not equivalent for (old) type ∞.

Ω free: $f(s) =_E f(t) \Rightarrow s =_E t$

Example:

$\mathcal{E}$ is defined by the term rewriting system:

$$
\begin{array}{lll}
f_i(g_1(x)) & \rightarrow g_2(f_i(x)) & i = 1,2,3,4 \\
f_1(k_1(x)) & \rightarrow f_2(k_1(x)) & \\
f_3(k_2(x)) & \rightarrow f_4(k_2(x)) & \\
k_1(h(x)) & \rightarrow k_2(h(x)) & \\
g_1(k_2(h(l(x)))) & \rightarrow k_2(h(x)) & \\
f_1(k_2(h(x))) & \rightarrow f_2(k_2(h(x))) & \\
g_2(f_2(k_2(h(l(x))))) & \rightarrow f_2(k_2(h(x))) & \\
g_2(f_4(k_2(h(l(x))))) & \rightarrow f_4(k_2(h(x))) &
\end{array}
$$

$\mathcal{E}$ is regular, Ω-free and simple
and is of old unification type ∞ !

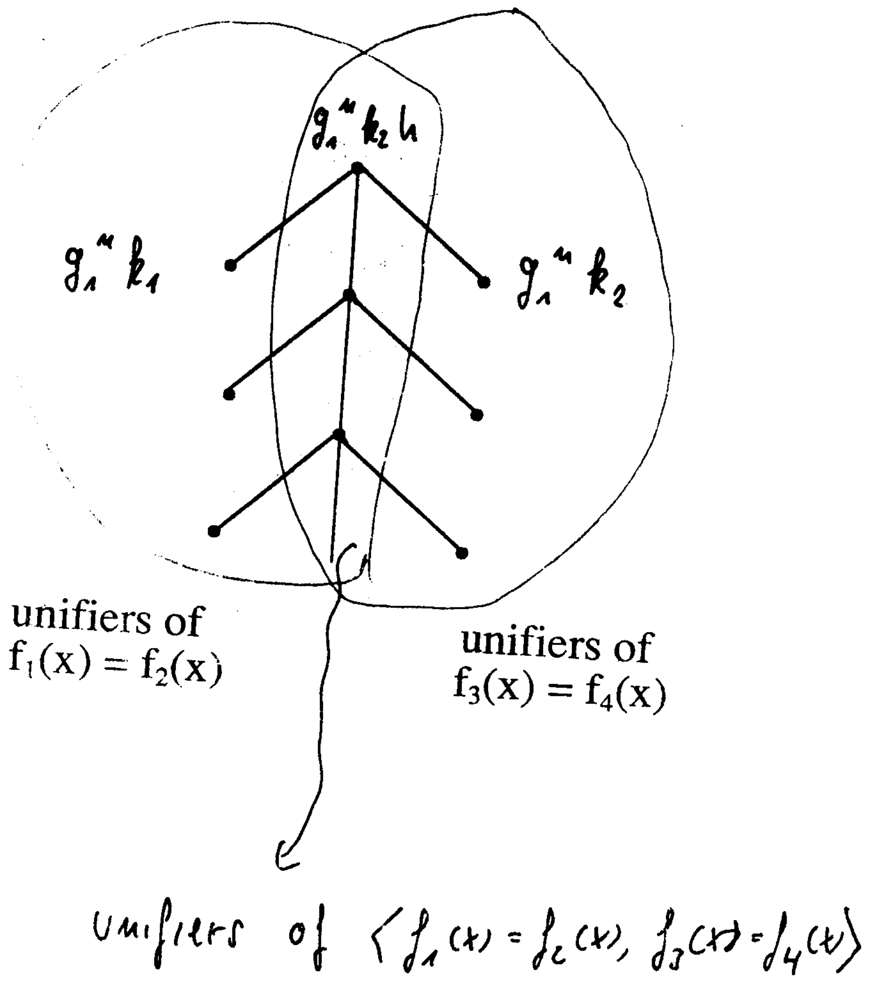
The equation system
$\langle f_1(x) = f_2(x),\ f_3(x) = f_4(x) \rangle$
has a complete set of unifiers:
$\{ \{ x \leftarrow g_1{}^n(k_2(h(z))) \},\ n \geq 0 \}$
This set has no minimal subset:

$$\{z \leftarrow l(z')\}\ g_1{}^n(k_2(h(z))) =_E g_1{}^{n-1}(k_2(h(z')))$$

regular : $s =_E t \Rightarrow V(s) = V(t)$

Ω-free : $f(s) =_E f(t) \Rightarrow s =_E t$

simple : $\langle x = t \rangle_E$ solvable,
iff $x \notin V(t)$


$g_1^u k_2 h$
$g_1^u k_1$
$g_1^u k_2$
unifiers of $f_1(x) = f_2(x)$
unifiers of $f_3(x) = f_4(x)$
unifiers of $\langle f_1(x) = f_2(x), f_3(x) = f_4(x) \rangle$

Proof (tedious)

1) E is Ω-free: $F(s) =_E F(t) \Rightarrow s =_E t$

2) E is simple:

$\langle x = t \rangle$ not unifiable if $x \in V(t)$

3. $f_1(s) =_E f_2(t) \Rightarrow s =_E t$

$f_3(s) =_E f_4(t) \Rightarrow s =_E t$

$k_1(s) =_E k_2(t) \Rightarrow s =_E t$

Induction + tedious case analysis

Different Subsumption Relations

Possibilities:

i) Subsumption $\leq_{\mathcal{E}}$ w.r.t free term algebra
$\sigma \leq_{\mathcal{E},f} \tau$ [W] iff $\exists\lambda.\ \lambda\sigma =_{\mathcal{E}} \tau$ [W]

ii) Subsumption $\leq_{\mathcal{E}}$ w.r.t initial term algebra
$\sigma \leq_{\mathcal{E},i} \tau$ iff
$\{$gr. inst. of $\tau\} \subseteq \{$gr. inst. of $\sigma\}$

Lemma: $\sigma \leq_{\mathcal{E},f} \tau$ [W] $\Rightarrow$ $\sigma \leq_{\mathcal{E},i} \tau$ [W]

Advantage of ii):
unification type may become finitary :
Unification in free idempotent semigroups is finitary for finitely many free constants.

Disadvantage of ii):
Unification is context-depenedent.
Not full compatible with combination

i) $\cong$ ii), if infinitely many free constants are available

Alexander Herold

Hans-Jürgen Bürckert

Manfred Schmidt-Schauß

A Classification of Equational Theories

The following classes of equational theories are presented : permutative, finite, simple, almost collapse free, collapse free, regular and Ω-free theories. The relationship between these classes are shown and the connection between these classes and the unification hierachy is pointed out.

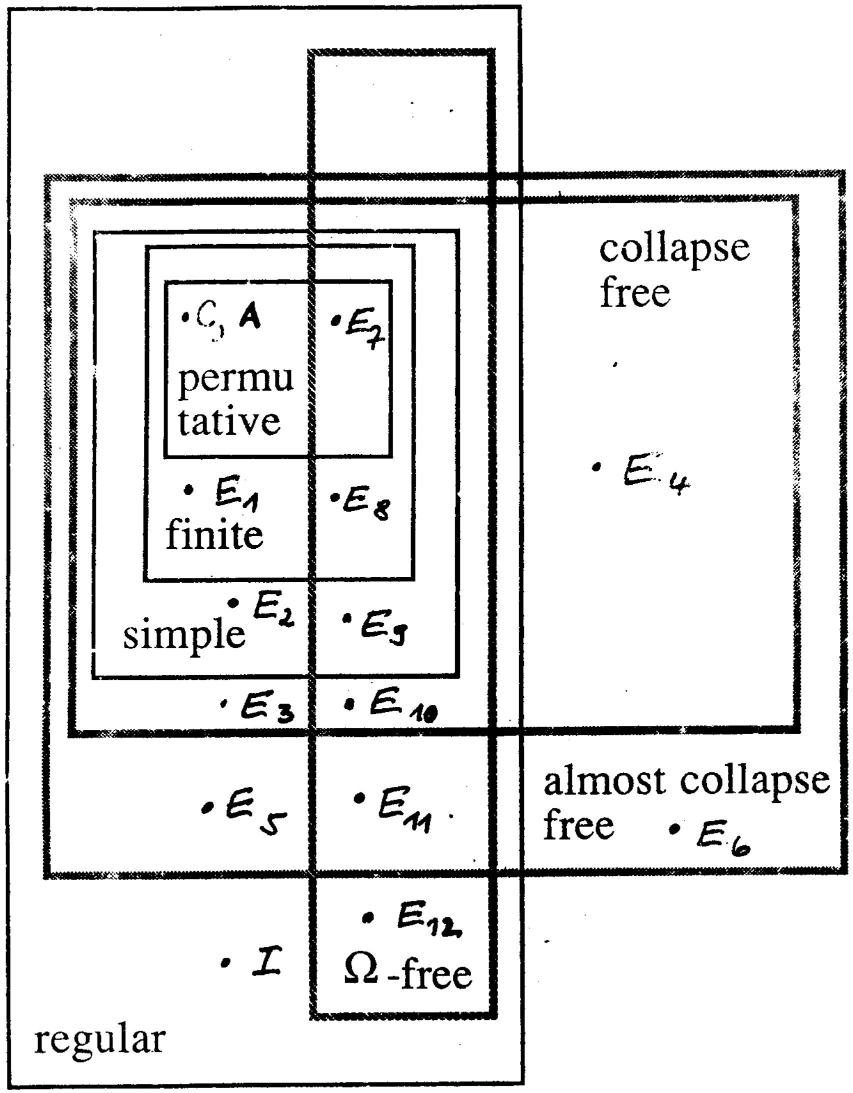
A Classification of Equational Theories

Hans-Jürgen Bürckert

Alexander Herold

Manfred Schmidt-Schauß

Universität Kaiserslautern



Permutative Theories

Lankford and Ballantyne 1977

An equational theory T is called **permutative** if for all equations s $=_T$ t the number of all symbols in s and in t is the same.

Examples: C, A, AC

Decidability: Yes (by an examination of the presentation)

Simple Theories

An equational theory T is said to be **simple** if for all equations $s =_T t$ the term s is not a subterm of t.

Examples: $E_2 = \{f(x \cdot f(y \cdot y)) = f(f(x \cdot x) \cdot y)\}$

Decidability: No

Properties:

- A variable x and a term t are T-unifiable iff $x \notin \mathbb{V}(t)$
- Simplicity is equivalent to strictness
- Simple theories are strongly complete
- There exist a simple theory that is of unification type nullary

Finite Theories

An equational theory T is said to be **finite** if every equivalence class induced by the congruence $=_T$ is finite.

Examples: $E_1 = \{f(a) = f(b)\}$

Decidability: No (Narendran, O'Dúnlaing, Rolletschek)

Properties:

- Minimal sets of unifiers always exist

Collapse Free Theories

An equational theory T is said to be **collapse free** if there is no equation of the form $x =_T t$.

Examples: $E_3 = \{f(x\ f(y\ y)) = f(f(x\ x)\ y)\}$,

$E_4 = \{x * 0 = 0\}$

Decidability: Yes (by an examination of the presentation)

Properties:

- The equivalence class of a variable only contains this variable

Almost Collapse Free Theories

Hans-Jürgen Bürckert 1986.

An equational theory T is said to be **almost collapse free** if there are no projection equations of the form $x_i =_T f(x_1 \dots x_i \dots x_n)$.

Examples: $E_5 = \{f(a) = f(b), g(x) = x\}$,

$E_4 = \{g(x\ y) = x)\}$

Decidability: No (reducing to a Markov property)

Properties:

- The same unification behaviour as collapse free theories (collapse equations can be dropped out by rewriting)

Regular Theories

An equational theory T is said to be **regular** if for all equations $s =_T t$ the set of variables occurring in s and t is the same.

Examples: $E_3 = \{f(x\ x) = x\}$

Decidability: Yes (by an examination of the presentation)

Properties:

- All terms in a equivalence class contain the same variable
- Minimal set of matchers always exist

Ω-free Theories

Szabó 1982

An equational theory T is said to be **Ω-free** if $f(s_1 \dots s_n) =_T f(t_1 \dots t_n)$ implies $s_i =_T t_i$ for all $i = 1,\dots,n$.

Examples:

$E_7 = \{f(g(a)) = g(f(a))\}$ (permutative)

$E_8 = \{f(a) = g(b)\}$ (finite)

$E_9 = \{f(g(h(x) = g(x)\}$ (simple)

$E_{10} = \{f(a\ a) = a\}$ (collpase free)

$E_{11} = \{f(g(x)) = x, f(x) = x\}$ (almost collpase free)

$E_{12} = \{f(g(x)) = x, g(f(x)) = x\}$ (regular)

Decidability: No (reducing to a Markov property)

Properties:

- Ω–free theories are exactly the regular unitary matching theories
- There exist a Ω-free theory that is of unification type nullary

The Unification Hierarchy

- **unitary** if μU_T is always a singleton or empty
- **finitary** if μU_T is always a finite set
- **infinitary** if μU_T is an infinite set for some problem
- **nullary** if μU_T does not exist for some problem

Decidability: No (reducing to a Markov property)

Is it possible to characterize nullary theories?

Remember: Finite theories are never nullary, since the instance relation is Noetherian.

An equational theory is **Noetherian** if the instance relation $\leq_T [W]$ is Noetherian on substitutions.

Hence finite theories are Noetherian, but the converse is false, but remark: There exists a finitary theory that is not Noetherian

RIGID E-UNIFICATION

Jean H. Gallier
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, Pa 19104

Abstract: Rigid E-Unification is a restricted type of E-unification that comes up naturally in generalizing the method of matings due to Andrews to first-order languages with equality. Let $E = \{(s_1 \doteq t_1), \ldots, (s_m \doteq t_m)\}$ be a finite set of equations, and $(u \doteq v)$ any equation.

Problem: It is decidable whether there is some substitution θ such that the set $\{\theta(s_1 \doteq t_1), \ldots, \theta(s_m \doteq t_m), \neg\theta(u \doteq v)\}$ is unsatisfiable? Equivalently, denoting by $\overset{*}{\Longleftrightarrow}_{\theta(E)}$ the least congruence induced by $\theta(E)$, treating the equations in $\theta(E)$ as ground equations, does $\theta(u) \overset{*}{\Longleftrightarrow}_{\theta(E)} \theta(v)$ hold, for some substitution θ?

Any substitution θ satisfying the above property is an E-unifier of u and v. However, the equations in E are used in a restricted fashion. Contrary to E-unification, in which there is no bound on the number of instances of the equations in E used to show that $\theta(u) \overset{*}{\Longleftrightarrow}_E \theta(v)$, in our situation, *only* the m instances in $\theta(E)$ can be used. For this reason, we call a substitution satisfying our problem a ***rigid E-unifier***.

We show that rigid E-unification is NP-complete in some nontrivial subcases and we conjecture that it is decidable in general.

RIGID E-UNIFICATION AND EQUATIONAL MATINGS

Jean H. Gallier
Joint work with Stan Raatz and Wayne Snyder

Department of Computer and Information Science
University of Pennsylvania
Philadelphia, Pa 19104

RIGID E-UNIFICATION AND EQUATIONAL MATINGS

MAIN GOAL: GENERALIZE ANDREWS'S METHOD OF MATINGS TO (FIRST-ORDER) LANGUAGES WITH EQUALITY

THEORETICAL BASIS: REDUCE THE UNSATISFIABILITY OF A *QUANTIFIED* FIRST-ORDER SENTENCE TO THE UNSATISFIABILITY OF A *QUANTIFIER-FREE* FORMULA, VIA A SEMANTIC VERSION OF *HERBRAND'S THEOREM*

- CASE 1: LANGUAGES WITHOUT EQUALITY
- TRADITIONAL CASE: PRENEX UNIVERSAL SENTENCES (AFTER SKOLEMIZATION)

$$\forall x_1 \ldots \forall x_n B, \quad B \text{ quantifier-free.}$$

SKOLEM-HERBRAND-GÖDEL THEOREM:

$A = \forall x_1 \ldots \forall x_n B$ is unsatisfiable iff there exist some *ground* substitutions $\sigma_1, \ldots, \sigma_k$ such that

$$C = \sigma_1(B) \wedge \ldots \wedge \sigma_k(B)$$

is unsatisfiable.

C is ~~a tautology,~~ unsatisfiable this is decidable (resolution, tableau systems, Gentzen systems, method of vertical paths, etc ...).

ANDREWS'S VERSION OF THE SKOLEM-HERBRAND-GÖDEL THEOREM

WOULD BE NICER IF A *SINGLE* SUBSTITUTION σ COULD BE USED

THIS IS POSSIBLE. USE ANDREWS'S *COMPOUND INSTANCES*

FOR SIMPLICITY, ALSO ASSUME FORMULAE IN *NEGATION NORMAL FORM (nnf)*

A *literal* is either an atomic formula or the negation of an atomic formula.

A formula A is in *nnf* iff either

(1) A is a literal, or

(2) $A = (B \vee C)$, where B and C are in nnf, or

(3) $A = (B \wedge C)$, where B and C are in nnf.

Let A be a universal sentence in nnf. The set of *compound instances* (c-instances) of A is defined inductively as follows:

(i) If A is either a ground atomic formula B or the negation B of a ground atomic formula, then A is its only c-instance;

(ii) if A is of the form $(B * C)$, where $* \in \{\vee, \wedge\}$, for any c-instance H of B and c-instance K of C, $(H * K)$ is a c-instance of A;

(iii) If A is of the form $\forall x B$, for any $k \geq 1$ closed terms $t_1, \ldots, t_k$, if H_i is a c-instance of $B[t_i/x]$ for $i = 1, \ldots, k$, then $H_1 \wedge \ldots \wedge H_i$ is a c-instance of A.

Theorem 1 (Andrews's version of the S-H-G theorem) Given a universal sentence A in nnf, A is unsatisfiable iff some c-instance C of A is unsatisfiable.

HOW DO WE GENERATE COMPOUND INSTANCES *NICELY*?

NOTION OF *AMPLIFICATION* (ANDREWS'S)

C is obtained from B by *quantifier duplication* iff C results from B by replacing some subformula $\forall x M$ of B by $(\forall x M \wedge \forall x M)$

If $C \Rightarrow C_2, \ldots, C_{n-1} \Rightarrow C_n$, with $B = C_1$, $C = C_n$ and C_{i+1} is obtained from C_i by quantifier duplication, $1 \leq i < n$, C is obtained from B by some *sequence of quantifier duplications*

If $A \Rightarrow^* B$ by some sequence of quantifier duplications

C is a rectified sentence equivalent to B,

D obtained from C by deleting the quantifiers in C,

then D is an *amplification* of A

Lemma 2 Given a universal sentence A in nnf, C is a c-instance of A iff there is some amplification D of A and some (ground) substitution θ such that $C = \theta(D)$

VERTICAL PATHS

FROM ANDREWS'S VERSION OF THE S-H-G THEOREM AND THE PREVIOUS LEMMA, WE HAVE:

Theorem 3 (Andrews) Given a universal sentence A in nnf, A is unsatisfiable iff there is some amplification D of A and some (ground) substitution σ such that $\sigma(D)$ is unsatisfiable.

HENCE, WE NEED A METHOD FOR SHOWING THAT GIVEN A QUANTIFIER FREE FORMULA D, THERE IS SOME SUBSTITUTION σ SUCH THAT $\sigma(D)$ IS UNSATISFIABLE

USE VERTICAL PATHS AND MATINGS

Let A be a quantifier free formula in nnf. The set $vp(A)$ of *vertical paths in A* is the set of sets of literals defined inductively as follows

If A is a literal, then $vp(A) = \{\{A\}\}$;

If $A = (B \wedge C)$ then $vp(A) = \{\pi_1 \cup \pi_2 \mid \pi_1 \in vp(B), \pi_2 \in vp(C)\}$;

If $A = (B \vee C)$ then $vp(A) = vp(B) \cup vp(C)$.

Lemma 4 Given a quantifier-free formula A in nnf, A is unsatisfiable iff every vertical path in A is unsatisfiable.

MATINGS

FOR LANGUAGES WITHOUT EQUALITY, A VERTICAL PATH $\{L_1, \ldots L_m\}$ IS UNSATISFIABLE IFF TWO OF THE LITERALS L_i, L_j ARE COMPLEMENTARY.

IF THE FORMULA IS OF THE FORM $\sigma(D)$, THIS MEANS THAT THERE ARE LITERALS $\sigma(L_i)$ and $\neg\sigma(L_j)$ SUCH THAT

$$\sigma(L_i) = \sigma(L_j).$$

HENCE σ IS A *UNIFIER* OF L_i AND L_j.

THIS LEADS TO *MATINGS*

Definition Given a quantifier-free formula A in nnf, a *mating* for A is a pair $\mathcal{M} = \langle MS, \sigma \rangle$, where

- MS is a set of pairs of literals of opposite signs (in A), and
- σ is a substitution such that, for every pair $(L, \neg L') \in MS$,

$$\sigma(L) = \sigma(L').$$

A mating is *p-acceptable* iff every vertical path (in A) contains some mated pair $(L, \neg L') \in MS$.

Lemma 5 (Andrews) Given a quantifier-free formula A in nnf, we have.

(1) Given a substitution θ, if $\theta(A)$ is unsatisfiable, then there is a p-acceptable mating $\mathcal{M}$ for A.

(2) If $\mathcal{M}$ is a p-acceptable mating for A with associated substitution σ_M, then $\sigma_M(A)$ is unsatisfiable.

CASE 2: LANGUAGES WITH EQUALITY

Andrews's version of the S-H-G theorem (theorem 3) can be generalized to languages with equality (nontrivial):

Theorem 3' (Gallier) Given a universal sentence A in nnf, A is unsatisfiable iff there is some amplification D of A and some (ground) substitution σ such that $\sigma(D)$ is unsatisfiable.

the lemma on vertical paths can also be generalized to languages with equality (trivial).

Lemma 4' Given a quantifier-free formula A in nnf, A is unsatisfiable iff every vertical path in A is unsatisfiable.

DIFFICULTY: IT IS NO LONGER TRIVIAL TO CHECK THAT A VERTICAL PATH IS UNSATISFIABLE.

BUT IT IS POSSIBLE: USE CONGRUENCE CLOSURE (KOZEN).

FIRST, REPLACE EVERY NONEQUATIONAL ATOM $P(t_1, \ldots, t_n)$ BY THE EQUATION $P(t_1, \ldots, t_n) \equiv T$.

(USE A TWO SORTED EQUATIONAL LANGUAGE)

CONGRUENCE CLOSURE

THEN, A VERTICAL PATH π IS OF THE FORM

$$\{(s_1 \doteq t_1), \ldots, (s_m \doteq t_m), \neg(s'_1 \doteq t'_1), \ldots, \neg(s'_n \doteq t'_n)\}.$$

TO CHECK THAT π IS UNSATISFIABLE, USE THE *CONGRUENCE CLOSURE* METHOD (KOZEN, 1976, OPPEN AND NELSON, 1979).

Let $TERMS(\pi)$ = set of all subterms of terms in π.

Construct labeled directed graph G_π as follows:

- Nodes of $G_\pi = TERMS(\pi)$.
- Node $f(t_1, \ldots, t_n)$ is labeled with f.
- For each node $f(t_1, \ldots, t_n)$, there is an edge from $f(t_1, \ldots, t_n)$ to each t_i.

A relation $\simeq$ on the set of nodes of G_π is *congruential* iff, for any two nodes $f(s_1, \ldots, s_n)$ and $f(t_1, \ldots, t_n)$, if

$$s_i \simeq t_i, \quad 1 \leq i \leq n,$$

$$f(s_1, \ldots, s_n) \simeq f(t_1, \ldots, t_n).$$

Given a vertical path

$$\pi = \{(s_1 \doteq t_1), \ldots, (s_m \doteq t_m), \neg(s'_1 \doteq t'_1), \ldots, \neg(s'_n \doteq t'_n)\},$$

let

$$E = \{(s_1 \doteq t_1), \ldots, (s_m \doteq t_m)\}.$$

Lemma 6 (Kozen) There is a smallest congruential equivalence relation containing E. It is called the *congruence closure* of E, denoted as $\overset{*}{\cong}_E$.

Theorem 7 (Kozen, Gallier) π is unsatisfiable iff for some j, $1 \leq$

$$s'_j \overset{*}{\cong}_E t'_j.$$

The congruence closure $\overset{*}{\cong}_E$ can be computed in polynomial time. Algorithms of Kozen, Oppen and Nelson: $O(n^2)$. Algorithm of Downey, Sethi and Tarjan: $O(n \log n)$.

We are now in a position to define equational matings.

EQUATIONAL MATINGS

Definition Let A be a quantifier-free formula in nnf. An *equational mating $\mathcal{M}$ for A* is a pair $\langle MS, \sigma \rangle$, where

- MS is a set of sets of literals called *mated sets* and
- σ is a substitution, such that,
- each mated set is a subset of some vertical path $\pi \in vp(A)$ and is of the form

$$\{(s_1 \doteq t_1), \ldots, (s_m \doteq t_m), \neg(s \doteq t)\} \subseteq \pi,$$

where $m \geq 0$, and,

- for every mated set $\{(s_1 \doteq t_1), \ldots, (s_m \doteq t_m), \neg(s \doteq t)\} \in MS$, the set of literals

$$\{\sigma(s_1 \doteq t_1), \ldots, \sigma(s_m \doteq t_m), \neg\sigma(s \doteq t)\}$$

is unsatisfiable.

An equational mating $\mathcal{M}$ is a *refutation mating* iff $\sigma_{\mathcal{M}}(A)$ is unsatisfiable.

An equational mating $\mathcal{M}$ is *p-acceptable* iff, for every path $\pi \in vp(A)$, there is some mated set

$$\{(s_1 \doteq t_1), \ldots, (s_m \doteq t_m), \neg(s \doteq t)\} \in \mathcal{M},$$

such that

$$\{(s_1 \doteq t_1), \ldots, (s_m \doteq t_m), \neg(s \doteq t)\} \subseteq \pi.$$

Lemma 8 (Andrews, Gallier) Given a quantifier-free formula A in nnf, we have:

(1) Given a substitution θ, if $\theta(A)$ is unsatisfiable, then there is a p-acceptable mating $\mathcal{M}$ for A.

(2) If $\mathcal{M}$ is a p-acceptable mating for A with associated substitution $\sigma_{\mathcal{M}}$, then $\sigma_{\mathcal{M}}(A)$ is unsatisfiable.

Theorem 9 (Andrews, Gallier) Given a universal sentence A in nnf, A is unsatisfiable iff some amplification D of A has a p-acceptable mating.

HOW DO WE FIND EQUATIONAL MATINGS? FIRST, EXAMPLE ILLUSTRATING THEOREM 9.

Example: Monoid such that $x^2 = 1$ for all x.

$$
\begin{aligned}
&\forall x \forall y \forall z(*(x, *(y, z)) \doteq *(*(x, y), z))) \wedge && (1)\\
&\forall u(*(u, 1) \doteq u) \wedge && (2)\\
&\forall v(*(1, v) \doteq v) \wedge && (3)\\
&\forall w(*(w, w) \doteq 1) \wedge && (4)\\
&\neg(*(a, b) \doteq *(b, a)). && (5)
\end{aligned}
$$

We want to show that such a monoid is commutative.

Consider the following amplification D of A and the set MS consisting of one set of literals.

$$
\begin{aligned}
D = &(*(u_1, 1) \doteq u_1)\\
&\wedge\ (*(w_1, w_1) \doteq 1)\\
&\wedge\ (*(x_1, *(y_1, z_1)) \doteq *(*(x_1, y_1), z_1)))\\
&\wedge\ (*(x_2, *(y_2, z_2)) \doteq *(*(x_2, y_2), z_2)))\\
&\wedge\ (*(w_2, w_2) \doteq 1)\\
&\wedge\ (*(1, v_1) \doteq v_1)\\
&\wedge\ (*(x_3, *(y_3, z_3)) \doteq *(*(x_3, y_3), z_3)))\\
&\wedge\ (*(x_4, *(y_4, z_4)) \doteq *(*(x_4, y_4), z_4)))\\
&\wedge\ (*(w_3, w_3) \doteq 1)\\
&\wedge\ \neg(*(a, b) \doteq *(b, a)).
\end{aligned}
$$

$$
\begin{aligned}
MS = \{\{&(*(u_1, 1) \doteq u_1),\\
&(*(w_1, w_1) \doteq 1),\\
&(*(x_1, *(y_1, z_1)) \doteq *(*(x_1, y_1), z_1))),\\
&(*(x_2, *(y_2, z_2)) \doteq *(*(x_2, y_2), z_2))),\\
&(*(w_2, w_2) \doteq 1),\\
&(*(1, v_1) \doteq v_1),\\
&(*(x_3, *(y_3, z_3)) \doteq *(*(x_3, y_3), z_3))),\\
&(*(x_4, *(y_4, z_4)) \doteq *(*(x_4, y_4), z_4))),\\
&(*(w_3, w_3) \doteq 1),\\
&\neg(*(a, b) \doteq *(b, a))\}\}.
\end{aligned}
$$

We use the substitution

$$
\begin{aligned}
&[a/u_1, (a * b)/w_1, a/x_1, (a * b)/y_1, (a * b)/z_1,\\
&a/x_2, a/y_2, b/z_2, a/w_2, b/v_1,\\
&b/x_3, (a * b)/y_3, b/z_3, a/x_4, b/y_4, b/z_4, b/w_3].
\end{aligned}
$$

We claim that $\langle MS, \theta \rangle$ is a mating for D. For simplicity of notation let adopt infix notation, and denote $*(s, t)$ as $s * t$. Then, we have:

$$
\begin{aligned}
a * b &= \\
&= \{a * 1\} * b && \text{by (2)}\\
&= \{a * [(a * b) * (a * b)]\} * b && \text{by (1)}\\
&= \{[a * (a * b)] * (a * b)\} * b && \text{by (1)}\\
&= \{[(a * a) * b] * (a * b)\} * b && \text{by (1)}\\
&= \{[1 * b] * (a * b)\} * b && \text{by (4)}\\
&= \{b * (a * b)\} * b && \text{by (3)}\\
&= b * \{(a * b) * b\} && \text{by (1)}\\
&= b * \{a * (b * b)\} && \text{by (1)}\\
&= b * \{a * 1\} && \text{by (4)}\\
&= (b * a), && \text{by (2)}
\end{aligned}
$$

which shows that $\langle MS, \theta\rangle$ is a p-acceptable mating for D (there is a single vertical path in D).

RIGID E-UNIFICATION

Recall the main condition for being an equational mating:

For each mated set $\{(s_1 \doteq t_1), \ldots, (s_m \doteq t_m), \neg(s \doteq t)\} \in \mathcal{M}$, the set

$$\{\sigma(s_1 \doteq t_1), \ldots, \sigma(s_m \doteq t_m), \neg\sigma(s \doteq t)\}$$

is unsatisfiable.

This implies that σ is an E-unifier of s and t modulo the set of equations $\{(s_1 \doteq t_1), \ldots, (s_m \doteq t_m)\}$. But it is a much stronger condition.

Definition Let $E = \{(s_1 \doteq t_1), \ldots, (s_m \doteq t_m)\}$ be a finite set of equations, and let $Var(E) = \bigcup_{(s \doteq t) \in E} Var(s \doteq t)$. A substitution θ is a *rigid E-unifier of u and v modulo E* iff

(1) (idempotence) $I(\theta) \cap D(\theta) = \emptyset$, and $D(\theta) \subseteq Var(E) \cup Var(u) \cup Var(v)$;

(2) The set

$$\{\theta(s_1 \doteq t_1), \ldots, \theta(s_m \doteq t_m), \neg\theta(u \doteq v)\}$$

is unsatisfiable. Equivalently, the equation $\theta(u \doteq v)$ is a consequence of the set $\{\theta(s_1 \doteq t_1), \ldots, \theta(s_m \doteq t_m)\}$ by the congruence closure method.

The property of being a rigid E-unifier constrains the use of the equations considerably. For E-unification, there is no bound on the number of instances of equations in E used in showing that $\theta(u)$ and $\theta(v)$ are congruent modulo E. On the other hand, for rigid E-unification, only the equations in the set $\{\theta(s_1 \doteq t_1), \ldots, \theta(s_m \doteq t_m)\}$ can be used (as ground equations).

A rigid E-unifier is an E-unifier, but the converse is not true

Example: Let $E = \{(f(a) \doteq a)\ (1), (f(a) \doteq x)\ (2)\}$, $u = x$ and $v = g(x)$. The substitution $\sigma = [g(a)/x]$ is a rigid unifier of u and v, because,

$$\begin{aligned} g(g(a)) &\Longrightarrow g(f(a)) && \text{by (2)} \\ &\Longrightarrow g(a). && \text{by (1)} \end{aligned}$$

The substitution $[a/x]$ is an E-unifier of u and v (rename x as y in the equation $(f(a) \doteq x)$), but it is *not* a rigid E-unifier.

The standard methods for showing undecidability do not apply because each equation in E can be instantiated only *once*.

We conjecture that rigid E-unification is decidable. Some subcases are decidable.

SYSTEMS

Definition A *system* S is a set $\{(u_1, v_1), \ldots, (u_m, v_m)\}$ of pairs of terms.

A substitution θ is a *rigid E-unifier of S* iff θ is a rigid E-unifier of every pair (u_i, v_i).

Given a system S, a pair $(u, v) \in S$ is *solved* (in S) iff u is a *variable* and this variable occurs *nowhere else* in S.

A system S is in *solved form* iff every pair $(u_i, v_i) \in S$ is solved.

A system S in solved form defines the substitution $\sigma_S = [v_1/u_1, \ldots, v_m/u_m]$ which is a rigid E-unifier of S.

TRANSFORMATIONS ON SYSTEMS

Definition (Transformations Rules) Let E be a set of m equations, R any system (possibly empty), and u, v be two terms.

$$\{(u,u)\} \cup R \Rightarrow R \qquad (1)$$

$$\{(v,x)\} \cup R \Rightarrow \{(x,v)\} \cup R, \qquad (2)$$

where x is a variable, and $x \neq v$;

$$\{(f(u_1,\ldots,u_k), f(v_1,\ldots,v_k))\} \cup R \Rightarrow \{(u_1,v_1),\ldots,(u_k,v_k)\} \cup R \qquad (3)$$

$$\{(x,v)\} \cup R \Rightarrow \{(x,v)\} \cup R[v/x], \qquad (4)$$

where x is a variable, $x \notin Var(v)$, $x \in Var(R)$, and $R[v/x]$ is the system obtained by substituting v for all occurrences of x in R.

Note: The transformations (1) to (4) are essentially those given by Herbrand and Martelli-Montanari.

To deal with equations, we also need:

$$\{(u,v)\} \cup R \Rightarrow \{(u,l_1),(r_1,l_2),\ldots,(r_{n-1},l_n),(r_n,v)\} \cup R, \quad (5)$$

where the $(l_i \doteq r_i) \in E \cup E^{-1}$ are $n \leq m$ distinct equations, $1 \leq i \leq n$, and u, v are not variables;

$$\{(x,v)\} \cup R \Rightarrow \{(x, v[\beta \leftarrow t])\} \cup \{(v/\beta, s)\} \cup R, \qquad (6)$$

where $|v| \geq 1$, $x \in Var(v)$, β is any address in the set $\{\beta \in dom(v) \mid \exists\gamma \neq e, v(\beta\gamma) = x\}$ of proper prefixes of paths ending in a leaf labeled with the variable x, $(s \doteq t)$ is an equation in $E \cup E^{-1}$, and $v[\beta \leftarrow t]$ denotes the term obtained by replacing the subterm at address β in v with t.

Note: $\sigma(v/\beta) = \sigma(v)/\beta$, and so

$$\sigma(x) \overset{*}{\Longleftrightarrow}_{\sigma(E)} \sigma(v[\beta \leftarrow t]) = \sigma(v)[\beta \leftarrow \sigma(t)] \overset{*}{\Longleftrightarrow}_{\sigma(E)} \sigma(v)[\beta \leftarrow \sigma(s)] \overset{*}{\Longleftrightarrow}_{\sigma(E)} \sigma(v),$$

using the independent derivations

$$\sigma(x) \overset{*}{\Longleftrightarrow}_{\sigma(E)} \sigma(v[\beta \leftarrow t])$$

$$\sigma(s) \overset{*}{\Longleftrightarrow}_{\sigma(E)} \sigma(v)/\beta = \sigma(v/\beta).$$

Example: Let $E = \{(f(g(u)) \doteq h(u))\ (1), (h(v) \doteq f(v))\ (2), (f(w) \doteq w)\ (3)\}$, and $S = \{\langle g(f(x)), x\rangle\}$. The following sequence of transformations leads to a system in solved form.

$$
\begin{aligned}
\{\langle g(f(x)), x\rangle\} &\Rightarrow_2 \{\langle x, g(f(x))\rangle\}\\
&\Rightarrow_6 \{\langle x, g(h(u))\rangle, \langle f(x), f(g(u))\rangle\}\\
&\Rightarrow_3 \{\langle x, g(h(u))\rangle, \langle x, g(u)\rangle\}\\
&\Rightarrow_4 \{\langle x, g(h(u))\rangle, \langle g(h(u)), g(u)\rangle\}\\
&\Rightarrow_3 \{\langle x, g(h(u))\rangle, \langle h(u), u\rangle\}\\
&\Rightarrow_5 \{\langle x, g(h(u))\rangle, \langle h(u), h(v)\rangle, \langle f(v), f(w)\rangle, \langle w, u\rangle\}\\
&\Rightarrow_3 \{\langle x, g(h(u))\rangle, \langle u, v\rangle, \langle f(v), f(w)\rangle, \langle w, u\rangle\}\\
&\Rightarrow_2 \{\langle x, g(h(u))\rangle, \langle v, u\rangle, \langle f(v), f(w)\rangle, \langle w, u\rangle\}\\
&\Rightarrow_3 \{\langle x, g(h(u))\rangle, \langle v, u\rangle, \langle v, w\rangle, \langle w, u\rangle\}\\
&\Rightarrow_4 \{\langle x, g(h(u))\rangle, \langle v, u\rangle, \langle u, w\rangle, \langle w, u\rangle\}\\
&\Rightarrow_2 \{\langle x, g(h(u))\rangle, \langle v, u\rangle, \langle w, u\rangle\}.
\end{aligned}
$$

Hence, $[g(h(u))/x, u/v, u/w]$ is a rigid E-unifier of S.

The main difficulty to show the completeness of the transformations, is to show that if a solution exists at all, then a *small* solution also exists.

Key to the elimination of the variable x in the case of a pair (x, v), where $|v| \geq 1$ and $x \in Var(v)$.

Lemma 10 Given a set E of equations, given any term v containing some occurrence of a variable x, and such that $|v| \geq 1$, if there is a term t with no occurrence of x such that

$$v[t/x] \overset{*}{\Longleftrightarrow}_E t,$$

then there is some subterm r of t, such that,

$$r \overset{*}{\Longleftrightarrow}_E t,\ v[r/x] \overset{*}{\Longleftrightarrow}_E r,$$

and, in the sequence of rewrite steps $v[r/x] \overset{*}{\Longleftrightarrow}_E r$, for every occurrence α of the variable $x \in dom(v)$, some rewrite rule is applied to a proper ancestor β of α.

Lemma 11 (Soundness) Given a set E of equations and a system S, if $S \Rightarrow^* S'$ and S' is in solved form, then, the substitution $\sigma_{S'}$ associated with S' is a rigid E-unifier of S.

SOME DECIDABLE SUBCASES
CASE 1: REGULAR AND GROUND EQUATIONS

Definition An equation $(l \doteq r)$ is *regular* iff $Var(l) = Var(r) \neq \emptyset$.

Theorem 12 Rigid E-unification is NP-hard when E is a set of ground and regular equations and both u, v are ground.

Proof: The satisfiability problem is reduced to rigid E-unification as follows. Let the set of function symbols consist of $\wedge$, $\vee$, $\neg$, and the constants $\top$ and $\perp$. Write down the set E_{bool} of 10 ground equations corresponding to the truth tables for $\wedge$, $\vee$, $\neg$. Given any clause A, if $Var(A) = \{x_1, \ldots, x_n\}$, let

$$B_A = (x_1 \wedge x_2 \wedge \ldots \wedge x_n \wedge \perp).$$

Finally, let $E_A = E_{bool} \cup \{A \doteq B_A\}$, $u = \top$ and $v = \perp$. It is easy to see that a substitution σ such that $\top$ and $\perp$ are congruent modulo $\sigma(E_A)$ exists iff A is satisfiable, since B_A is false for every truth assignment Hence, satisfiability is reduced to rigid E-unification.

Theorem 13 Assume that the equations in E are either ground or regular, and that we consider systems such that for every pair $(u, v) \in S$, one of u, v is ground. Then, rigid E-unification is NP-complete. If S has a rigid E-unifier, there is a system S' in solved form such that, $S \Rightarrow^* S'$, for every pair $(u, v) \in S'$, v is ground, and the substitution $\sigma_{S'}$ is a rigid E-unifier of S. Furthermore, a finite complete set of rigid E-unifiers can be obtained using the transformations.

CASE 2: STRONG E-UNIFICATION

Definition Given a pair (u, v) of terms and a set E of equations, assume that any two equations in E have disjoint sets of variables, and that $Var(u \doteq v) \cap Var(E) = \emptyset$. A substitution θ is a *strong* E-unifier of u and v iff there is a sequence of rewrite steps $\theta(u) \overset{*}{\Longleftrightarrow}_{\theta(E)} \theta(v)$, such that, every nonground equation $\theta(s \doteq t)$ is used at most once.

The method of matings is complete using strong E-unifiers.

Lemma 14 If the amplification D of a universal sentence A in nnf has a mating found using rigid E-unification, then there is some (further) amplification D' of A which has a mating found using strong E-unification.

Theorem 15 Strong E-unification is NP-complete. If S has a strong E-unifier, there is a system S' in solved form such that, $S \Rightarrow^* S'$, and the substitution $\sigma_{S'}$ is a strong E-unifier of S. Furthermore, a finite complete set of strong E-unifiers can be obtained using the transformations.

FURTHER WORK

Very recently, in collaboration with P. Narendran, we believe that we have come very close to showing that rigid E-unification is indeed decidable. The techniques involved use a type of Knuth-Bendix completion procedure, and Kruskal's tree theorem.

Find other decidable subcases and pin down their complexity.

Higher-order case?

Solving Disequations

Claude KIRCHNER

Pierre LESCANNE

Abstract

We present a general study of equations (objects of form s = t and disequations (objects of form s ≠ t)) solving. The problem is approached from its fully general mathematical definition clearly separating universally and existentially quantified variables. In addition it is showed to have many connections with unification in equational theories like associativity commutativity, in particular methods similar to those used to solve equational unification problem works in solving disequations. This abstract framework is then applied to study the sufficient completeness of a rewrite rule based definition of a function.

EQUATIONAL PROBLEMS

This a problem of the form

$$\exists (y_1,y_2,\dots,y_n)\forall(x_1,x_2,\dots,x_m)P(y_1,y_2,\dots,y_n,x_1,x_2,\dots,x_m)$$

the y_j are **Existential** variables and the x_i are **Universal** variables and $P = S_1 \vee \dots \vee S_n$ is a disjunction of *systems* where a system S_i is a conjunction of *equations* $s=t$ and *disequations* $s\neq t$.

A **Restricted Unification** problem (Bürckert)

$$(x_1,x_2,\dots,x_m)\ \forall\ \exists\ (y_1, y_2, \dots, y_n)$$
$$t(y_1,y_2,\dots,y_n,x_1,x_2,\dots,x_m)= t'(y_1,y_2,\dots,y_n,x_1,x_2,\dots,x_m)$$

is an equational problem.

A **solution** of an equational problem $P= S_1 \vee \dots \vee S_n$ is a substitution σ such that $D(\sigma)\subseteq \{y_1,y_2,\dots,y_n\}$ and there exists a system S_i such that for all equation $s=t$ in S_i, $\sigma(s) = \sigma(t)$ and for all disequation $s\neq t$ in S_i $\sigma(s) \neq \sigma(t)$.

Sufficient Completeness
and
Inductive reducibility

Definition: A term *t* is inductively reducible w.r.t. a Term Rewriting System R if each instance of *t* is reducible w.r.t. R.

The **sufficient completeness** of *f* w.r.t. R is the inductive reducibility of $f(x_1,\ldots,x_n)$.

$$\forall y_{1,1}\in T(C)\ldots\forall y_{j,h}\in T(C)\ \ldots\exists x_{1,1}\in T(C)\ldots\exists x_{i,k}\in T(C)\ldots$$
$$\bigvee_{s\in Sub(t)} \bigvee_{l\rightarrow g\in R}\ s(\underline{y_j}) = l(\underline{x_i})$$

$Sub(t)$ is the set of subterms of *t*.

Indeed, the **reducibility** means there exists a non variable subterm *(disjunction on the subterms)* that matches *(existence of a values $x_{i,k}\in T(C)$)* the left-hand-side of a rule *(disjunction on the rules).*

The inductive reducibility means that this has to be satisfied for each ground instance *(universal quantification over $y_{j,h}\in T(C)$).* This is not a unification problem because of the quantifiers. In a unification the existential quantifier ∃ is in first position.

RULES

Basic rules

$\xi\neq\xi \mapsto$ *false*

$\xi=\xi \mapsto$ *true*

Anteposition

$s=\xi \mapsto \xi=s$ if s is not a variable

$s\neq\xi \mapsto \xi\neq s$ if s is not a variable

Quantifier rules

$\forall x(P \vee Q) \mapsto (\forall xP)\vee Q$

provided Q does not contain x

$\forall x(P \wedge Q) \mapsto (\forall xP)\wedge (\forall xQ)$

Occur checks

Definition: $\xi=t <_S \xi'=t'$ if and only if $\xi\in Var(t')$, $\xi\neq t'$, $(\xi=t)\in S$ and $(\xi'=t')\in S$.

$S \mapsto$ *false* if $S\ni(\xi=s)$ s.t. $(\xi=s) <^+_S (\xi=s)$

$S \mapsto S$ if $\xi\in Var(s)$ and $\xi\neq s$ or there exists

$(\xi=s) \overset{*}{<}_S (\xi''=s'')$, $\xi''\in Var(s)$ and $\xi\in Var(s')$

Clashes and decompositions

$f(u_1,u_2,\ldots,u_n) \neq g(v_1,v_2,\ldots,v_m) \mapsto true$

$f(u_1,u_2,\ldots,u_n) = g(v_1,v_2,\ldots,v_m) \mapsto false$

$f(u_1,u_2,\ldots,u_m) \neq \cancel{g}f(v_1,v_2,\ldots,v_m) \mapsto \bigvee_{1 \le i \le m}(u_i \neq v_i)$

$f(u_1,u_2,\ldots,u_m) = \cancel{g}f(v_1,v_2,\ldots,v_m) \mapsto \bigwedge_{1 \le i \le m}(u_i = v_i)$

Sliding

$\xi = s \wedge \xi \neq t \mapsto \xi = s \wedge s \neq t$ *if* s *and* t *are not* ξ

$\xi = s \wedge \xi = t \mapsto \xi = s \wedge s = t$ *if* s *and* t *are not* ξ

Boolean Rules

$p \wedge false \mapsto p$

$p \wedge p \mapsto p$

etc...

Substitution

$\forall x(P \wedge \{x \neq s\}) \vee Q) \mapsto (\forall x[(P \wedge Q(s)) \vee (Q(x) \wedge Q(s))]$

if $x \notin Var(s)$

Specific instances of this rule are .

$\forall x(\{x \neq s\} \vee Q) \mapsto Q(s)$

$\forall x\{x \neq s\} \mapsto false$

Problems in Normal Form

A problem is said to be in **normal form**, if all the system\it contains are made of disequations of the form $y \neq s$ and of equations of the form $y=s$.

In addition, if there is an equation of the form $y=s$, there is no equation of the form $y=t$ or disequation of the form $y \neq u$.

Theorem: Given a problem P, there exists always an equational problem P' in normal form such that

$$P \longmapsto^* P'$$

Theorem: If P contains only equations, then P' contains only equations, therefore P' determines a family of substitutions.

If ~~If~~ P contains only disequations, then P' contains only disequations.

Getting ride of disequalities

Now the quantifications are done on *T(C)* **instead of** *any algebra.*

$$\exists y_1' \ldots \exists y_m' \forall x_1' \ldots \forall x_n'$$
$$[P \vee \forall x_{1,1} \in T(C) \ldots \forall x_{i,k} \in T(C)\ (Q \quad \wedge \bigwedge_{i \in I} y \neq f_i(\underline{x}_i))]$$
$$\mapsto$$
$$\exists y_1' \ldots \exists y_m' \exists y_{1,1} \ldots \exists y_{j,h} \ldots \forall x_1' \ldots \forall x_n'$$
$$[P \vee (Q \quad \wedge \bigvee_{j \notin C-I} y = f_j(\underline{y}_j)\,)]$$

$I \subseteq C$

[illegible] x_i [illegible] apparaissent pas dans P ni [illegible]

Sufficient Completeness and Inductive reducibility

ground

Definition: A term *t* is inductively reducible w.r.t. a Term Rewriting System R if each instance of *t* is reducible w.r.t. R.

The **sufficient completeness** of *f* w.r.t. R is the inductive reducibility of *f(x₁,...,xₙ)*.

$$\forall y_{1,1} \in T(C) \ldots \forall y_{j,h} \in T(C) \ldots \exists x_{1,1} \in T(C) \ldots \exists x_{i,k} \in T(C) \ldots$$
$$\bigvee_{s \in Sub(t)} \bigvee_{l \to g \in R} s(\underline{y}_j) = l(\underline{x}_i)$$

Sub(t) is the set of subterms of *t*.

Indeed, the **reducibility** means there exists a non variable subterm *(disjunction on the subterms)* that matches *(existence of a values $x_{i,k} \in T(C)$)* the left-hand-side of a rule *(disjunction on the rules)*.

The inductive reducibility means that this has to be satisfied for each ground instance *(universal quantification over $y_{j,h} \in T(C)$)*. This is not a unification problem because of the quantifiers. In a unification the existential quantifier *∃* is in first position.

Its **negation** is,

$$\exists y_{1,1}\in T(C)...\exists y_{j,h}\in T(C)\ ...\ \forall x_{1,1}\in T(C)...\ \forall x_{i,k}\in T(C)...$$
$$\bigwedge_{s\in Sub(t)} \bigwedge_{l\rightarrow g\in R}\ s(\underline{y}_j)\neq l(\underline{x}_i)$$

This now has a **flavor of unification**; **since the** existential quantifier is in first position.

This is an equational problem.

·HOW TO REDUCE DISEQUATIONS

H. Comon
LIFIA BP 68,
38402 Saint Martin D'Hères cedex
France

Abstract

Let $T_\Sigma(X)$ be the algebra defined in the usual way, Σ being a finite set of functional symbols together with a typing function and X an enumerable set of variables. Let A be a subset of X. We say that a substitution σ is an A-solution of the disequation $t \# t'$ iff

(i) for every $x \in X\text{-}A$, $\sigma(x) = x$

(ii) $\sigma(t)$ and $\sigma(t')$ are not unifiable

This may be viewed as an universal quantification of the variables of t and t' which do not belong to A. Note that t and t' may share variables. $\sigma(X)$ may also share variables with t and t'; i.e. σ may be not idempotent. Finally, we are interested in the solutions of such disequations in $T_\Sigma(X)$ and not only in the ground solutions.

In order to simplify such disequations some problems arise. For example <x,x> # <y,z> where x,y,z are variables, is not equivalent to x # y or x # z or y # z, even if we restrict ourself to substituitons σ such that every term in $\sigma(X)$ is linear.

Indeed, assuming that there exists three functional symbols: 0 (0-ary), s (unary) and f (ternary), the substitution x = f(x1,x2,x3), y = f(x2,x3,x1), z = f(g(x3),x4,x5) is a solution of the above disequation and is neither a solution of x#y nor of x#z nor of y#z.

Thus we look only at what we call *A-linear solutions*. Such a substitution transforms every linear term of $T_\Sigma(A)$ into a linear term. Ground substitutions are particular cases of the latter.

We show now how to simplify such disequations *as far as possible* . More precisely, it is possible to show that a single disequation can be *reduced* to equations and at most one disequation between two variables. We cannot expect more since the X-solutions of a disequation between two variables are given by all the non-unifiable pairs of terms.

Finally, a comparison with related work will be given;

Refrences

A. Colmerauer *PROLOG II, Manuel de référence et modèle théotique.* GIA, Luminy

H. Comon *Sufficient Completeness, Term Rewriting Systems and Anti-Unification* Proc. CADE 8

H. Comon *About Inequations Simplification* LIFIA, Grenoble, 1987.

C. Kirchner , P. Lescanne *Solving Disequations* CRIN, Nancy, 1986.

JL. Lassez, M. Maher, K. Mariott *Unification Revisited* IBM, Yorktown Heights, 1986.

HOW TO REDUCE DISEQUATIONS

H. Comon, LIFIA
comon@lifia.uucp

UNIFICATION

and SOLVING INEQUATIONS

(A. Colmerauer)

Some solutions of $f(x,x) \# f(y,z)$ in $T_\Sigma(X)$ where $\Sigma = \{0, s, f\}$:

① $x=0, \; y=0, \; z=s(0)$

② $x=0, \; y=s(0), \; z=s(0)$

③ $x=0, \; y=s(0)$

④ $y=0, \; z=s(0)$

⑤ $x=s(x'), \; y=f(y',z')$

⑥ $y=s(z)$

• • •

MATCHING

and "ANTI-MATCHING"

(H. Comon)

$f(x,x) \# f(y,z)$: some solutions

① $x=0 \; \cancel{, y=0} \; , z=s(0)$

② $x=0, \; \cancel{y=s(0)}, \; z=s(0)$

③ $x=0 \; \cancel{, y=s(0)}$

④ $y=0, \; z=s(0)$ ok

⑤ $x=s(\cancel{x'}), \; y=f(y',z')$

⑥ $y=s(z)$ ok

• • •

Only variables of the right hand side may be instantiated

USE OF ANTI-MATCHING FOR THE COMPLETENESS OF DEFINITION

Solve
$$\begin{cases} l_1 \# f(x_1, \dots, x_n) \\ \vdots \\ l_m \# f(x_1, \dots, x_n) \end{cases}$$

The solutions are the terms having f as their root and which are not "covered" by the left hand sides

RESTRICTED-UNIFICATION and DISUNIFICATION

(Introduced by P. Lescanne).

Some solutions of $f(x,x) \# f(y,z)$ with the restrictions that $Dom(\sigma) \subseteq \{x, y\}$

① $x=0, \cancel{y=0}, z=s(0)$

③ $x=0, y=s(0)$ OK

④ $y=0, \cancel{z}=s(0)$

⑤ $x=s(x'), y=f(y',z')$ OK

⑥ $y=s(z)$ OK

...

The "universally" quantified variables are precised.

BASIC DISUNIFICATION ALGORITHM

RULE 5: ELIMINATION OF UNIVERSALLY QUANTIFIED VARIABLES.

Let x be universally quantified, then

σ is a solution of

$$< x_1, \dots, x_i, x, x_{i+1}, \dots, x_n > \# < v_1, \dots, v_i, v, v_{i+1} \dots v_n >$$

iff

σ is a solution of

$$\theta < x_1, \dots, x_i, x_{i+1}, \dots x_n > \# \theta < v_1, \dots v_i, v_{i+1}, \dots v_n >$$

where θ is the substitution

$(x \leftarrow v)$

REPRESENTATION OF THE SOLUTIONS: SOME PROBLEMS.

example: $x \# y$

1) Infinite set of solutions

   $\{ x = 0, y = s(0)$; $x = s(0)$; $y = 0$; $x = s^k(0)$ & $y = s^l(0)$ with $k \neq l$

   ---

2) Infinite set of "minimal" solutions

   $x = s^n(0)$; $y = s^{n+1}(3)$

3) Infinite set of "cyclic" and minimal solutions

   $x = s^n(y)$

REPRESENTATION OF THE SOLUTIONS:

A FIRST APPROACH

TO RESTRICT THE PROBLEM TO:

FIND A REDUCED FORM WHICH INSURES THAT THERE EXISTS AT LEAST A (GROUND) SOLUTION.

for the ground solutions and without universally quantified variables:

INDEPENDENCE OF INEQUATIONS (Lassez, Maher, Mariott)

$E \& I_1 \dots \& I_n$ has at least a solution

iff

$E \& I_j$ has at least a solution, for every j.

REDUCTION OF DISEQUATIONS A SECOND APPROACH

Theorem

A disequation with existentially en universally quantified vari in the initial algebra is equivalent to a finite disjunctions of sets (E_i, c_i) where E_i is a set of equations and $c_i = \phi$ or is a disequation between two variables.

In other words: A disequation may be reduced in a disequation between two variables.

BASIC DISUNIFICATION ALGORITHM

RULE 12

$u_1, \dots, u_n$ are variables

$v_1, \dots, v_n$ are linear terms

$\forall j \geq i, \quad u_i \in V(v_j)$

σ is a (...) solution of $\langle u_1, \dots, u_n \rangle \# \langle v_1, \dots, v_n \rangle$

iff

σ is a (...) solution of one of the following:

(1) $u_i \# v_i \quad , \quad 1 \leq i \leq n$

(2) $v_i \# v_j \quad , \quad 1 \leq i,j \leq n$ and $u_i = u_j$

(3) $\langle u_i, u_j \rangle \# \langle v_i, v_j \rangle \quad 1 \leq i < j \leq n$ and

$$\begin{cases} V(v_i) \cap V(v_j) \neq \phi \\ \text{or} \quad u_j \in V(v_i) \end{cases}$$

NOT ONLY GROUND SOLUTIONS

$\langle x, x \rangle \# \langle y, z \rangle$ is not equivalent to

$x \# y$ or $y \# z$ or $z \# x$

(in general)

example:

$x = f(x_1, x_2, x_3)$; $y = f(x_2, x_3, x_4)$; $z = f(g(x_3), x_4, x_5)$

is a solution of $\langle x, x \rangle \# \langle y, z \rangle$

which is not a solution of any $x \# y, y \# z, z \# x$

NOT ONLY GROUND SOLUTIONS :

V- LINEAR SOLUTIONS

V is a finite set of variables • - A Substitution σ is V- linear iff

- $\forall x \in V$, $\sigma(x)$ is linear
- $\forall x, y \in V$, $\mathcal{V}(\sigma(x)) \cap \mathcal{V}(\sigma(y)) = \phi$

Rules (formalism of C. Kirchner & P. Lescanne)

$$\frac{\langle x, \dots, x \rangle \# \langle y_1, \dots, y_n \rangle}{(\bigvee_i x \# y_i) \vee (\bigvee_{i,j} y_i \# y_j)}$$

($x, y_1, \dots y_n$ are variables)

GENERALISATION TO EQUATIONAL THEORIES

- The Independence of inequations holds when there is no universelly quantified variables (?)
- In order to "solve" $t \# t'$, Solve first $t = t'$. Then replace the equalities by $\#$.

ABSTRACT

INDUCTIVE REDUCIBILITY PROBLEMS
AND
SOLVING INEQUATIONS

JOINT WORK WITH H. COMON AND J. HSIANG

We will show how to reduce various inductive reducibility problems (reducibility of all ground instances of a term) to the existence of solutions for a given set of equations and inequations. The construction of the set depends on the rewriting resation used : standard rewriting, rewriting modulo, and extented rewriting will be considered. Resolution of the obtained set must accomodate non free symbols, AC-symbols as well as "non free inequalities", e-q, the recursive path ordering.

Inductive Reducibility Problems

and

solving

$$\left\{\begin{array}{l}\text{Equations} \quad + \quad \text{Disequations} \\ \quad + \qquad\qquad\qquad\quad + \\ \text{Inequations} \quad + \quad \text{Disinequation}\end{array}\right.$$

Hubert Comon , LIFIA , Grenoble

Jieh Hsiang , SUNY , Stony brook

Jean-Pierre Jouannaud , LRI, Orsay

Inductive Completion described by Inference Rules:

Input: R_0 : a church-Rosser set of rules
E_0 : a set of equations to be proved

Adding critical pairs: $E, R \vdash E \cup \{s=t\}, R$ if $(s,t) \in CP(R)$

Simplifying equations:

$E \cup \{s=t\}, R \vdash E \cup \{s'=t\}, R$ if $s \xrightarrow[R]{} s'$

$E \cup \{s=t\}, R \vdash E \cup \{s=t'\}, R$ if $t \xrightarrow[R]{} t'$

Eliminating trivial equations:

$E \cup \{s=s\}, R \vdash E, R$

Simplifying rules:

$E, R \cup \{s \to t\} \vdash E, R \cup \{s \to t'\}$ if $t \xrightarrow[R]{} t'$

$E, R \cup \{s \to t\} \vdash E \cup \{s'=t\}, R$ if $s \xrightarrow[l \to r \in R]{} s'$ and $s \neq l$

orienting equations into rules:

$E \cup \{s=t\}, R \vdash E, R \cup \{s \to t\}$ if $s > t$ and
s inductively reducible by R_0

$E \cup \{s=t\}, R \vdash E, R \cup \{t \to s\}$ if $t > s$ and
t inductively reducible by R_0

is proving equations:

$\cup \{s=t\}, R \vdash$ disproof if $s > t$ and
s not inductively reducible by R_0

$E \cup \{s=t\}, R \vdash$ disproof if $t > s$ and
t not inductively reducible by R_0

Inductive Reducibility:

t is inductively reducible by R iff every ground instance of t is reducible by R

Example: 0 (zero) and S (successor)

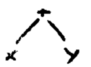
$R_0 = \{ S\,S\,0 \to 0 \}$

$S\,S\,x$ is inductively reducible

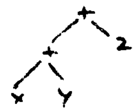
$S\,x$ is not inductively reducible

Example: 0, S, +

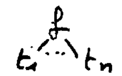
$$R_0 : \begin{cases} 0 + x \to x \\ S(x) + y \to S(x+y) \end{cases}$$



is inductively reducible



is inductively reducible

Remark: If f is completely defined with respect to a set of constructors, then  is inductively reducible

Theorem: Inductive reducibility is decidable

Inductive Reducibility and Solving Disequations

basic idea: search for a counter-example, i.e.,:

t is NOT inductively reducible by R iff there exists an irreducible ground instance $t\gamma$ of t, i.e.:

there exists an assignement γ of variables of t by irreducible ground terms s.t.:

$$(t/_p)\gamma \neq l\sigma \quad \forall l \to r \in R,\ \forall p \in \bar{D}(t),\ \forall \sigma \in \Sigma.$$

Important: $\neq$ is the synctactic difference
variables of t are existentially quantified
variables of l are universally quantified

Definition: The disequation $t \neq l$ is solvable iff there exists θ such that $t\theta$ and l do not unify: Disunification

Note: we could say $t\theta$ and $l\theta$ do not unify with $D(\theta) \subseteq$ set of existential variables.

Inductive Reducibility:

t is NOT inductively reducible by R iff the set of disequations

$$\{\ t/_p \neq l \mid \forall p \in \bar{D}(t),\ \forall l \to r \in R\}$$

has irreducible ground solutions

Description of Normal Forms: [redacted]

[Comon & Remy].

Example: $0, S, \quad SS0 \to 0$

$NF = NF_0 \cup NF_S$

$NF_0 = \{0\}$

$NF_S = \{ Sx \mid Sx \neq SS0 \text{ and } x \in NF\}$

Example: $0, S, + \quad \begin{cases} 0 + v \to v \\ S(u) + v \to S(u+v) \end{cases}$

$NF = NF_0 \cup NF_S \cup NF_+$

$NF_0 = \{0\}$

$NF_S = \{ Sx \mid x \in NF\}$

$NF_+ = \{ x + y \mid x + y \neq 0 + v \text{ and } x + y \neq S(u) + v$
$\text{and } x \in NF \text{ and } y \in NF\}$

Theorem [Comon]:

emptyness, finiteness of such descriptions is decidable.

inductive Reducibility can be stated as:

$$\{\ t/_p \neq l \mid \forall p \in \bar{D}(t),\ \forall l \to r \in R\}$$
$$\wedge \{x \in NF \mid \forall x \in V(t)\}$$

How to solve disequations (and equations) (bis)

-
- by using inference rules to transform a unification/disunification problem into a simpler one.
- Key remark: the ability to decompose until <u>variables are obtained</u> permits to extract the
  - mgu.
  - or
  - mgdus

Inductive completion modulo equations ($\mathcal{E}$):

- Reduction uses $\mathcal{E}$-pattern matching
- CP-computation uses $\mathcal{E}$-unification
- Inductive reducibility uses $\mathcal{E}$-pattern matching

$$\rightsquigarrow$$

$\mathcal{E}$ must be built into disequation solving,
<u>$\mathcal{E}$-disunification</u>

- Proving equations like commutativity to be inductive consequences of the axioms needs a more sophisticated notion: Inductive co-reducibility. This also gives rise to a disunification problem or an $\mathcal{E}$-disunification problem.

UnFailing Completion [Hsiang - Rusinowitch]

ideas:
- avoid failure of completion by using rules and equations for reduction.
- reduction relation with R and E:

  $t \xrightarrow[R]{} s$ as usual

  $t \xrightarrow[l=r\in E]{} s$ iff:

  (1) $\exists p \in \bar{D}(t)$, $\exists \sigma \in \Sigma$, $t/_p = l\sigma$

  (2) $s = t[p \leftarrow r\sigma]$

  (3) $l\sigma > r\sigma$

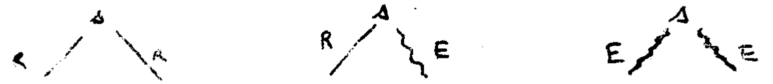
Note: depending on the result of (3) $l=r$ or $r=l$ is used

- use a reduction ordering $>$ <u>total</u> on ground terms.

Example: $x+y = y+x$ $>$ is RPO with precedence $a<b<+$ + lexico

$b+a \longrightarrow a+b$

- Critical pairs:



Inductive Unfailing Completion:

- inductive reducibility:

same definition, with respect to the new notion of reduction

- Converting into a disunification problem:

$NF_f = \{ f(x_1, \ldots, x_n) \mid x_1, \ldots, x_n \in NF$

$\bigwedge \{ f(x_1, \ldots, x_n) \# l \mid \forall l \to r \in R \}$

$\bigwedge \{ [(f(x_1, \ldots, x_n) \# g) \vee (f(x_1, \ldots, x_n) = g \wedge g \not> d) \mid \forall g=d \in E$

t is NOT inductively reducible iff

$\{ t/_p \# l \mid \forall p \in \bar{D}(t), \forall l \to r \in R \}$

$\wedge$

$\{(t/_p \# g) \vee (t/_p = g \wedge g \not> d) \mid \forall g=d \in E\}$

$\wedge$

$\{x \in NF \mid \forall x \in V(t)\}$

has a solution

- **what candidate for $>$?** ☺
  - $>_{rpo}$ because

    $>_{rpo}$ (and $\not>_{rpo}$) have good decomposition properties that allow to <u>reach variables</u>:

- $\underset{t_1 \cdots t_n}{f} > \underset{s_1 \cdots s_n}{f} \vdash \{t_1 \ldots t_n\} >_{mul} \{s_1 \ldots s_n\}$
- $\underset{t_1 \cdots t_n}{f} > \underset{s_1 \cdots s_m}{g} \vdash \exists j \,,\ t_j > \underset{s_1 \cdots s_m}{g}$, if $f <_{\mathcal{F}} g$
- $\underset{t_1 \cdots t_n}{f} > \underset{s_1 \cdots s_m}{g} \vdash \forall j \,,\ \underset{t_1 \cdots t_n}{f} > s_j$ if $f >_{\mathcal{F}} g$

- **Conjecture:**
  - Comon's results extend to this case

    $\Rightarrow$ inductive unfailing completion is <u>complete</u>,

    i.e., if $s = t \notin \mathrm{Ind}(R,E)$

    then IUC will return disproof

    in finite time.

- **back to the table:**

  $=$ equation

  $\neq$ disequation

  $\sim$ inequation

Some problems with unification on a lattice of types (as used in ERIL)

A. J. J. Dick

Informatics Division
Rutherford Appleton Laboratory
Chilton, Didcot, OXON OX11 0QX
U. K.

ABSTRACT

The term rewriting system ERIL (Equational Reasoning: an Interactive Laboratory) permits a form of order-sorted algebra in which the undefined (or absurd) sort is included as a bottom element, thus creating a lattice of sorts [CUN85, DIC85] . Each function signature is overloaded to allow its sort to vary according to the sorts of its arguments. The unification algorithm is modified to be sort-preserving. Implicit in this scheme, is that every well-formed term, regardless of its static sort, may actually be of a lower sort, perhaps even undefined. As a term is rewritten, therefore, it may be reduced to something of undefined sort.

Whilst operationally ERIL seems to be sensible in its behaviour, the author has experienced considerable difficulty in finding a sound theoretical model of the method. The problem lies in the nature of the underlying equational logic, in which the satisfaction relation is not transitive.

CUN85. R. J. Cunningham and A. J. J. Dick, "Rewrite Systems on a Lattice of Types," *Acta Informatica* **22,** pp. 149-169 (1985).

DIC85. A. J. J. Dick, "ERIL - Equational Reasoning: an Interactive Laboratory," Rutherford Appleton Laboratory, Report RAL-86-010 (Mar 1985).

COERCION RULE 1

$$\frac{E \cup \{x:s = op(t_1 \cdots t_n):s'\} \quad \text{where } s' \not\leq s}{E \cup \left\{\begin{array}{l} x:s = op(x_1:s_1 \cdots x_n:s_n) \\ x_1:s_1 = t_1 \\ \quad\vdots \\ x_n:s_n = t_n \end{array}\right\} \quad \begin{array}{l}\text{where} \\ op: s_1 \cdots s_n \to s'' \\ \text{and } s'' \leq s\end{array}}$$

COERCION RULE 2

$$\frac{E \cup \{x:s = y:s'\} \quad \text{where } s' \not\leq s}{E \cup \{x:s = y':s \sqcap s'\}}$$

sorts $A < B$

ops
$f : B \to B$
$f : A \to A$
$g : B \to B$
$g : A \to A$

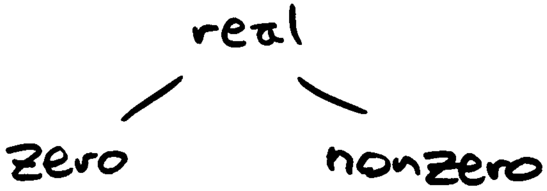
$$\{\, x:B = f(g(y:A)) \,\}$$

$$\left\{\begin{array}{l} x:B = f(x_1:B) \\ x_1:B = g(y:A) \end{array}\right\}$$

$$\left\{\begin{array}{l} x:B = f(x_1:B) \\ x_1:B = g(x_2:B) \\ x_2:B = y:A \end{array}\right\}$$

$$\left\{\begin{array}{l} x:B = f(x_1:B) \\ x_1:B = g(x_2:B) \\ y:A = x_2:B \end{array}\right\}$$

$$\left\{\begin{array}{l} x:B = f(g(x_2:B)) \\ x_1:B = g(x_2:B) \\ y:A = x_2:B \end{array}\right\}$$



D: zero (singleton)

COERCION RULE 3

$$\frac{E \cup \{\, x:s = y:s' \,\}}{E \cup \{\, x:s = c \,\}}$$

where $s' < s$ and s' sing

where c is of type s'

ORDER SORTED EQUATIONAL UNIFICATION

Claude Kirchner
CRIN INRIA
BP 239
54506 Vandoeuvre Les Nancy Cedex
France (E-mail: inria!crin!ckirchne)

Abstract

Order sorted unification is studied from an algebraic point of view. We show how order sorted equational unification algorithms can be built when the equational theory A is sort preserving, that is such that any A-equal terms have the same lowest sort. Under this condition the results obtained in the unsorted framework extend without major difference to the order sorted one. This concern in particular combination of unification algorithms. An important application is order sorted associative commutative unification for which no direct algorithm was given until now.

This is a preliminary announcement of results; much of it is work in progress.

ORDER SORTED EQUATIONAL
UNIFICATION

Claude Kirchner

CRIN
INRIA - Lorraine

UNIFORM CONCEPTION OF UNIFICATION PROCEDURE

UNIFICATION ALGORITHM
=
TRANSFORMATION ALGORITHM
(from any kind of equation
to a set of equations of the form x == t)
+
RESOLUTION OF SYSTEMS OF
EQUATIONS OF THE FORM
x == t

The first goal: SIMPLIFY

⟶ 3 transformations:

- DECOMPOSITION

(simplify without taking into account the axioms)

- MERGING

(merge the constrains)

- MUTATION

(simplify with respect to the axioms)

THE COMPLETE THEORIES

- theories for which the resolution of equation of the form $x == t$ is decidable.

THE STRONGLY COMPLETE THEORIES

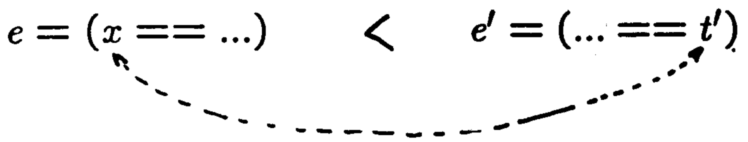
- theories for which any equation $x == t$ has an CSU which elements σ are all such that $D(\sigma) = \{x\}$

Example: AC, C, minus:
$((-(-x)) = x \; and \; -(x+y) = (-y)+(-x))$

but for example if $A = \{a+b=a\}$ then the equation
$x+y == x$ has A-solution $\{(x \leftarrow a), (y \leftarrow b)\}$.

THE STRICT THEORIES

- $e < e'$ iff there exists a variable x in $V(e)$ and a term t' in e' such that $x \in Var(t')$.



$e = (x == ...) \quad < \quad e' = (... == t')$

- A is strict iff
(S has A-solutions) $\Rightarrow$ ($\overset{+}{<}$ is a strict order on S)

- permutative $\Rightarrow$ strict

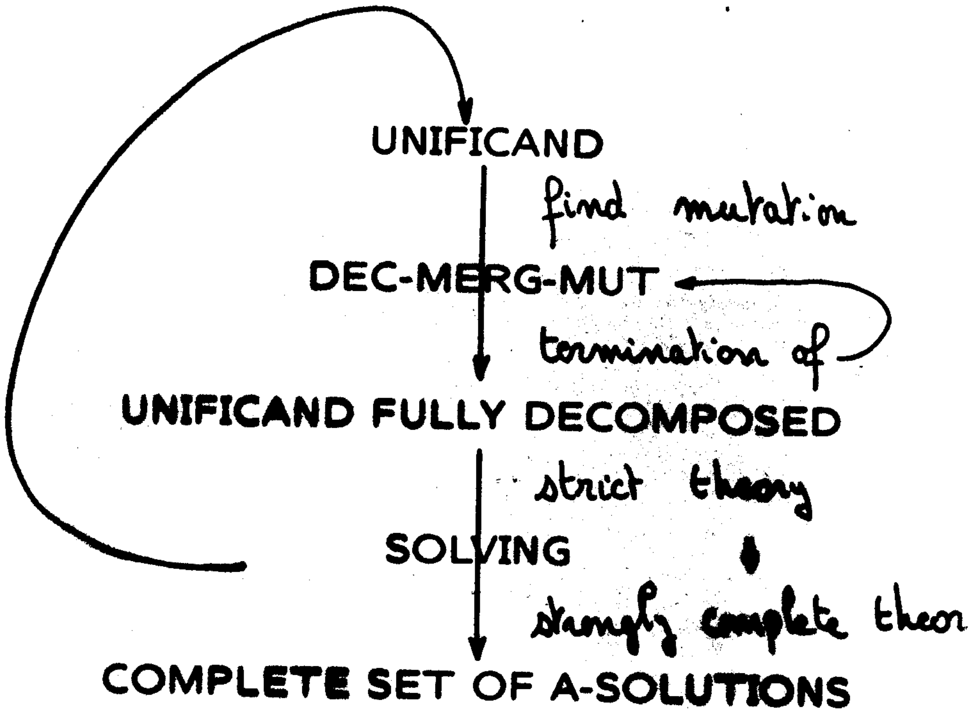
- For $A = \{x * 0 = 0\}$
$e = (z == y * z)$ has for A-solution $(z \leftarrow 0)$ and $e < e$

RESOLUTION OF FULLY DECOMPOSED UNIFICANDS IN STRICT AND STRONGLY COMPLETE THEORIES

Let S a system of multiequations,

IF one can sort in decreasing order
with respect to $<$ the elements
of S (let $Q = \{e_1, e_2, ..., e_n\}$ the result)
THEN the set of all the $\sigma = \sigma_n ... \sigma_2 . \sigma_1$
with $\sigma_i \in CSU(e_i, A)$ is a CSU of S,
ELSE SU(S,A) $= \emptyset$

ALTOGETHER



UNIFICAND

DEC-MERG-MUT

UNIFICAND FULLY DECOMPOSED

SOLVING

COMPLETE SET OF A-SOLUTIONS

Extention to the order sorted framework

=-décomposition $$\frac{f(t_1, -, t_n) == f(t'_1, -, t'_n)}{\bigwedge_i t_i == t'_i} \quad \text{if } f \in F_d^A$$

=-clash $$\frac{f(t_1, -, t_n) == g(t'_1, -, t'_p)}{\text{failure}} \quad \text{if } f \neq g \text{ and } f, g \in F_d^A$$

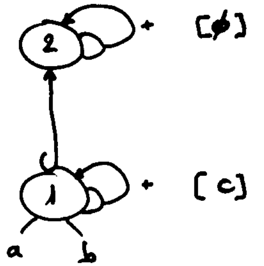
Merging $$\frac{x == t \wedge x == t'}{x = t == t'}$$

Mutation

ⓐ With no overloading with ≠ properties

same as in the monosorted case

ⓑ else



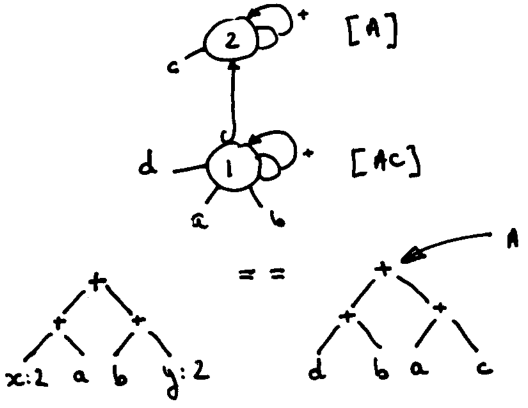
$x:2 + a == y:2 + b$

→ try both possibilities

but even if the theories is sort preserving

$$\left(t =_A t' \Rightarrow l_s(t) = l_s(t')\right)$$

it can be worst :



→ the single sorted case can not be applied

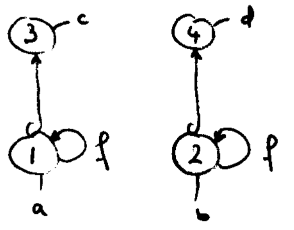
•→ The mutation transformation is specific to the order sorted framework when there is overloading with different properties

How to solve elementary équations ?

- In the ϕ theory

  equations $x == t$ are no more unitary in general

- In non empty theories



$(a = b) = A$

$e ::= (x:s == y:s')$

$SU(e, A) = \{ x \leftarrow f^n(a),\ y \leftarrow f^n(b) \mid n \geq 0 \}$

but : If the theory is sort preserving then equation

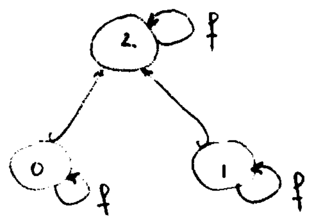
$x == t$ can be solved as in the empty theory

How to solve fully decomposed systems ?

Ⓐ Strictness is always useful

(S has A-solutions $\Rightarrow$ S has no cycle)

Ⓑ But strict theories are no more strongly complete in general



$$x':0 == f(y:2) \dashrightarrow \begin{cases} x \leftarrow f(z':0) \\ y \leftarrow z':0 \end{cases}$$

$$x:1 == f(y:2) \longrightarrow \begin{cases} y \leftarrow z:1 \\ x \leftarrow f(z:1) \end{cases} *$$

→ one needs to iterate full decomposition and solving phases.

ABSTRACT

UNIFICATION IN AN ORDER-SORTED CALCI II US WITH DECLARATIONS

This talk presents unification in order-sorted term algebras (with no additional axioms) where the syntactical sort of a term is defined not by function declarations (i.e., declarations of the type f: S1xS2x...xSn →Sn+1) but also by explicit term declarations (i.e., declarations of the form t:s which means term t has sort S). This extension is equivalent to unconditioned sort-constraints of Goguen & Meseguer, but preserves computability of the sort of a term. In order-sorted term algebras with finitely many term declarations a minimal set of unifiers exists, is recursively enumerable but may be infinite, but unifiability is undecidable in general.

We exhibit some subcases of linear signatures (i.e. in every term declaration t:S the term t is linear) that have a decidable unification problem.

Unification in an Order-sorted Calculus with Declarations

Goguen:	declarations Sort constraints
Wadge:	Classified algebras

function declaration:

$f: S_1 x \ldots x S_n \rightarrow S \quad\quad \cong \quad f(x_{S_1}, \ldots, x_{S_n}):S$

term declaration: t:S

<u>Example:</u> Specification of even numbers:

EVEN ⊏ NAT
0:EVEN
s:NAT → NAT
$s(s(x_{EVEN}))$: EVEN

even ground terms: 0, s(s(0)), $s^4(0)$,...

The sort of a term is defined recursively:

~~$t{:}S \Rightarrow S \in S_\Sigma(t)$~~

~~$S \in S_\Sigma(t),\ S(x) \in S_\Sigma(s) \Rightarrow S \in S_\Sigma(\{x \leftarrow s\}t)$~~

For finite signatures:
The sort of a term is decidable and computable in linear time.

well-sorted terms $\mathbf{T}_\Sigma \cong \{t \mid S_\Sigma(t) \neq \emptyset\}$

well-sorted substitutions σ:
$S(x) \in S_\Sigma(\sigma x)$ for all x.

Requirements:
$\mathbf{T}_\Sigma$ subterm-closed.
$\mathbf{T}_\Sigma$ regular, i.e. every term has a minimal sort

$\mathbf{T}_\Sigma$ is a free algebra.

$\mathbf{T}_{\Sigma,gr}$ is the initial algebra.

Construction of well-sorted terms

* $t : S$ term declaration $\Longrightarrow$ $t \vdots S$

* $x : S$ $\Longrightarrow$ $x \vdots S$

* $t \vdots S \wedge S \sqsubseteq R \Longrightarrow t \vdots R$

* $t \vdots S,\ x \in V(t),\ x : R,\ s \vdots R \Rightarrow \{x \leftarrow s\}t \vdots S$

well-sorted substitution σ

$x : S \Longrightarrow \sigma x \vdots S$

Matching and Unification.

Prop: Matching is decidable and has linear complexity

Theorem:

i) Unification is not of type 0.
ii) Minimal unifier sets are recursively enumerable
iii) It may be of type ∞.
iv) Unification may be undecidable

Example for unificaiton type ∞:

$NAT \sqsubset INT$,
$0:NAT$,
$s(0): NAT$
$s(s(x_{NAT})):NAT$

$\langle s(x_{NAT}) \sqsubseteq NAT \rangle$
has the infinite set of unifers:
$\{0, s(0), s(s(0)), \ldots \}$.

Linear Signatures:

$t:S$ only for linear terms t.

In elementary signatures:
Unification is decidable and finitary.

Complexity for elementary signatures:
Unification is linear for simple signatures
Unification is NP-complete for nonsimple signatures.
The number of unifiers may be exponential.

If the signature is linear, then linear unification problems are decidable.

If all function symbols have arity ≤ 1, then unification is decidable.

If declarations are of the form:
$f(x_1,x_2,g(a)):S$,
then unification is decidable.

Properties of a a calculus with declarations:

Order-sorted resolution is complete.

Order-sorted paramodulation is incomplete:

$$
\begin{aligned}
\Sigma := \{\quad & B,C \sqsubset A,\\
& f\colon A\mathrm{x}A \to A,\ B\mathrm{x}B \to B\\
& \quad C\mathrm{x}C \to B\\
& h\colon B \to B\\
& b{:}B,\ c{:}C\}
\end{aligned}
$$

$$
\begin{aligned}
\{ & b = c\\
& h(f(b\ b)) \neq h(f(c\ c))\\
& x = x\ \}
\end{aligned}
$$

is unsatisfiable, but not refutable.

<u>Term rewriting systems:</u>

Requirements:

Sort-preserving:

$s \longrightarrow_R t$ should imply $\mathbf{LS}_\Sigma(s) \sqsupseteq \mathbf{LS}_\Sigma(t)$.

Theorem: If R is

i) sort-preserving,
ii) critical pairs are confluent
iii) R is terminating

then R is canonical.

Proposition: If Σ is linear, then
R is sort-preserving, if all
critical sort-relations are satisfied.

<u>critical sort-relation:</u>

~~If Σ is linear,~~
~~$t{:}S$, $l \to r$,~~
~~overlap l with a subterm of t, rewrite~~
~~$t \to t'$.~~
~~$LS_\Sigma(t') \sqsubseteq S$ is the critical sort-relation.~~

critical sort relation

$t : S$ term declaration

$l \to r$ rewrite rule

σ : unifier of t/π and l

$((\sigma t)[\pi \leftarrow \sigma r]) : S$

linearity of Σ :

$(t\, [\pi \leftarrow \sigma r]) : S$.

Example : $EVEN \sqsubseteq NAT$

$0 : EVEN$

$s : NAT \to NAT$

$s(s(x_{EVEN})) : EVEN$

$R : \; s(s(x_{EVEN})) \longrightarrow x_{EVEN}$

Note : $s(y_{EVEN}) = z_{EVEN}$ unsolvable

No critical pair

One critical sort relation :

$\boxed{s(s(x_{EVEN})) : EVEN}$

Σ not linear,

•

Example: INT, ZERO,NAT
s,+,-

(x – x): ZERO

((s(0) + s(0)) - (s(0) + s(0))) of sort ZERO.

↓

(s(s(0)) - (s(0) + s(0))) of sort INT

↓

(s(s(0)) - s(s(0))) of sort ZERO

Idea: parallel term rewriting on the same term.

Theorem: If R is
- i) (parallel) sort-preserving,
- ii) critical pairs are confluent
- iii) R is terminating,

then R is canonical.

Proposition:
R is parallel sort-preserving, iff all parallel critical sort-relations are satisfied.

Feature Unification

Gert Smolka, Universitaet Kaiserslautern, West Germany

Hassan Ait-Kaci, MCC, Austin, Texas

Feature terms are record-like data structures for knowledge representation. Unification of two feature terms computes a new feature term representing their combined information. Feature terms and their unification are employed in grammar formalisms in computational linguistics and in the logic programming languages LOGIN (MCC) and CIL (ICOT).

We give a semantics of feature terms using order-sorted equational logic. This semantics accommodates feature terms as the syntactic representation of certain equation systems, thus providing for meaningful initial models and the coexistence of feature terms with ordinary terms.

Based on our semantic reconstruction, we generalize the notion of unification such that it accommodates feature unification. Unification is seen as a constraint solving process, which simplifies the equation system to be solved until it either detects inconsistency (no solution) or arrives at an equation system in solved form (at least one solution).

FEATURE UNIFICATION

Gert Smolka

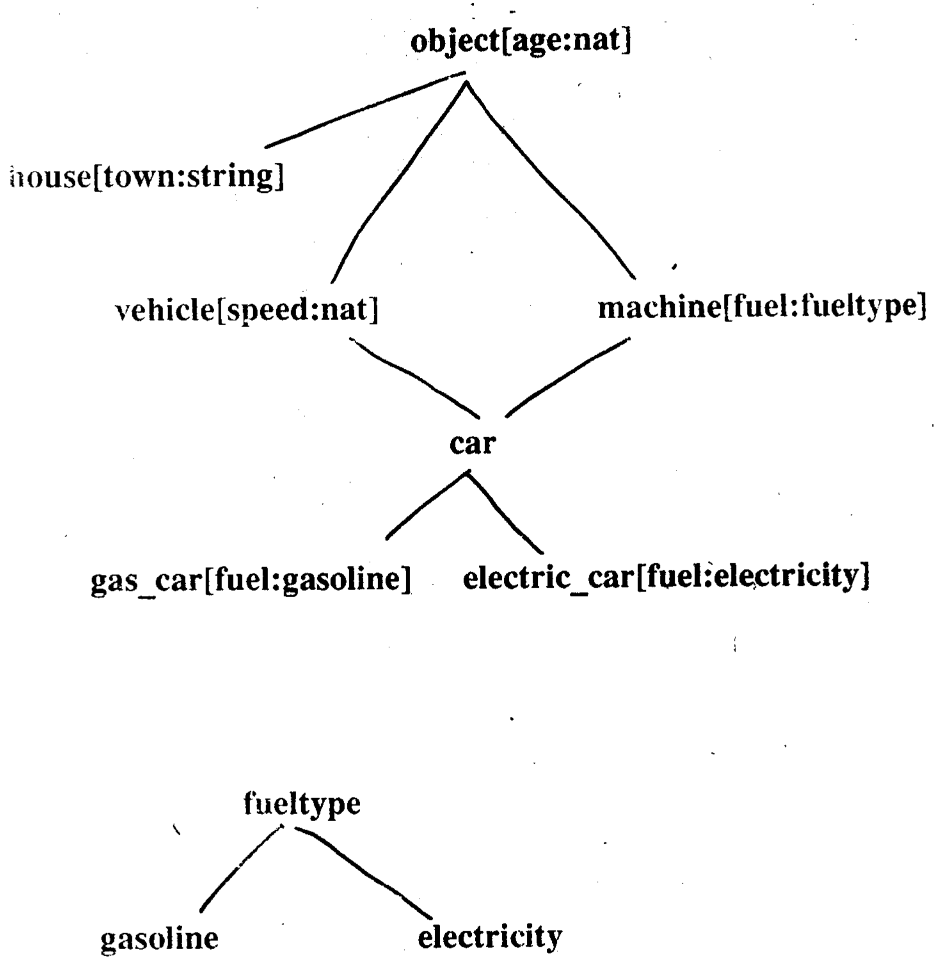
Universität Kaiserslautern

and

Hassan Ait-Kaci

MCC

An Inheritance Hierarchy



object[age:nat]

house[town:string]

vehicle[speed:nat]

machine[fuel:fueltype]

car

gas_car[fuel:gasoline]

electric_car[fuel:electricity]

fueltype

gasoline

electricity

Feature Terms and Feature Unification

machine[age ⇒ 5, fuel ⇒ electricity]

+

vehicle[age ⇒ N:nat, speed ⇒ N]

car[age ⇒ 5, speed ⇒ 5, fuel ⇒ electricity]

Feature terms represent information;
feature unification is an information combining operation.

Hassan Ait-Kaci, PhD Thesis 1984
presents feature terms and feature unifieation in a
lattice-theoretic framework.

What is Feature Unification good for?

Logic Programming

- **LOGIN (Ait-Kaci and Nasr at MCC)**
- **Unification Grammars**
- **Object-Oriented Programming**

Knowledge Representation

- **Frames, Semantic Networks, Inheritance**

Generalizes Ordinary Unification

$f(a, g(b,x))$

$f[1 \Rightarrow a, 2 \Rightarrow g[1 \Rightarrow b, 2 \Rightarrow x]]$

- **arity of functions becomes flexible**
- **functions turn into types~~functions~~ that are partially ordered.**

What has Feature Unification to do with Logic?

- **Semantics? Models? Initial Models?**
- **Completeness and Soundness Properties?**
- **What about combination with existing operational methods like rewriting, narrowing, theory unification?**

These questions aren't answered in Ait-Kaci's thesis.

In the Summer of 1986 we came up with the ideas for a reconstruction of feature terms and their unification in order-sorted Horn logic. This reconstruction provides a well-understood semantical foundation and answers the questions raised above.

Order-Sorted Horn Logic

definite clause logic with equations and subtypes

Some historical remarks:

- Order-Sorted Algebra

  Goguen 1978
  for use in algebraic specifications
  ...
  Meseguer, Jouanaud, Smolka, Kirchners

- Order-Sorted Unification

  Walther 1983
  for use in automated theorem proving
  ...
  Schmidt-Schauß, Cunningham and Dick

- was actually invented by logicians before the advent of computer science: Arnold Oberschelp published 1962 in the Mathematische Annalen a paper describing a predicate logic with subtypes and multiple declarations.

Abstract Syntax

Declarations		
	$\xi < \eta$	subtype declaration
	$f: \xi_1 \ldots \xi_n \rightarrow \xi$	function declaration
	$p: \xi_1 \ldots \xi_n$	relation declaration
Signature	Σ : set of declarations	
Σ-Variables	x, y, z	τx : type of x
Σ-Terms	x, $f(s_1,\ldots,s_n)$	τs : least type of s
Σ Atoms	$s=t$, $p(s_1,\ldots,s_n)$	
Σ Goals	P_1 & ... & P_n	
Σ Clauses	$P \leftarrow G$	
Specification	$\mathcal{S} = (\Sigma, \mathcal{C})$	

Models and Homomorphisms

A Σ-model $\mathcal{A}$ consists of denotations $\xi_{\mathcal{A}}$, $f_{\mathcal{A}}$, $p_{\mathcal{A}}$ as follows:

- $\xi_{\mathcal{A}}$ is a set; the union $A := \bigcup \xi_{\mathcal{A}}$ is called the carrier of $\mathcal{A}$

- if $(\xi < \eta) \in \Sigma$ then $\xi_{\mathcal{A}} \subseteq \eta_{\mathcal{A}}$

- $f_{\mathcal{A}}$ is a partial function $A^{|f|} \rightarrow A$

- if $(f: \xi \rightarrow \eta) \in \Sigma$ then $f_{\mathcal{A}}$ is defined on $\xi_{\mathcal{A}}$ and $f_{\mathcal{A}}(\xi_{\mathcal{A}}) \subseteq \eta_{\mathcal{A}}$
- $p_{\mathfrak{A}} \subseteq A^{|p|}$

A Σ-homomorphism from $\mathcal{A}$ to $\mathcal{B}$ is a mapping $\gamma: A \rightarrow B$ such that:

- $\gamma(\xi_{\mathcal{A}}) \subseteq \xi_{\mathcal{B}}$

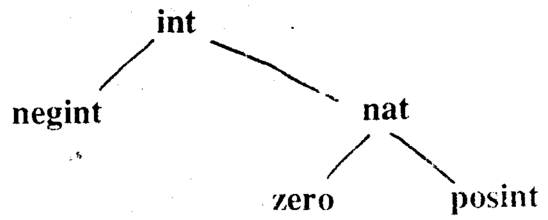
- if $f_{\mathcal{A}}$ is defined on $D \subseteq A$ then $f_{\mathcal{B}}$ is defined on $\gamma(D)$

- $\gamma(f_{\mathcal{A}}(a)) = f_{\mathcal{B}}(\gamma(a))$

THEOREM. Every specification has an intial model.

Validity $\mathcal{S} \models G$ is defined as usual.

Constructor-Oriented Type Definitions



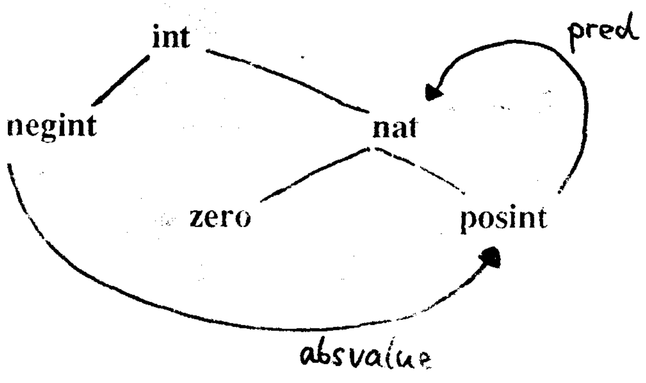
negint<int, zero<nat, posint<nat, nat<int

o: → zero, s: nat → posint, -: posint → negint

int := negint + nat
negint := {-: posint}
nat := zero + posint
zero := {o}
posint := {s: nat}

Semantics: free initial algebra

Features(= Selectors)



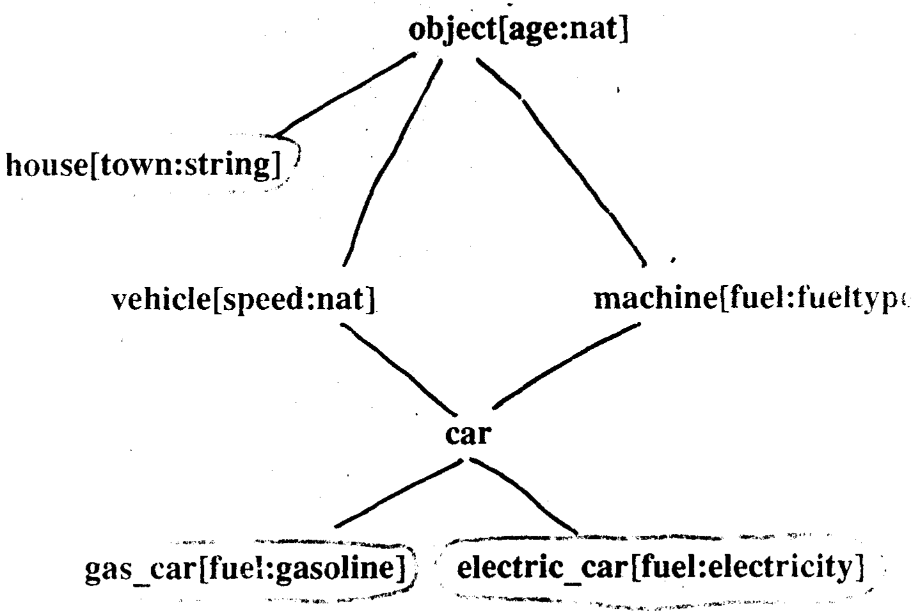
pred: posint → nat
  pred(s(N)) = N

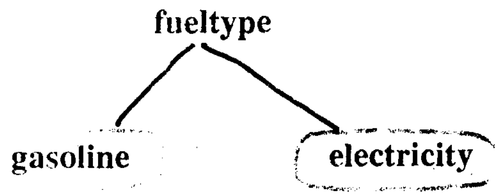
absvalue: negint → posint
  absvalue(-P) = P

Feature Term Syntax

o	zero
s(o)	posint[pred ⇒ zero]
s(s(o))	posint[pred ⇒ posint[pred ⇒ zero]]
⋮	
-s(o)	negint[absvalue ⇒ posint[pred ⇒ zero]]
⋮	

• Implicit Constructors:





Constructor-Oriented Definition of the Inheritance Hierarchy

```
object := house + vehicle + machine
house := {con_house: nat x string}
vehicle := car
machine := car
car := gas_car + electric_car
gas_car := {con_gas_car: nat x nat x gasoline}
electric_car := {con_electric_car: nat x nat x electricity}
fueltype := gasoline + electricity
gasoline := {con_gasoline}
electricity := {con_electricity}

age: object → nat
    age(con_house(A,T)) = A
    age(con_gas_car(A,S,G)) = A              Projections
    age(con_electric_car(A,S,E)) = A

fuel: machine → fueltype                     Multiple
fuel: gas_car → gasoline                     Declarations
fuel: electric_car → electricity
    age(con_gas_car(A,S,G)) = G
    age(con_electric_car(A,S,E)) = E

        ⋮
```

Feature Terms are Syntactic Sugar

```
M = machine[age ⇒ 4; fuel ⇒ electricity]
V = vehicle[age ⇒ N:nat, speed ⇒ N]
M = V
```

get rid of feature term syntax

```
age(M) = 5 & fuel(M) = E &
age(V) = N & speed(V) = N &
M = V
where M:machine, V:vehicle, E:electricity, N:nat
```

feature unification

```
M= V =C &
age(C) = 5 & speed(C) = 5 & fuel(C) = E &
where M:machine, V:vehicle, E:electricity
```

go back to feature term syntax

```
M = V = car[age ⇒ 5, speed ⇒ 5, fuel ⇒ electricity]
```

Unification as Constraint Solving

$\mathcal{S} = (\Sigma, \mathcal{E})$ specification

$VAR = PV \uplus AV$ primary and auxiliary variables

$U_{\mathcal{S}}(E) := \{\theta|_{PV} \mid \mathcal{S} \models \theta G\}$ $\mathcal{S}$-unifiers of E

Crucial Question: Is $U_{\mathcal{S}}(E)$ nonempty?

Define solved form for equation systems and constraint solving rules $E \rightarrow E'$ such that:

1. if E is solved, then $U_{\mathcal{S}}(E)$ is nonempty

2. Invariance: if $E \rightarrow E'$, then $U_{\mathcal{S}}(E) = U_{\mathcal{S}}(E')$

3. Completeness: if $U_{\mathcal{S}}(E)$ is nonempty, then there exists a solved equation system S such that $E \rightarrow^{*} S$

4. Effectiveness:
   (a) $E \rightarrow E'$ terminates
   (b) it is decidable whether E is in solved form

1. Example

Σ-Unification where Σ is single-sorted

- E is in solved form if E is the equational representation of an idempotent substitution: $x_1=s_1$ & ... & $x_m=s_m$.

- Constraint solving rules [Herbrand 1930]

  Decomposition
  $f(s_1,...,s_n)=f(t_1,...,t_n)$ & E → $s_1=t_1$ & ... & $s_n=t_n$ & E

  Isolation
  x=s & E → x=s & E[x/s]
  if x occurs in E but not in s

  Orientation
  s=x & E → x=s & E

  Elimination
  x=x & E → E

Constrained-oriented approach to unification was rediscovered by Huet, Colmerauer, Martelli/Montanari, ...

2. Example: Feature Unification

Solved Form:

$x_1=s_1$ & ... & $x_m=s_m$ & $l_1(y_1)=t_1$ & ... & $l_n(y_n)=t_n$

where $m,n \geq 0$ and

1. $x_1,\dots,x_m$ occur only once
2. $l_1(y_1),\dots,l_n(y_n)$ are pairwise distinct quasi-variables
3. $\tau s_i \leq \tau x_i$ and $\tau t_j \leq \tau l_j(y_j)$ for all i and j
4. no cycles

$l(x)$ is called quasi-variable if l is a feature function and x is a variable

By precompilation bring all equations into the form

can_term = can. term

or quasi_variable = can_term

Feature Unification Rules

Decomposition

- $f(s_1,\dots,s_n)=f(t_1,\dots,t_n)$ & E → $s_1=t_1$ & ... & $s_n=t_n$ & E

Isolation

- $x=s$ & E → $x=s$ & E[x/s]
  if x occurs in E but not in s and $\tau s \leq \tau x$

  $x=y$ & E → $x=z$ & $y=z$ & E[x/z, y/z]
  if not $\tau y \leq \tau x$ and z is a new auxiliary variable such that τz is is the greatest common subtype of τx and τy

- $l(x)=y$ & E → $l(x)=z$ & $y=z$ & E[y/z]
  if not $\tau y \leq \tau l(x)$ and z is a new auxiliary variable such that τz is the greatest common subtype of $\tau l(x)$ and τy

Merging

- $l(x)=s$ & $l(x)=t$ & E → $l(x)=s$ & $s=t$ & E

Orientation

- $s=x$ & E → $x=s$ & E
  if s is neither a variable nor a quasi-variable

Elimination

- $x=x$ & E → E

Feature Unification is better than Narrowing

Feature unification is unitary.

Order-sorted unification and narrowing aren't unitary.

age: object → nat
  age(con_house(A,T)) = A
  age(con_gas_car(A,S,G)) = A
  age(con_electric_car(A,S,E)) = A

age(o) $\overset{?}{=}$ 71

→ O = object[age ⇒ 71]

→ O = con_house(71, T)
  ∨ O = con_gas_car(71, S, G)
  ∨ O = con_electric_car(71, S, E)

Order-Sorted Normalizing Basic Narrowing

Gert Smolka and Werner Nutt

Universitat Kaiserslautern, West Germany

We present a constraint-oriented narrowing calculus, which realizes Hullot's basic narrowing strategy in a natural way. The calculus is based on orded-sorted equational logic and applies to rewrite systems that are type-decreasing, confluent and terminating.

The calculus is optimized by adding a rule that can be used to rewrite the equation system to be solved. This rule keeps the solution space invariant. We prove the completeness of the thus obtained combination of basic and normalizing narrowing.

In general, order-sorted unification isn't unitary, thus blowing up the search spaces defined by our narrowing calculus. To avoid this problem, we consider so-called stratified rewrite systems, for which our calculus requires unification only with respect to a subsignature.

Order-Sorted
Basic
Normalizing
Narrowing

Gert Smolka
Werner Nutt

FB Informatik
Universität Kaiserslaute

Overview

1. Narrowing as Constraint Solving

   Basic Narrowing

2. Optimizations

   Basic Normalizing Narrowing

3. Order-Sorted Rewriting

4. Order-Sorted Basic Normalizing Narrowing

·Preliminaries

$$\mathcal{R} = (\Sigma, \mathcal{E})$$

$$u \to v$$

confluent, terminating

$$E = s_1 \doteq t_1 \,\&\, \ldots \,\&\, s_n \doteq t_n$$

$$V = PV \uplus AV$$

$$\mathcal{U}_{\mathcal{R}}(E) = \{ \theta_{|PV} \mid \mathcal{R} \models \theta E \}$$

$$\mathcal{NU}_{\mathcal{R}}(E) = \{ \theta_{|PV} \mid \begin{array}{l} \mathcal{R} \models \theta E \\ \theta \ \mathcal{R}\text{-normal} \end{array} \}$$

Proposition

$$_{\mathcal{R}}(E) = \mathcal{U}_{\mathcal{R}}(E') \iff \mathcal{NU}_{\mathcal{R}}(E) = \mathcal{NU}_{\mathcal{R}}(E$$

Unification

Unification calculus (Herbrand 1930)

- "$E \xrightarrow[u]{} E'$" terminating
- $E \xrightarrow[u]{} E' \Rightarrow \mathcal{U}_\Sigma(E) = \mathcal{U}_\Sigma(E')$
- $\left.\begin{array}{l} E \text{ not solved} \\ \mathcal{U}_\Sigma(E) \neq \phi \end{array}\right\} \Rightarrow \exists E'.\, E \xrightarrow[u]{} E'$

Theorem

If $\mathcal{U}_\Sigma(E) \neq \phi$,
then there exists S
s.t.

- $E \xrightarrow[u]{*} S$
- $\mathcal{U}_\Sigma(E) = \mathcal{U}_\Sigma(S)$

Narrowing.

Narrowing pair $S.E$

solved part

unsolved part

$\mathcal{R}$-normal

Devise narrowing calculus

$\longrightarrow S'.E'$ "

such that

$$S.E \longrightarrow S'.E' \Rightarrow \mathcal{U}_\mathcal{R}(S' \& E') \subseteq \mathcal{U}_\mathcal{R}(S \& E$$

Soundness

$$\left.\begin{array}{l} \mathcal{R} \models \theta E \\ \theta \text{ normal} \end{array}\right\} \Rightarrow \exists S.\ \begin{array}{c} \phi.E \xrightarrow{\mathcal{R}\ *} S.\phi \\ \text{and} \\ \theta \in \mathcal{U}_\Sigma(S) \end{array}$$

Completeness

Rules for Basic Narrowing

Rule

$$S.P\&E \xrightarrow[u]{\mathcal{R}} S'.E$$

$$\text{if} \quad S\&P \xrightarrow[u]{*} S'$$

Application Rule (Martelli, ...)

$$S.P\&E \longrightarrow S.\ P/\pi \doteq u \ \&\ P[\pi \leftarrow v] \ \&\ E$$

$$\text{if} \quad u \longrightarrow v \in \mathcal{R}$$

$$\mathrm{topsym}(P/\pi) = \mathrm{topsym}(u)$$

Rule

$$S.P\&E \xrightarrow[n]{\mathcal{R}} S'.\ P[\pi \leftarrow v] \ \&\ E$$

$$\text{if} \quad u \longrightarrow v \in \mathcal{R}$$

$$\mathrm{topsym}(P/\pi) = \mathrm{topsym}(u)$$

$$S \ \&\ P/\pi \doteq u \xrightarrow[u]{*} S'$$

: no narrowing on S

THEOREM

The calculus for Basic Narrowing is sound and complete.

Proof:

Soundness ✓

Completeness: proved with lifting techni...

$$\begin{tikzcd}
 & \theta S \text{ trivial} & \theta' S' \text{ trivial} \\
\text{narrowing} & S.E \arrow[r] & S'.E' \\
\text{level} & \text{PUSH UP} & \text{THEOREM} \\
\text{rewriting} & \theta E \arrow[r] \arrow[u, Rightarrow] & \theta' E
\end{tikzcd}$$

Optimizations

$$S.E \xrightarrow[u,r]{\mathcal{R}} S'.E' \longrightarrow \cdots \quad ?$$

$$\mathcal{U}_{\mathcal{R}}(S\&E) \supsetneqq \mathcal{U}_{\mathcal{R}}(S'\&E') \supsetneqq \cdots$$

Devise a rewriting rule

$$S.E \xrightarrow[r]{\mathcal{R}} S'.E'$$

$$\mathcal{U}_{\mathcal{R}}(S\&E) = \mathcal{U}_{\mathcal{R}}(S'\&E')$$

$$S.P\&E \xrightarrow[r]{\mathcal{R}} S.P[\pi \leftarrow v] \,\&\, E$$

if $u \to v \in$ Instances$(\mathcal{R})$

and $\langle S \rangle (P/\pi) = u$

Example:

rule $\quad 0 + s(N) \longrightarrow s(N)$

$$x \doteq 0 \,\&\, y \doteq s(s(s(z))) \,.\, x + y \doteq w + w'$$

$$\Big\downarrow_{r}^{\mathcal{R}}$$

$$x \doteq 0 \,\&\, y \doteq (s(s(s(z))) \,.\, \underbrace{s(s(s(z)))}_{\text{too big}} \doteq w + w'$$

$x \doteq$ & $y \doteq s$

$z' \doteq s(s(z))$

$$\Big\downarrow_{r}^{\mathcal{R}}$$

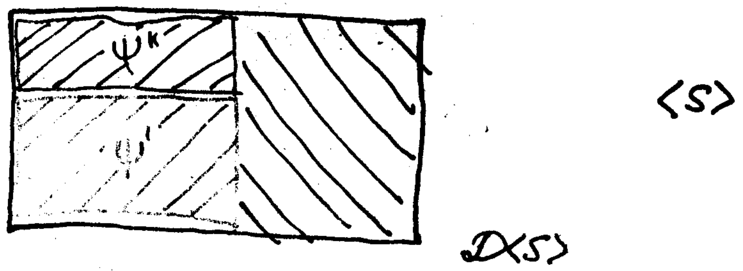
$$x \doteq 0 \,\&\, y \doteq s(s(s(z))) \,\&\, z' \doteq s(s(z)) \,.$$

$$.\; s(z') \doteq w + w'$$

Optimized Rewriting Rule

11

factorisation of $\langle S\rangle$:



$\psi'' x = \langle S\rangle x$, if $x \in \mathcal{D}$

$$S.P\&E \xrightarrow[\rho]{\mathcal{R}} S\&[\psi''].P[\pi\leftarrow v]\&E$$

if $u \rightarrow v \in$ Instances $(\mathcal{R})$

ψ'', ψ' partial factorisation of $\langle S\rangle$

$\psi''(P/\pi) = u$

Normalizing ~~Lazy~~ Basic Narrowing

$$S.E \xrightarrow[e]{\mathcal{R}} S'.E'$$

if

$$\left.\begin{array}{l} S.E \xrightarrow[u]{\mathcal{R}} S''.E'' \\ S.E \xrightarrow[n]{\mathcal{R}} S''.E'' \end{array}\right\} \xrightarrow[\rho]{*} S'.E'$$

where $\langle S'\rangle E'$ is $\mathcal{R}$-normal

THEOREM Normalizing ~~Lazy~~ Basic Narrowing is sound and complete

Proof:

Soundness ✓

Completeness

θS trivial

$$\begin{tikzcd} S.E \arrow[r, "\rho"'] \arrow[d, Rightarrow, "\text{PUSH}"', "\text{DOWN THEOREM}"] & S'.E' \\ \theta E \arrow[r] & \theta' E' \end{tikzcd}$$

$\exists\theta.$

narrowing level

verification level

$\theta' S'$ trivial

Order-Sorted

is type decreasing if

$s \xrightarrow{\mathcal{R}} t$ implies $type(t) \leq type(s)$

and type decreasing

If [illegible] type decreasing, then

(Decidability)

[illegible] finite, then

[illegible] type decreasing ?"

is [illegible]

Rewriting

$\mathcal{R} = (\Sigma, \mathcal{E})$

Theorem (Soundness and Completeness)

If $\mathcal{R}$ is confluent then

$\mathcal{R} \models s \doteq t \iff s \downarrow_{\mathcal{R}} t$

(Knuth/Bendix)

$\mathcal{R}$ locally confluent $\iff$ all critical pairs of $\mathcal{R}$ converge

- **Order-Sorted Narrowing.**

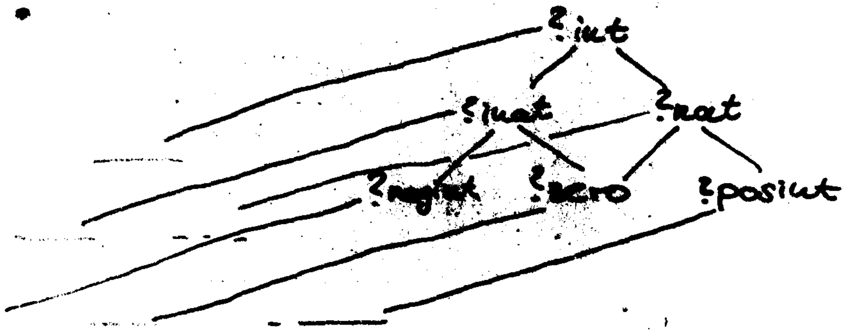
$\mathcal{R} = (\Sigma, \mathcal{E})$

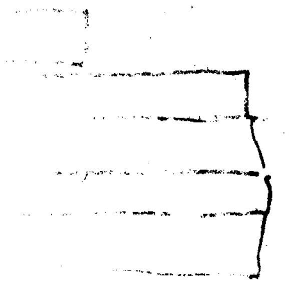
results carry over if

$\mathcal{R}$ type-decreasing

Σ unification or unitary

Stratified



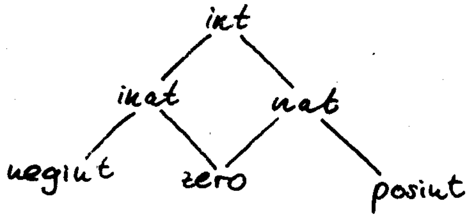


s,p: ?int ⟶ ?int

? ? ?
? ? ?
? ? ?

Initial Semantics doesn't change!

Order-sorted Specifications



0 : zero

s : nat $\longrightarrow$ posint

int $\longrightarrow$ int

p : inat $\longrightarrow$ negint

int $\longrightarrow$ int

$$s(p(I)) \doteq I$$

$$p(s(I)) \doteq I$$

VAR I,J : int

$$\begin{array}{lllll} + : & int & \times & int & \longrightarrow & int \\ & posint & \times & nat & \longrightarrow & posint \\ & nat & \times & posint & \longrightarrow & posint \end{array}$$

$$0 + I \doteq I$$

$$s(I) + J \doteq s(I+J)$$

$$p(I) + J \doteq p(I+J)$$

Advantages:

$$\emptyset \,.\, I + s(s(0)) \doteq 0$$

$$\xrightarrow{\mathcal{R}\ *} I \doteq p(p(0)) \,.\, \emptyset$$

Disadvantages:

$$\emptyset .\ I+J \doteq K$$

$$\xrightarrow{\mathcal{R}\ *} I \doteq 0,\ K \doteq J \quad .\ \emptyset$$

$$\xrightarrow{\mathcal{R}\ *} I \doteq s(0),\ K \doteq s(J) \quad .\ \emptyset$$

$$\xrightarrow{\mathcal{R}\ *} I \doteq p(0),\ K \doteq p(J) \quad .\ \emptyset$$

. . .

infinitely many solutions

Improving basic narrowing techniques

Pierre Réty

Centre de Recherche en Informatique de Nancy
BP 239
54506 Vandoeuvre Les Nancy Cedex
France
E–mail: mcvax!inria!crin!rety

Abstract

In this paper, we propose a new and complete method based on narrowing for solving equations in equational theories. It is a combination of basic narrowing and narrowing with eager reduction, which is not obvious, because their naive combination is not a complete method. We show that it is more efficient than the existing methods in many cases, and for that establish commutation properties on the narrowing. It provides an algorithm that has been implemented as an extension of the REVE software.

IMPROVING BASIC NARROWING TECHNIQUES

Narrowing $(\rightsquigarrow)$: $\begin{cases} \text{instanciation by the m.g. unifier} \\ \text{- réduction by rules} \end{cases}$

S-narrowing $(\rightsquigarrow)$: réduction by one rule

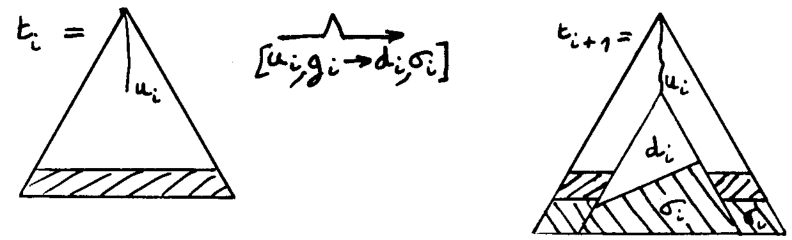
N-narrowing $(\rightsquigarrow)$: réduction = normalization

Basic S-narrowing [Hullot]

$$t_0 \xrightarrow[{[u_0, g_0 \to d_0, \sigma_0]}]{\wedge} \cdots \xrightarrow[{[u_n, g_n \to d_n, \sigma_n]}]{\wedge} t_{n+1}$$

is based on U_0 iff for all i

- $u_i \in U_i$
- $U_{i+1} = [U_i - \{v \in U_i \,/\, v \geqslant u_i\}] \cup \{u_i.v \,/\, v \in O(d_i)\}$



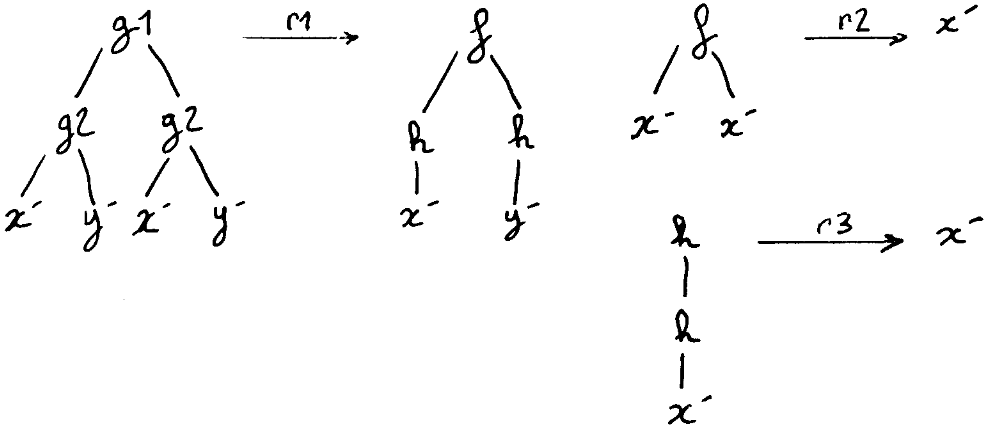
Theorem: The above narrowing relations give complete methods for solving equations

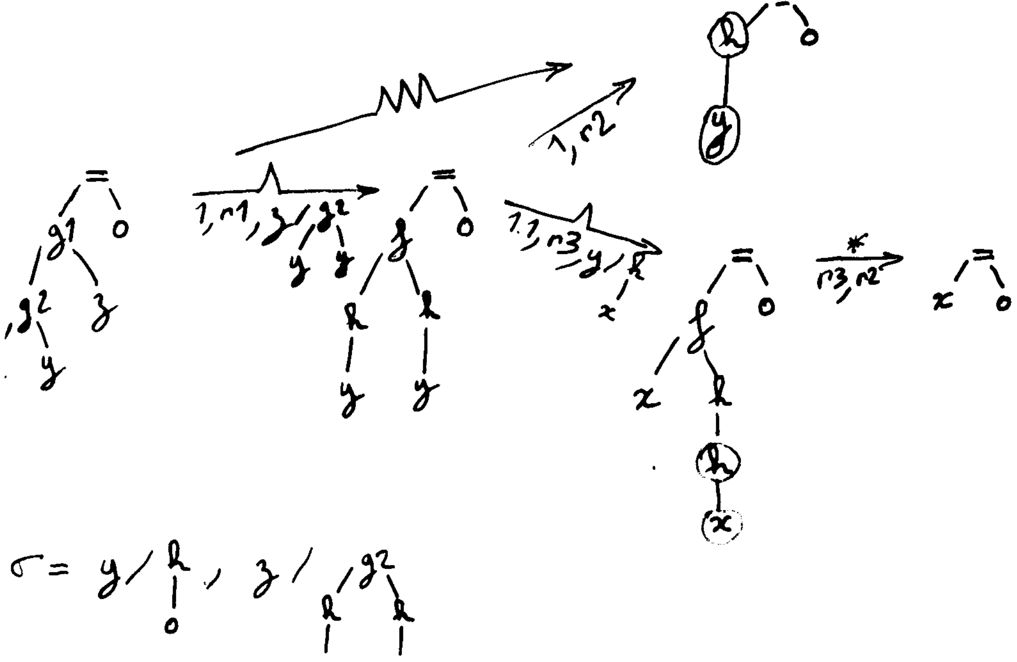
Naive basic N-narrowing

$t_0 \rightsquigarrow^* t_n$ is basic $\quad (\equiv t_0 (\overset{\wedge}{\longrightarrow} \xrightarrow{*})^* t_n)$

iff $\quad t_0 \overset{\wedge}{\longrightarrow}{}^* t_n$ is basic

This is not a complete method.





Along a reduction, a subterm can be preserved

Antecedent (dual of residual [Huet-Levy])

$t \longrightarrow [u, g \to d, \sigma]\ t'$

$v' \in \mathcal{D}(t')$

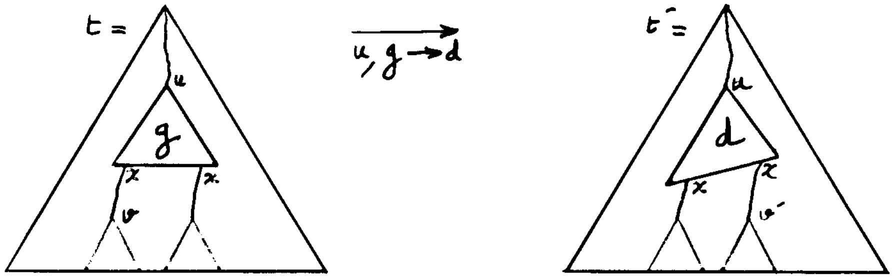
$v \in \mathcal{D}(t)$ is an antecedent of v' iff

v' is not comparable to u and $v = v'$

there exists an occurrence p' of a variable x in d such that

$v' = u.p'.w$

$v = u.p.w$ where p is an occurrence of x in g



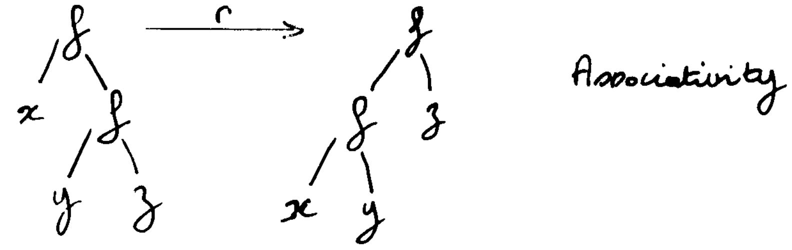
weakly. basic reduction

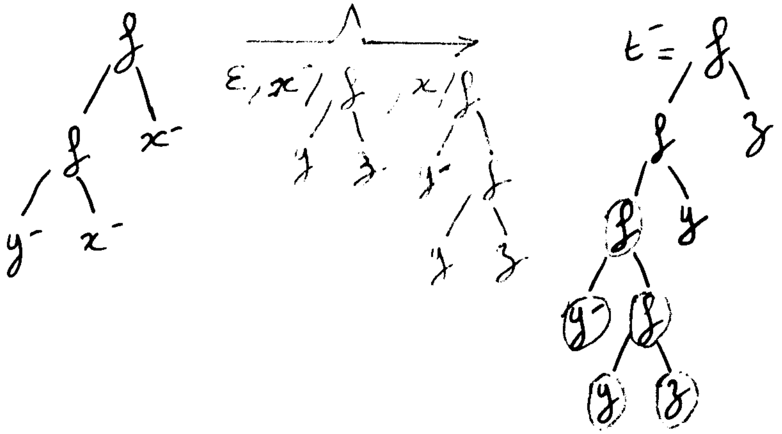
$t_0 \xrightarrow[u_0]{} \quad \cdots \quad \xrightarrow[u_{n-1}]{} t_n$

is weakly based on U_0 iff for all i:

- $u_i \in U_i$
- $U_{i+1} = \{ v \in O(t_{i+1})$, all the antécedents of v (in t_i) belong to $U_i \}$

Ihe reduction is not always possible with respect to basic occurrences.





t^- cannot be normalized by a basic reduction

Sufficient largeness

Consider a term t, and $U \subset D(t)$
U is sufficiently large on t iff

$u \in D(t),\ u \notin U \implies t/u$ is normalized

Lemma: If U is sufficiently large on t then any derivation issued from t is weakly based on U.

SL-basic narrowing

The step of narrowing

$t_0 \xrightarrow{M} t_n \qquad (\equiv t_0 \xrightarrow{\Lambda} t_1 \xrightarrow{*} t_n)$

is SL-based on $U_0 \subset D(t_0)$ iff

a) $t_0 \xrightarrow{\Lambda} t_1$ is based on U_0

b) The set U_1 obtained by a basic computation from U_0, is sufficiently large on t_1

(Then $t_1 \xrightarrow{*} t_n$ is weakly based on U_1)

c) Along $t_1 \xrightarrow{*} t_n$ the sets of basic occurence are computed by a weakly basic computation

We extend this definition:
SL-basic narrowing derivation

Particular cases:

SL-basic S-narrowing
$\neq$ basic S-narrowing [Hullot]

SL-basic N-narrowing.

Theorem (completeness)

R being a confluent and noetherian rewriting system The set of substitutions σ such that

- There exist a narrowing derivation issued from $t_0 = t'_0$ and SL-based on $O(t_0 = t'_0)$

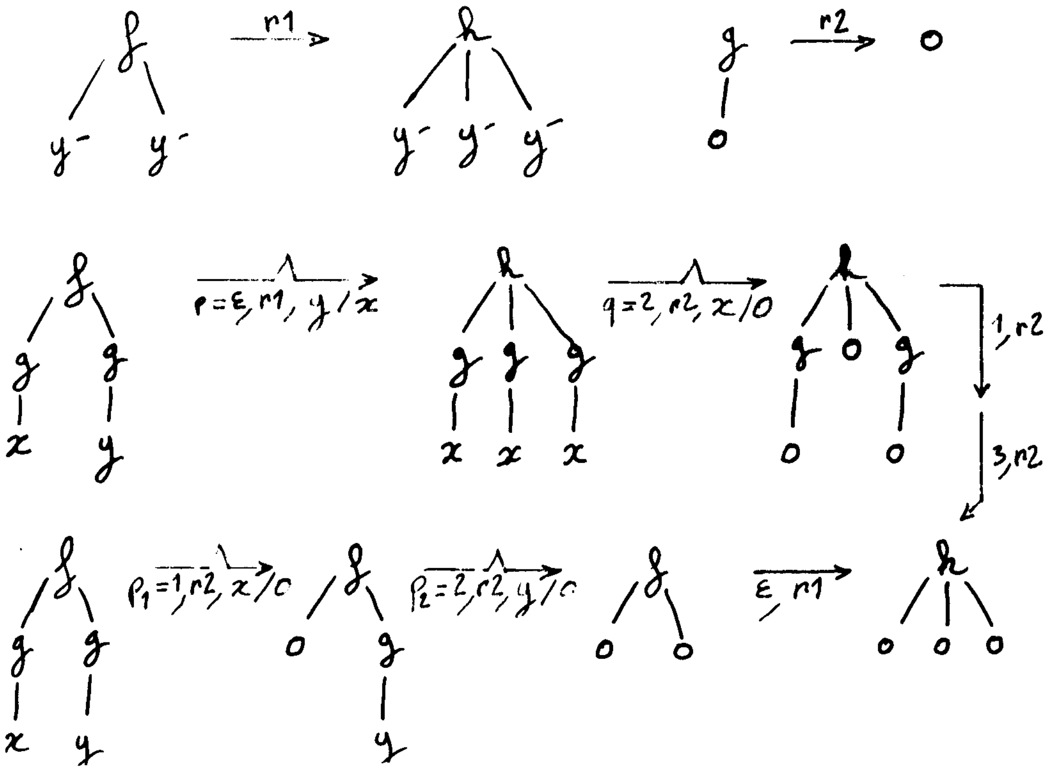
$$t_0 = t'_0 \xrightarrow[\sigma_1]{\mathcal{M}} - \xrightarrow[\sigma_n]{\mathcal{M}} t_n = t'_n$$

where t_n and t'_n are unifiable by the m.g.u. θ

- $\sigma = \theta . \sigma_n \ldots \sigma_1$

is a complete set of R-unifiers of t_0 and t'_0.

Commutation



Theorem: (commutation property)

Let $s \xrightarrow[p, g \to d, \sigma]{\wedge} t \xrightarrow[q, l \to r, \theta]{\wedge} u$ (1)

be two steps of S-narrowing such that

a) q admits antecedents in s: $p_0, \ldots, p_{m-1}$

b) $\theta . \sigma|_{V(s)}$ is normalized

c) $V(r) = V(l)$ or g is linear

Then (1) can be commuted into:

$$s \xrightarrow[p_0, l \to r, \theta'_0]{\wedge} t'_1 \xrightarrow{\wedge} \cdots \xrightarrow[p_{m-1}, l \to r, \theta'_{m-1}]{\wedge} t'_m \xrightarrow[p, g \to d, \sigma']{\wedge} u'$$

such that

a') $\sigma' . \theta'_{m-1} \ldots \theta'_0 = \theta . \sigma \quad [V(s)]$

b') $u \xrightarrow[q_1, l \to r]{} \cdots \xrightarrow[q_m, l \to r]{} u'$

where $q_1, \ldots, q_m$ are the others residuals of p

Comparison

SL-basic S-narrowing $\subset_{str}$ basic S-narrowing

Ex: Associativity.

SL-basic narrowing vs basic S-narrowing

Theorem: Let R be a term rewriting system such that
- R is right-linear
- R is regular or left-linear

Let $t_0 \xrightarrow[\theta]{\mathcal{M}\,*} t_n$ be a narrowing derivation based on $U_0 \subset O(t_0)$, such that $\theta|_{V(t_0)}$ is normalized

Then there exists a S-narrowing derivation $t_0 \xrightarrow[\theta]{\Lambda\,*} t_n$ based on U_0.

Ex: naive

SL-basic N-narrowing vs SL-basic S-narrowing

If R has no critical pair the above theorem holds with these relations

Narrowing optimizations

Peter Padawitz
Universität Passau

Narrowing optimizations are goal transformations applied to subgoals produced by narrowing derivations. The purpose is to speed up the deduction process. Various optimizations have been discussed in the literature (cf. Fribourg, Hußmann,...). We give a local correctness condition for optimizations, which guarantees that narrowing remains sound and complete when equipped with optimizations. Since locally correct optimizations are closed under composition, soundness and completeness holds true even if several optimizations are applied at the same time.

DERIVATION RULES

Base Rule

$\exists$ substitution f obtained from goal $\gamma = \{p_1, \dots, p_n\}$ by revolution with $x \equiv x$ and Horn clause axioms which are not conditional equations

$\Longrightarrow \quad \gamma \vdash <\emptyset, f>$

Narrowing Rule

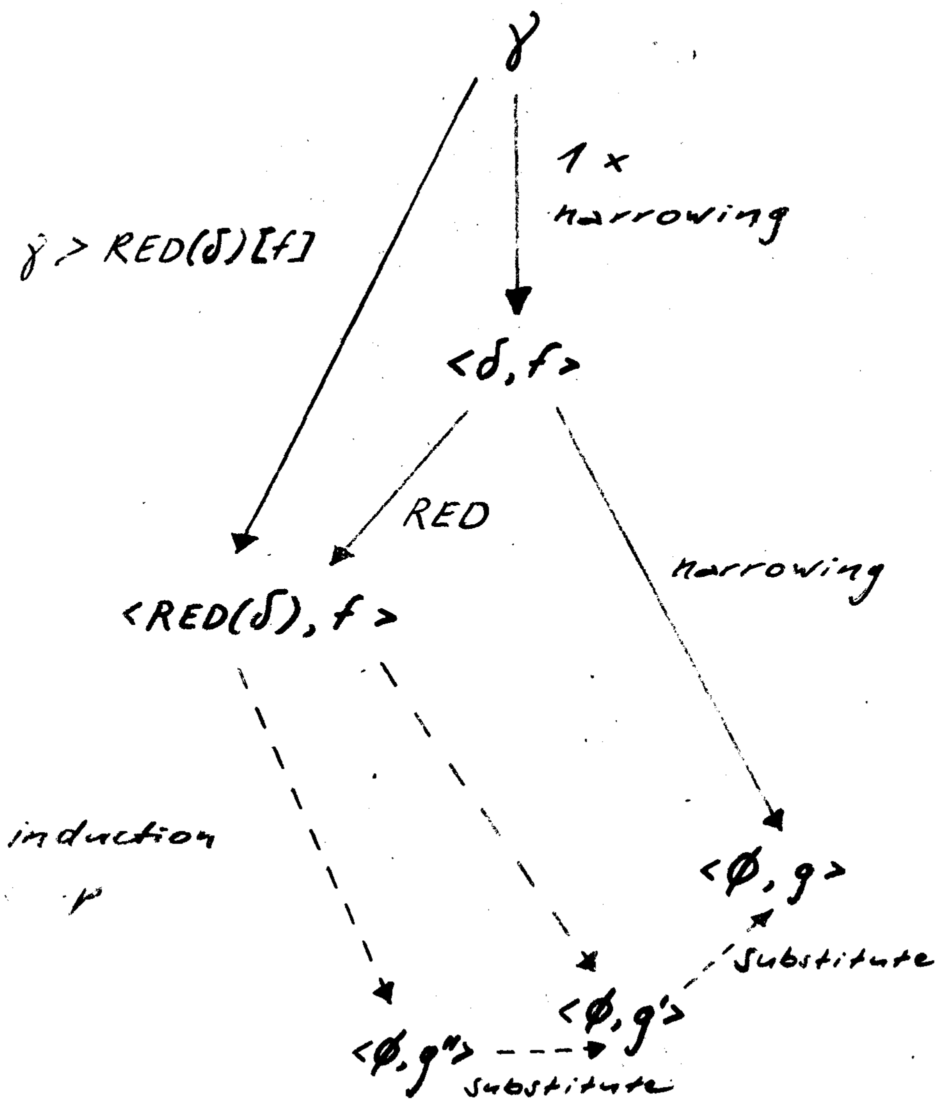
Let $u \equiv u' \Leftarrow \vartheta$ be an axiom, $u[f] = t[f]$, $t \notin Var$

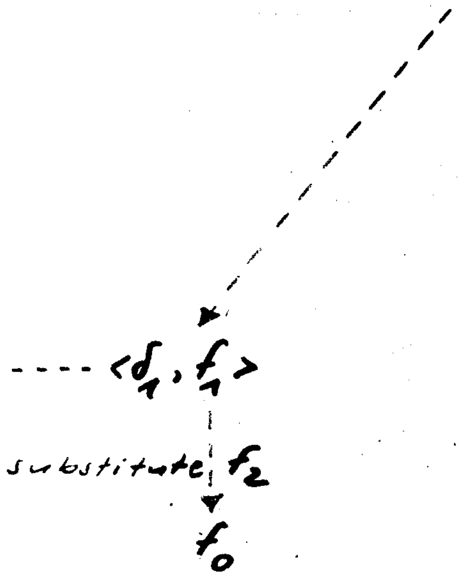
$\Longrightarrow \quad \gamma[t/x] \vdash <(\gamma[u'/x] \cup \vartheta)[f], f>$

Composition Rule

$\gamma \vdash <\delta, f>, \; \delta \vdash <\varphi, g> \Rightarrow \gamma \vdash <\varphi, f[g]>$

LAZY NARROWING



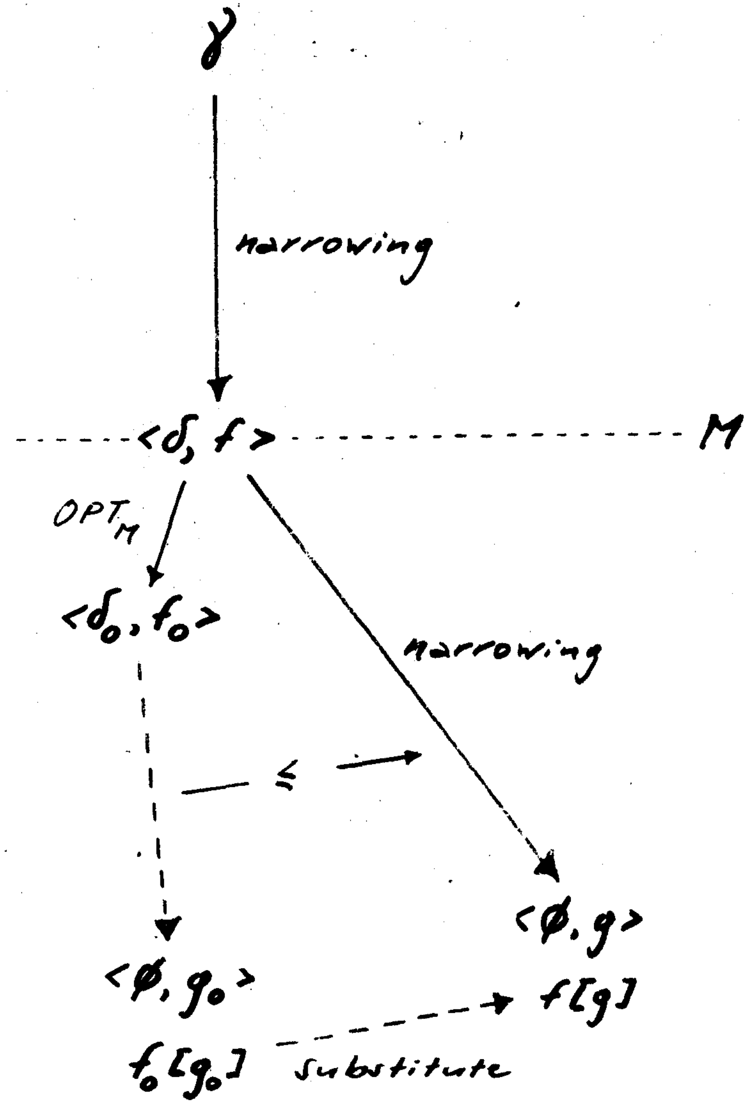


$\vdash \delta_0 [g_1]$

implies

$\delta_1 [f_2][g_1]$

OPTIMIZED NARROWING



OPTIMIZATIONS

$OPT_M(<\delta, f>) = <\delta_0, f_0>$

Goal subsumption

$\delta = \delta_0[h]$ $f = f_0[h]$

$<\delta_0, f_0> \in M$ h base

CE(AX) does not overlap base terms

Solution subsumption

$\delta_0 = \phi$ $f = f_0[h]$

$<\delta_0, f_0> \in M$

Binding

$\delta = \lambda \cup \{x \equiv t\}$ $x \notin var(t)$

$\delta_0 = \lambda[t/x]$ $f_0 = f[t/x]$

CE(A[illegible]) do[illegible] not o[illegible] t

Decomposition I

$\delta = \lambda \cup \{\sigma<t_1 \dots t_n> \equiv \sigma<u_1 \dots u_n>\}$

$\delta_0 = \lambda \cup \{t_1 \equiv u_1, \dots, t_n \equiv u_n\}$

$f_0 = f$ CE(AX) does not overlap σ

Decomposition II

$\delta = \lambda \cup \{x \equiv \sigma<t_1 \dots t_n>\}$ $x \notin var(\sigma<t_1 \dots t_n>)$

$\delta_0 = \lambda[\sigma<x_1 \dots x_n>/x] \cup \{x_1 \equiv t_1, \dots, x_n \equiv t_n\}$

$f_0 = f[\sigma<x_1 \dots x_n>/x]$

CE(AX) does not overlap σ

$OPT_M(<\delta, f>)$ undefined

Clash

$\delta = \lambda \cup \{\sigma<t_1 \dots t_n> \equiv \tau<u_1 \dots u_n>\}$ $\sigma \neq \tau$

CE(AX) does not overlap σ

Refutation

$\delta = \lambda \cup \{p\}$ CE(AX) does not overlap p,

p is not resolvable

?

Narrowing with inductively defined functions

(A. Bockmayr)

The aim of this talk is to investigate the behaviour of the narrowing algorithm when it is applied to typical functional programs. We introduce the notion of a function inductively defined over a set of constructors C and show that for these functions the narrowing algorithm is a variant of the trivial universal unification algorithm that enumerates the term algebra T(C,X). Moreover there are several inefficiencies in this enumeration process.

KAP - PROLOG - GROUP

KAP

METHODS AND TOOLS FOR THE OPTIMIZATION OF LOGIC PROGRAMMING LANGUAGES

UNIVERSITY KARLSRUHE
DFG SONDERFORSCHUNGSBEREICH 314

A. Bockmayr
N. Lindenberg
B. Neidecker
I. Varsek

GMD RESEARCH INSTITUTE
AT THE UNIVERSITY KARLSRUHE

R. Dietrich
P. Kursawe

NARROWING STEP

$t \; N\text{–}\sigma_t\text{–>} \; t'$ means

that there are

an occurrence u in t
a rule $l \text{–>} r$

such that

t/u and l are unifiable with mgu σ

$\sigma = \sigma_t \cup \sigma_l$ where $D(\sigma_t) \subseteq V(t)$ and $D(\sigma_l) \subseteq V(l)$

$t' = \sigma_t(t)\,[u \leftarrow \sigma_l(r)] \downarrow$

σ_t <u>narrowing substitution</u>

OBSERVATION

Different occurrences or different rules may lead to the same narrowing substitution σ_t

NARROWING DERIVATION

$t_1 \; N\text{–}\sigma_1\text{–>} \; t_2 \; N\text{–}\sigma_2\text{–>}\ldots\ldots\ldots \; N\text{–}\sigma_{n-1}\text{–>} \; t_n$

<u>narrowing substitution</u> $\sigma = \sigma_n \circ \ldots \circ \sigma_1$

OBSERVATION

The composition of different (single step) narrowing substitutions σ_i may lead to the same total narrowing substitution σ.

EXAMPLES

Addition and multiplication

```
plus(0,x) —> x ,
plus(s(x),y) —> s(plus(x,y)) ,
plus(x,0) —> x ,
plus(x.s(y)) —> s(plus(x,y)) ,

mult(0,x) —> 0,
mult(s(x),y) —> plus(y,mult(x,y)),
mult(x,0)  —> 0,
mult(x,s(y)) —> plus(mult(x,y),x) .
```

plus and mult are inductively defined over {0,s} in the first and second argument.
~~plus and mult are inductively defined over {s}, but not over {0,s}~~ .

Concatenation of lists

```
append(nil, x) —> x ,
append(cons(a,x), y) —> cons(a,append(x,y))
```

append is inductively defined over $\underset{\{nil,cons\}}{\underset{\shortparallel}{C_0}}$ in the first argument.

Size of a binary tree

```
size (make(n)) —> s(0) ,
size(cons(l,r)) —> plus(size(l),size(r))
```

size is inductively defined over {make,cons}.

Definition

A function $f \in D$, $f : s_1 \ldots s_m \text{—>} s$, $m \geq 1$, is called inductively defined over C_0 in the i-th argument , $i \in \{1,\ldots,m\}$, $s_i \in S_0$, iff for all constructors $c \in C_0$, $c : s_1' \ldots s_n' \text{—>} s_i$, $n \geq 0$, the following condition holds:

1. If $n \geq 1$ and $\{j_1,\ldots,j_k\} =_{def} \{ j \in \{1,\ldots n\} \mid s_j' \in S_0 \} \neq \emptyset$

   then there exists a rule in **R** of the form

   $f(x_1,\ldots,x_{i-1}, c(y_1,\ldots,y_n), x_{i+1},\ldots,x_m) \text{—>}$

   $t\,[\, f_{j1}(x_1,\ldots, x_{i-1}, y_{j1}, x_{i+1},\ldots, x_m) ,\ldots, f_{jk}(x_1,\ldots, x_{i-1}, y_{jk}, x_{i+1},\ldots, x_m)\,]$

   where

   the f_{jq} are inductively defined functions over C_0 in the i–th argument of arity

   $s_1 \ldots s_{i-1} s_{jq}' s_{i+1} \ldots s_m \text{—>} s_{jq}''$,

   the x_p (resp. y_j) are pairwise distinct variables of sort s_p (resp. s_j') and

   t is a term of sort s.

2. If $n \geq 1$ and $\{ j \in \{1,\ldots n\} \mid s_j' \in S_0 \} = \emptyset$

   then there is a rule in **R** of the form

   $f(x_1,\ldots x_{i-1}, c(y_1,\ldots,y_n), x_{i+1},\ldots, x_m) \text{—>} t$

   where the x_p (resp. y_j) are pairwise distinct variables of sort s_p (resp. s_j') and t a term of sort s.

3. If $n = 0$ then there is a rule in **R** of the form

   $f(x_1,\ldots, x_{i-1}, c , x_{i+1},\ldots, x_m) \text{—>} t$

   with pairwise distinct variables x_p of sort s_p and a term t of sort s.

THEOREM

Let **R** be a regular canonical term rewriting system such that all left-hand sides are of the form

$$f(x_1,\ldots,x_{i-1},c(y_1,\ldots,y_n),x_{i+1},\ldots,x_m)$$

with some $f \in D$, $i \in \{1,\ldots,m\}$, $c \in C$, $n \geq 0$, $m \geq 1$ and pairwise distinct variables x_p and y_j.

Let C_0 be a non-empty subset of C.
Let $f \in D$, $f : s_1 \ldots s_m \text{—>} s$, $m \geq 1$, be a function inductively defined over C_0 in the i-th argument.

Then for any constructor term $c \in T(C_0,X)$ of sort s_i there is a narrowing derivation

$$f(x_1,\ldots,x_m) \;\; N\text{—}[x_i/c]\text{—>} \; t ,$$

with some term $t \in T(F,X)$.

INTERPRETATION

- For functions inductively defined over some set of constructors C we can determine a priori the substitutions that will be generated by the narrowing algorithm.

- Essentially the narrowing algorithm generates the whole constructor term algebra T(C,X) and is therefore a variant of the trivial generate-and-test algorithm.

- The enumeration as it is done in the NARROWER is not direct. Therefore the same narrowing substitution may be generated in many different ways.

A UNIFICATION ALGORITHM FOR CONFLUENT THEORIES

Steffen Hölldobler
Universität der Bundeswehr München
Werner-Heisenberg-Weg 39
D-8014 Neubiberg

INFINITE
f → c(f)

$\langle x=c(x)\rangle_{\mathrm{INFINITE}}$

{y←f} is a solution !

UNIFICATION PROBLEM:

$\langle s=t\rangle_R$

Solution

$\exists\sigma\colon\ \exists r\colon\ \sigma s \longrightarrow^{*} r \wedge \sigma t \longrightarrow^{*} r$

- Decidability Problem
- Existence Problem
- Enumeration Problem

A UNIFICATION ALGORITHM FOR CONFLUENT THEORIES

Steffen Hölldobler

Universität der Bundeswehr München
Werner-Heisenberg-Weg 39
D-8014 Neubiberg

EQUATIONS

trivial equation: $=\{t\}$

non-trivial equation: $=\{s,t\}$

$=\{x,y) \quad =\{a,b\}$

$\{x\leftarrow a, y\leftarrow b\}$

$\{x\leftarrow b, y\leftarrow a\}$

CONFLUENT THEORIES

$$EP_R = \{\ l=r\leftarrow \mid l\rightarrow r\in R\ \} \cup \{(r),(t),(s)\}$$

Theorem:

σ *is a solution for* $\langle s=t\rangle_R$ ***iff*** σ ***is*** *a correct answer substitution for* $EP_R\cup\{\leftarrow s=t\}$

TERM DECOMPOSITION

$$\leftarrow D \cup \{f(s_1,\dots,s_n) = f(t_1,\dots,t_n)\} \to_d \leftarrow D \cup \{s_i = t_i \mid 1 \le i \le n\}$$

$\to \qquad \to$

VARIABLE ELIMINATION

If $x \notin Var(t)$ then $\leftarrow D \cup \{x = t\} \to_v \leftarrow \{x \leftarrow t\}D$

REMOVAL OF TRIVIAL EQUATIONS

$$\leftarrow D \cup \{t = t\} \to_t \leftarrow D$$

LAZY PARAMODULATION

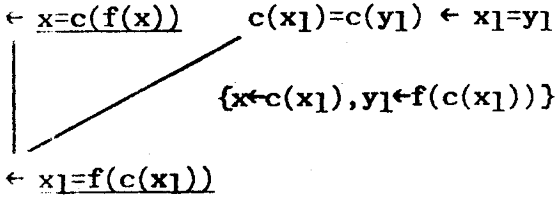
If $f(t_1,\dots,t_n) \to t_{n+1} \in R$ then

$$\leftarrow D \cup \{f(s_1,\dots,s_n) = s_{n+1}\} \to_p \leftarrow D \cup \{s_i = t_i \mid 1 \le i \le n+1\}$$

R
f(x) → b

6

$$\begin{tikzcd}[row sep=small]
\leftarrow \underline{h(f(a))=h(b)} \arrow[d, "d"] \\
\leftarrow \underline{f(a)=b} \arrow[d, "p"] \\
\leftarrow x=a,\underline{b=b} \arrow[d, "t"] \\
\leftarrow \underline{x=a} \arrow[d, "v", "\{x\leftarrow a\}"'] \\
\square
\end{tikzcd}$$



FAILED OCCUR CHECK 1

If $x \in Var(h(t_1,\ldots,t_n))$ then

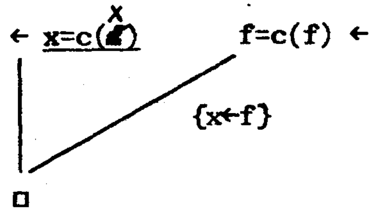
$\leftarrow D \cup \{x=h(t_1,\ldots,t_n)\}$

$\rightarrow_{f1}$ $\leftarrow \{x \leftarrow h(x_1,\ldots,x_n)\}(D \cup \{x_i=t_i \mid 1 \leq i \leq n\})$

$$\begin{tikzcd}
\leftarrow \underline{x=c(f(x))} \arrow[d, "f1\quad \{x\leftarrow c(x_1)\}"] \\
\leftarrow \underline{x_1=f(c(x_1))}
\end{tikzcd}$$

INFINITE

$f \rightarrow c(f)$



FAILED OCCUR CHECK 2

If $x \in Var(h(t_1,...,t_n))$ and $s \rightarrow h(s_1,...,s_n) \in R$

then

$\leftarrow D \cup \{x=h(t_1,...,t_n)\}$

$\rightarrow_{f2} \; \leftarrow \{x \leftarrow s\}(D \cup \{s_i=t_i \mid 1 \leq i \leq n\})$

$$\begin{tikzcd}
\leftarrow \underline{x=c(x)} \arrow[d, "f2\ \{x \leftarrow f\}"] \\
\leftarrow \underline{f=f} \arrow[d, "t"] \\
\square
\end{tikzcd}$$

Theorem:

If there exists a refutation of $EP_R \cup \{\leftarrow D\}$ *wrt the resolution rule and computed answer substitution* θ*, then there exists a refutation of* $R \cup \{\leftarrow D\}$ *wrt RULES. Furthermore, if* σ *is the computed answer substitution of the refutation of* $R \cup \{\leftarrow D\}$ *wrt RULES, then* θ *is an R-instance of* σ*.*

__ALGORITHM 1:__

Find all computed answer substitutions for refutations of $R \cup \{\leftarrow s=t\}$ wrt RULES

IMPROVING THE ALGORITHM

Simplification of a goal clause:

apply $\rightarrow_c$, $\rightarrow_v$, $\rightarrow_t$ as long as possible

INTEGERS

```
int(x) → x:int(s(x))                    (i)
first(0,y) → []                         (f1)
first(s(x),y:z) → y:first(x,z)          (f2)
```

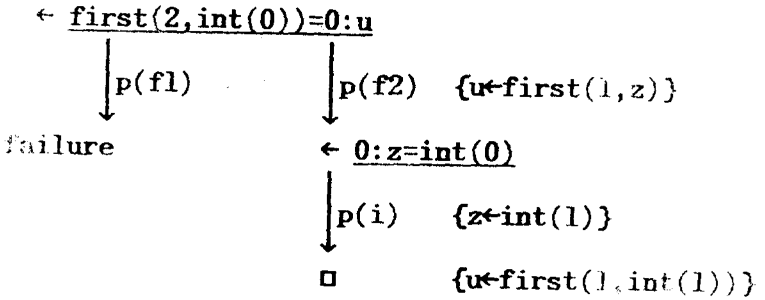
```
← first(2,int(0))=0:u
│
│ p(f2)
↓
← s(s(0))=s(x),int(0)=y:z,0:u=y:first(x,z)
│
│ d
↓
← s(0)=x,int(0)=y:z,0:u=y:first(x,z)
│
│ v {x←s(0)}
↓
← int(0)=y:z,0:u=y:first(s(0),z)
│
│ d
↓
← int(0)=y:z,0=y,u=first(s(0),z)
│
│ v {y←0}
↓
← int(0)=0:z,u=first(s(0),z)
│
│ v {u←first(s(0),z)}
↓
← int(0)=0:z
```

```
← first(2,int(0))=0:u
│
│ p(f1)
↓
← s(s(0))=0,int(0)=y,0:u=[]

failure
```

S-DERIVATION, S-REFUTATION

ALGORITHM 2:

Find all computed **answer** substitutions for s-refutations of R∪{←s=t} wrt $\{\rightarrow_f, \rightarrow_p, \rightarrow_{f1}, \rightarrow_{f2}\}$



PROCEDURE INVOCATION:

If $Q(s_1,..,s_n) \leftarrow C'$ is a new variant of a program clause then

$\leftarrow C \cup \{Q(t_1,\ldots,t_n)\} \rightarrow_i \leftarrow C \cup C' \cup \{s_i = t_i | 1 \le i$

Lazy E-Unification. A Method to Delay Alternative Solutions

Hans-Jürgen Bürckert, FB Informatik, Universität Kaiserslautern

Abstract

One of the most unsuitable properties of E-unification is the existence of more than one most general E-unifier. We want to present a general method to delay these alternative solutions in the context of Logic Programming, since there it will be most problematic. The idea is to be lazy in unification, that is to unify at most those parts of a unification problem that will not split up the solution space [Ohlbach], [Bürckert]. The partial unifier will be used for resolution and the remaining part of the unification problem will be kept in memory. If the empty clause is derived, the collected residues of the unification problems will be totally E-unified. If they are not E-unifiable, backtracking takes place.

LAZY E-Unification: A Method to Delay Alternative Solutions

Hans-Jürgen Bürckert

E-Unification

Problem:

more than one most general unifier

Solution:

Lazy unification

=

unification of a unitary part

Example: associative function
A: $x(y\,z) = (x\,y)\,z$

$$\langle p(h(x\,y),\ x\cdot a,\ y\cdot b) = p(h(z\,b),\ a\cdot x,\ b\cdot v)\rangle_A$$

$$\downarrow$$

$$\langle h(x\,y) = h(z\,b),\ x\cdot a = a\cdot x,\ y\cdot b = b\cdot v\rangle_A$$

$$\downarrow$$

$$\langle x = z,\ y = b,\ z\cdot a = a\cdot z,\ b\cdot b = b\cdot v\rangle_A$$

$$\downarrow$$

$$\langle \underline{x = z},\ \underline{y = b},\ \underline{z\cdot a = a\cdot z},\ \underline{v = b}\rangle_A$$

partial unifier: $\{x \leftarrow z,\ y \leftarrow b,\ v \leftarrow b\}$

residue: $\langle z\cdot a = a\cdot z\rangle_A$

Logic Program

set of definite clauses

$h \Leftarrow b_1 \,\&\, \ldots \,\&\, b_n \quad (n \geq 0)$

Query

goal clause

$\Leftarrow b_1 \,\&\, \ldots \,\&\, b_n \quad (n \geq 1)$

strategy

= ordering on goal reductions

Examples:

- standard strategy (SLD-Resolution)

$$\to \ldots \xrightarrow[u]{*} G_{n-1} \xrightarrow[R]{} G_n \xrightarrow[u]{*} \underbrace{G_{n+1}}_{\substack{\text{solved}\\ u\text{-conditions}}} \xrightarrow[R]{} \ldots \to$$

lazy strategies

- totally lazy strategy (constraint res.)

$$\xrightarrow[R]{} \ldots \xrightarrow[R]{} G' \xrightarrow[R]{} G'' \ldots G_n' \xrightarrow[u]{*} G \quad \text{solved goal}$$

lazy strategies ⇝ lazy E-unification

Goal Reduction Rules

(R) $p(t_1 \ldots t_n) \,\&\, G \rightarrow \overset{\text{unification cond.}}{\boxed{s_1 = t_1 \,\&\ldots\&\, s_n = t_n}} \,\&\, B \,\&\, G$

if $p(s_1 \ldots s_n) \Leftarrow B$ is a variant of a clause

$$1) \begin{cases} (T)\; x = x \,\&\, G \rightarrow G \\ (O)\; t = x \,\&\, G \rightarrow x = t \,\&\, G \\ (B)\; x = t \,\&\, G \rightarrow x = t \,\&\, \{x \leftarrow t\}G \\ \qquad \text{if } x \notin Var(t) \text{ and } x \in Var(G) \\ (D)\; f(s_1, \ldots s_n) = f(t_1 \ldots t_n) \,\&\, G \rightarrow s_1 = t_1 \,\&\ldots\&\, s_n = t_n \,\&\, G \end{cases}$$

Goal reduction $\quad G \rightarrow G'$

refutation $\quad G_0 \rightarrow \ldots \rightarrow G_n$

G_n in solved form

$x_1 = t_1 \,\&\, \ldots \,\&\, x_m = t_m$

- $\{x \leftarrow t$ $\ldots t\}$

program with equality

- set of definite clauses P (no equality head)
- set of equality facts E

Replace "$\underset{u}{\overset{*}{\rightarrow}}$" by E-Unification in the standard strategy

$\Rightarrow$ E-SLD-Resolution sound and complete

<u>However,</u>

more than one

Lazy E-unification

- take any lazy strategy
- apply (D) only to E-decomposable equations

  i.e., iff

  $\mathcal{U}_E(f(s_1..s_n)=f(t_1..t_n))=\mathcal{U}_E(s_1=t_1,\dots,s_n=t_n)$

- add some E-unification rule for unitary equality subgoals

  i.e., iff

  $|\mu\mathcal{U}_E(\Gamma)|=1$

- failure termination rules

  for non-E-unifiable goals

Lazy E-refutation

$G_0 \to \dots \to G_n$

- G_n E-unifiable equality goal
- (R) + lazy-E-unification "$\to$"

is sound and complete

E-answer substitutions

= E-unifiers of G_n

WARREN machine

(Abstract Prolog Machine)

- stacks for environment, backtracking, etc
- registers for arguments of calling goal
- instructions for unification, backtracking, etc

$h \Leftarrow b_1, \dots, b_n$

is compiled to

- allocate environment
- unify head arguments with arg. reg.
- initialize arg. reg. with arguments of b_1
- call b_1

⋮

- initialize arg. reg. with b_n
- call b_n

Warren Machine with E-unification

- additional stack for lazy unification residues
- lazy-unification instructions
- E-unification procedure
  - for unitary E-unifications
  - for E-unification of the residues

"Symbolic Computation and Architecture Research in the ISA project at MCC"

Hassan Ait-Kaci
Roger Nasr (Speaker)

Abstract

The I. S. A. group in MCC's AI Program is in the third phase of a research project developping language and architecture support technologies for symbolic computation. The target computation model embodies the integration of logic programming, functional programming, and typing ideas (mostly concerned with partially-ordered type objects and inheritance among them, but not excluding polymorphic types and related issues).

Our work is essentially based on generalizing the notion of Unification from what it is on first-order terms to various similar syntactic algebraic structures. Depending on the richness of these syntactic algebras and their corresponding special-purpose "unification" operation, this method provides the key to inject more "semantics" into syntax for a large class of logic, algebraic, and functional computation.

The talk will summarize the research completed to date and will give a glimpse into plans for future work.

LIFE,

A LOGIC OF INHERITANCE FUNCTIONS AND EQUATIONS

Hassan Ait-Kaci
Roger Nasr

MCC AI-Program

Austin, Texas

References

Aït-Kaci, H. : A Lattice-Theoretic Approach to computation Based on a Calculus of Partially-Ordered Type Structures. Ph.D. Thesis, C.I.S., University of Penn. Philadelphia, 1984

Aït-Kaci, H. : An Algebraic Semantics Approach to the Effective Resolution of Type Equations. To appear in J. of T.C.S. 45. 1987

Aït-Kaci, H. and R. Nasr : LOGIN: A LOGIC PROGRAMMING LANGUAGE WITH BUILT-IN INHERITANCE J. of Logic Programming 1986 : 3 : 185-215

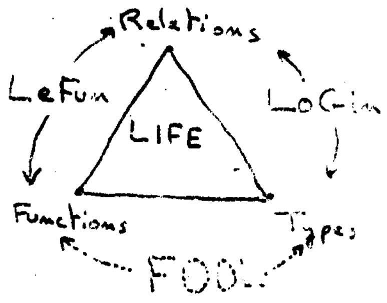
Aït-Kaci, H and R. Nasr : Residuation: A Paradigm for Integrating Logic & Functional Programming. MCC Tech. Rep. AI-359-86

Intelligent Systems Architecture project

Theme : Develop a Symbolic Computation model,
a representative language, and
a Supporting Architecture.

odology : Integrate Logic and Functional Programming Concepts with 'type' ideas into a new Programming paradigm.

[illegible] - Gather experience through partial integrations



- Full Integration + Language + Abstract Machine

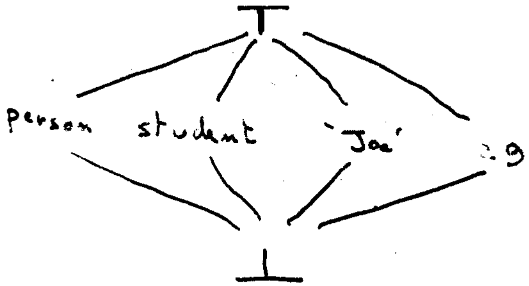
②

FROM F.O.T's to ψ-Terms

```
person ('Joe', 29, student).
```

```
person ( name => 'Joe',
         age => 29,
         occupation => student).
```

where :

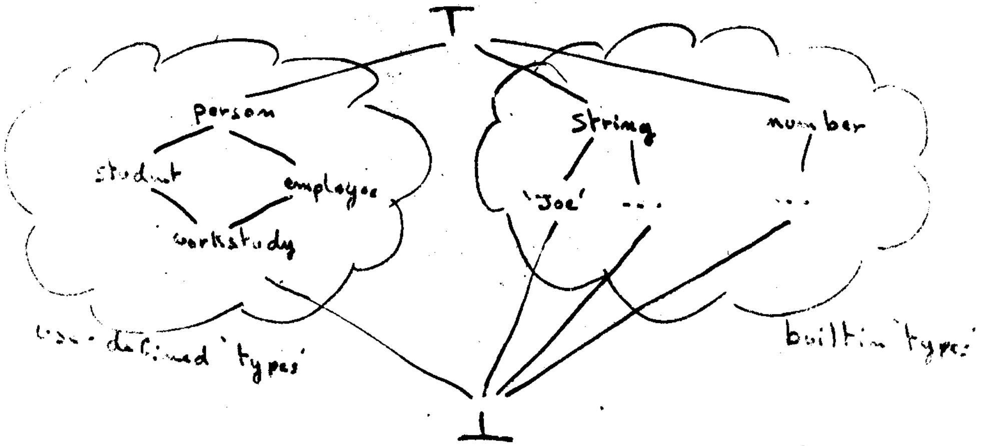


③

More on ψ-Terms

```
ℙ: student ( name => ( first => 'Joe',
                       last => 𝕃: string),
             father => employee ( name => ( last => 𝕃),
                                  son => ℙ              ))
```

where :



Ψ-Term Unification

(1)

- Extend the quasi-ordering on types
  t a " " Ψ-Terms

(Term Subsumption:

person (name ⇒ string)

student (name ⇒ 'Joe',
age ⇒ 25).)

- Given terms t_1 & t_2, their unification produces $t_1 \wedge t_2$ such that
$t_1 \wedge t_2$: $\lceil \underline{t_1} \cap \underline{t_2} \rceil$

. The algorithm is an adaptation of the UNION/FIND procedure.

- At the heart of the algorithm we use the g.l.b. on the type symbols in their ordering ...

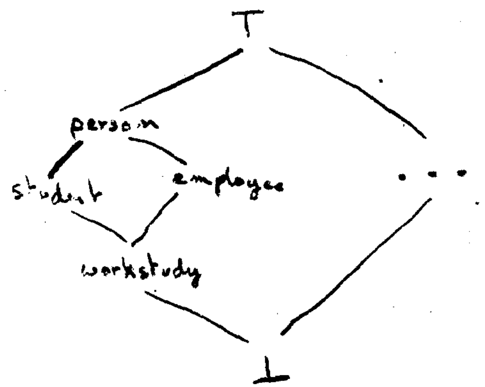
Ψ-Term Unification
(example)

(1)

t_1 is P: student (age ⇒ 19,
father ⇒ person (son ⇒ P))

t_2 is employee (salary ⇒ number,
father ⇒ employee (age ⇒ 55))

$t_1 \wedge t_2$ is then P: workstudy (age ⇒ 19
father ⇒ employee (son ⇒ P,
age ⇒ 55
salary ⇒ number).



Generalized ε-Terms

< Tag , Excl.Classes , Untagged >

Syntax: $X \setminus\setminus \{e_1, \dots, e_n\} : \{t_1; \dots; t_m\}$

Tag : identity/address of the term (also term's coreference class)

Excl. Classes: Mutual Exclusion Classes to which this term (Tag) belongs: i.e. Unification of this term with another one belonging to (at least) one of those classes should fail immediately ...

Untagged : The general case is that this is a disjunctive type ($\{t_1; \dots; t_m\}$ reads "of type t_1 or ... or t_m") with each of the $t_1 \dots t_m$ being a pair:

< principal-type-symbol , list-of-attributes >

where the 'list-of-attributes' elements are of the form 'label ⇒ ε-Term'

Generalized ε-Terms (Cont'd)

6.1

e.g.

person (pet ⇒ { dog ; cat } ,
sex ⇒ { male; female })

semantically equivalent to:

{ person (pet ⇒ dog,
sex ⇒ male)
; person (pet ⇒ dog,
sex ⇒ female)
; person (pet ⇒ cat,
sex ⇒ male)
; person (pet ⇒ cat,
sex ⇒ female) }

- This kind of normalization is not needed though at the operational level ... More on that later

Mutual Exclusions Classes

(simple Example)

$$
\begin{aligned}
\backslash\backslash e : \text{person}\,(\;&\text{father} \Rightarrow X \backslash\backslash e : \text{person},\\
&\text{mother} \Rightarrow \backslash\backslash e : \text{person},\\
&\text{provider} \Rightarrow X\;)
\end{aligned}
$$

will not unify

with $p : (\text{father} \Rightarrow p)$.

or with $(\text{mother} \Rightarrow X,$
$\text{provider} \Rightarrow X)$.

or with $(\text{mother} \Rightarrow X,$
$\text{Father} \Rightarrow X)$.

etc...

E-Term Unification

simple case : atomic disjuncts in the disjunctive types:

simply: $\{t_1; \ldots ; t_n\} \underset{\epsilon}{\wedge} \{s_1; \ldots ; s_m\} = \Big\lceil \bigcup_{\substack{i=1\to n\\ j=1\to m}} \underline{t_i} \cap \underline{s_j}$

More Complex case : Non-Atomic disjuncts :

$$\{NAt_1; \ldots ; NAt_n\} \underset{\epsilon}{\wedge} \{NAs_1; \ldots ; NAs_m\} =$$

$$\left\{\begin{array}{l}
NAt_1 \underset{\psi}{\wedge} NAs_1; \ \ldots\ ; \ NAt_1 \underset{\psi}{\wedge} NAs_m;\\
\ldots\ ;\\
NAt_n \underset{\psi}{\wedge} NAs_1; \ldots\ ; \ NAt_n \underset{\psi}{\wedge} NAs_m
\end{array}\right\}$$

LogIn
(Logic + Inheritance)

•

- Main Computational model is relational Horn Logic, but:
  - generalized ∈-Terms replace F.O.T.'s
  - in SLD-Resolution, LOGIN-Unification (adaptation of ∈-Term Unification) replaces F.O.T. Unification.

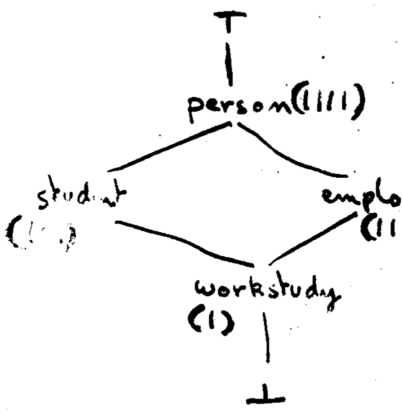
- Tag scope is the horn clause or the goal to be resolved (Dynamically that becomes the resolvent)

.. Encoding Mechanism is introduced for a much higher (constant time) performance of the g.l.b. operation (at the price of space, of course!).

Encoding Mechanism

•

- Binary signature encoding:



	person	student	employee	workstudy
person	r			
student	d	r		
employee	d		r	
workstudy	c	d	d	r
	(1111)	(101)	(11)	(1)

therefore:

$$\begin{aligned} \text{student} \wedge \text{employee} &= \text{decode}(101\ .\text{AND}.\ 11) \\ &= \text{decode}(1) \\ &= \text{workstudy} \end{aligned}$$

- Decoding is not needed until results need to be 'printed'. Intermediate results are never decoded...

LOGIN Representation of ∈-Terms

- atomic disjuncts: Such terms are simply represented by their code: i.e.

$$code(\{t_1;\ldots;t_n\}) = code(t_1) .OR. \ldots .OR. code(t_n)$$

- non-atomic disjuncts:
  - First the case of a singleton non-atomic term: such term is represented by its familiar skeleton but with types replaced by their codes. Such terms have a 'so-called' screening-code that corresponds to the code of their principal type symbol...
  - Next the case of a disjunction of such non-atomic terms: such disjunctive terms are represented by a pair consisting of
    - the disjunction's 'screening-code' and
    - a list of the representations of the disjuncts as described above (This will be refered to as the disjunctive continuation...)

LOGIN-Unification

Same as in ∈-Term unification but:

- Scope of tags is the whole clause (at run-time this will be the resolvent...
- glb is replaced with 'AND'ing of the type symbol codes... (within the UNION/FIND inspired 'meet' algorithm
- Unification of non atomic disjunctions uses SLD-Resolutions OR-continuation (choice points) to 'lazily' consider the disjuncts...
- Decode the codes back into type-symb. expressions ONLY at 'result printing' time

LeFun

Residuation

- A simple integration of a Prolog-like language and a functional Prog. Lang.

- F.O.T.'s are generalized to accept, at any subterm level (including the 'root') a functional expression.

- F.O.T-Unification is modified to handle functional expressions with <u>uninstantiated</u> variables by delaying such unifications as long (and only as long) as necessary : until all variables therein are instantiated.

- Generalize this delaying mechanism to cover not only unification but other built-in preds (e.g. comparative preds, 'freeze/2' and 'dif/2' à la Prolog-II etc

Residuation

(Examples).

```
41        q(X,Y,Z) :-
                  p(X, Y, Z, Z),
                  pick(X, Y).

P1 ——   p(X, Y,  X+Y , X*Y).
P2 ——   p(X, Y,  X+Y,  (X*Y)-14).

Pick 1 — pick (3,5).
     2 — pick (2,2).
     3 — pick (4,6).
- - - - - - - - - - - - - - - - - -

   |- q(A, B, C).
   ______________
(P1 -->  p(A, B, C, C), pick(A, B).
           p(X, Y, X+Y, X*Y)
         ______________________
          pick (A, B).      (  with
CP2->                       :  A -> B ->
                            :     resid(A+B=A*B)
                            :  i.e. A & B are still
                            :  uninstantiated but
          pick(3,5).        :  have a pending resid.
          ________________
      fail ( because the resid. is released and fails
                3+5 ≠ 3*5)
   backtrack chronologically to CP2 and:
          pick(2,2).
          ________________
      succeed ( because the resid. is released, succeeds
                  and is discarded ...)
      C = 4 (;) <- force backtracking

              etc ...
```

Residuation

(Examples cont'd)

•

```
sq(X) = X *X.

twice(F,X) = F(F(X)).

valid-op(twice).

p(1).

pick(lambda(X,X)).

q(Val) :-  G = F(X),
           Val = G(1),
           valid-op(F),
           pick(X),
           p(sq(Val)).


?- q(Ans).
```

LIFE-Terms

(roughly ...)

•

< Tag, Excl-Classes, Untagged-LIFE-TERM >

__where__: - Tag & Excl-Classes are the familiar ones

- Untagged-LIFE-TERM is,
  - either an untagged-E-Term
  - or a functional expression

__e.g.__

```
block ( volume => V,
        density => D,
        weight => weight(V,D)).
```

LIFE-Unification

- LIFE-Unification integrates:
    - E-Term Unification, and
    - LeFun (Residuation) Unification

  with the following clarification...

- Functional LIFE expressions are ready to act as Unificands only when Tags therein are coerced to minimal types (Values; typesymbols that are one step above ⊥...) otherwise unifications that involve them Residuate...

LIFE-Grammars

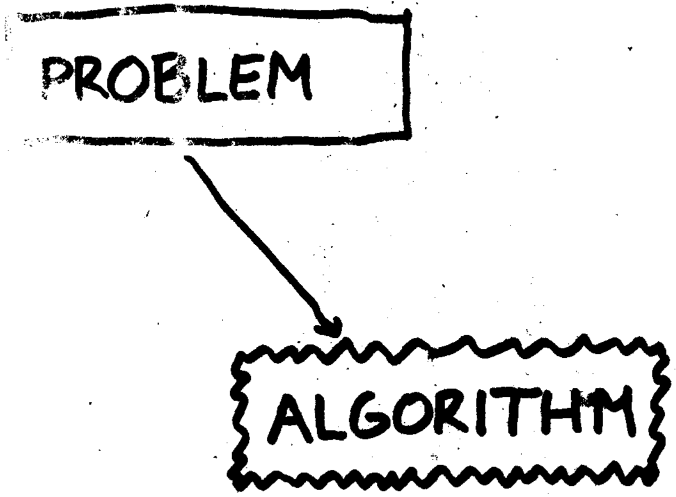
- They are to LIFE as D.C.G.'s are to Prolog
- Example: where m → binary number
  ℓ → binary word
  b → binary digit

```
m(val => V) -> ℓ(val => V;
                  scale => 0).
m(val => V1+V2) -> ℓ(val => V1,
                      scale => 0), ['.'], ℓ(val => V2,
                                            length => L,
                                            scale => -L).

ℓ(val => V,
  length => 1,
  scale => S) -> b(val => V,
                   scale => S).
ℓ(val => V1+V2,
  length => L+1,
  scale => S ) -> ℓ(val => V1,
                    length => L,
                    scale => S+1),    b(val => V2,
                                        scale => S).

b(val => 0) -> ['0'].
b(val => 2^S,
  scale => S) -> ['1'].
```

```
?- life-phrase( m (val => X), [1,0,1,'.',0,1]).
```

reads parse the binary number '101.01' using the syntax 'm' and return in X the value of that number

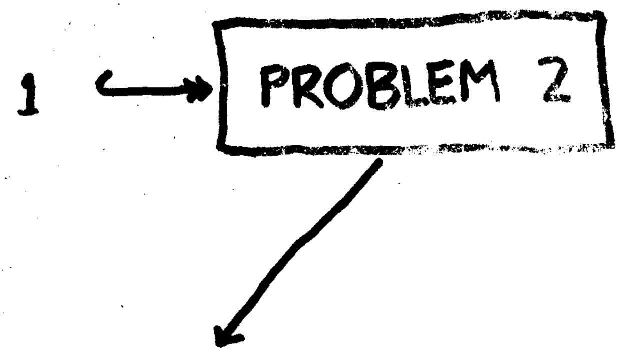
HOMOMORPHISMS + PROBLEM SOLVING



PROBLEM

ALGORITHM

PETER RUFFHEAD



MUST PROVE SOUNDNESS
AND COMPLETENESS

i SHOW THAT EITHER THE PROBLEM IS TRIVIAL OR ELSE IT IS INTUITIVELY EQUIVALENT TO ANOTHER PROBLEM.

- SEEMS TO BE CORRECT
- IS AS WEAK AS POSSIBLE
- GIVES A STRONG RESULT

ii SHOW THAT ARE NO INFINITE DECREASING SEQUENCES OF EQUIVALENT PROBLEMS.

iii SHOW THAT THE ALGORITHM PRODUES A DECREASING SEQUENCE OF EQUIVALENT PROBLEMS

[illegible]

RD PROBLEM

[illegible]TRODUCE UNIVERSAL ALGEBRA

[illegible] A SIGNATURE OF FUNCTION SYMBOLS

[illegible]EBRAS (X, σ)

[illegible]MORPHISMS

$[illegible](\sigma_\omega(t_1, \dots, t_n)) =$

$\sigma'_\omega(\theta(t_1) [illegible]$

[illegible] Σ ALGEBRAS

[illegible] INTERPRETATION

[illegible]FF'S THEOREM

[illegible]NTACTIC INTERPR[illegible]

WORD PROBLEM

$E \vdash t_1 = t_2$

$E \models t_1 = t_2$

$t_1, t_2 \in VT_\Sigma$

(t_1, t_2) <u>HOLDS</u> IN (X, σ) IFF

$\forall\ \theta : (VT_\Sigma, V\Delta_\Sigma) \rightarrow (X, \sigma)$

$\theta(t_1) = \theta(t_2)$

- INJECTIVE HOMOMORPHISM METHOD

$$\begin{tikzcd}
 & X \arrow[dd, hook, "i"] \\
VT_{\Sigma} \arrow[ur, "\theta"] & \\
\rotatebox{90}{$\in$} \; t_1, t_2 & X'
\end{tikzcd}$$

IF (t_1, t_2) HOLDS IN (X', σ'). THEN (t_1, t_2) HOLDS IN (X, σ)

IF DISTRIBUTIVITY HOLDS FOR THE REALS ($\mathbb{R}$), IT ALSO HOLDS FOR THE INTEGERS ($\mathbb{Z}$)

SURJECTIVE HOMOMORPHISM METHOD

$\mathbb{Z} \to \mathbb{Z}_2$

- IMAGE METHOD

$$\begin{tikzcd}
 & X \\
VT_{\Sigma} \arrow[ur, "\theta"] & \\
\rotatebox{90}{$\in$} \; t_1, t_2 & X'
\end{tikzcd}$$

IF (t_1, t_2) HOLDS IN X', THEN $(\theta(t_1), \theta(t_2))$ HOLDS IN X'.

SUBSTITUTIVITY

HOMOMORPHISMS TRANSFORM LOCAL RESULTS

Unification in Varieties of Idempotent Semigroups.

F. Baader

We have classified all varieties of idempotent semigroups with respect to the unification types of their defining sets of identities. Almost all of them - with the exception of eight theories which are finitary unifying - are of unification type zero.

This talk gives a short survey of two methods used in the proof of these results.

E set of identities.

$=_E$ equality of terms induced by E.

V_0 finite set of variables.

$\theta_1 =_E \theta_2 \langle V_0 \rangle$ $x\theta_1 =_E x\theta_2$ for all $x \in V_0$.

$\theta_1 \le_E \theta_2 \langle V_0 \rangle$ $\exists \lambda$ $\theta_1 =_E \theta_2 \circ \lambda$.

$\mu U_E(s,t)$ set of most general E-unifiers for s,t.

Varieties of Idempotent Semigroups

set of identities $U=V$ where $U,V \in X^+$. X is a countable set of variables.

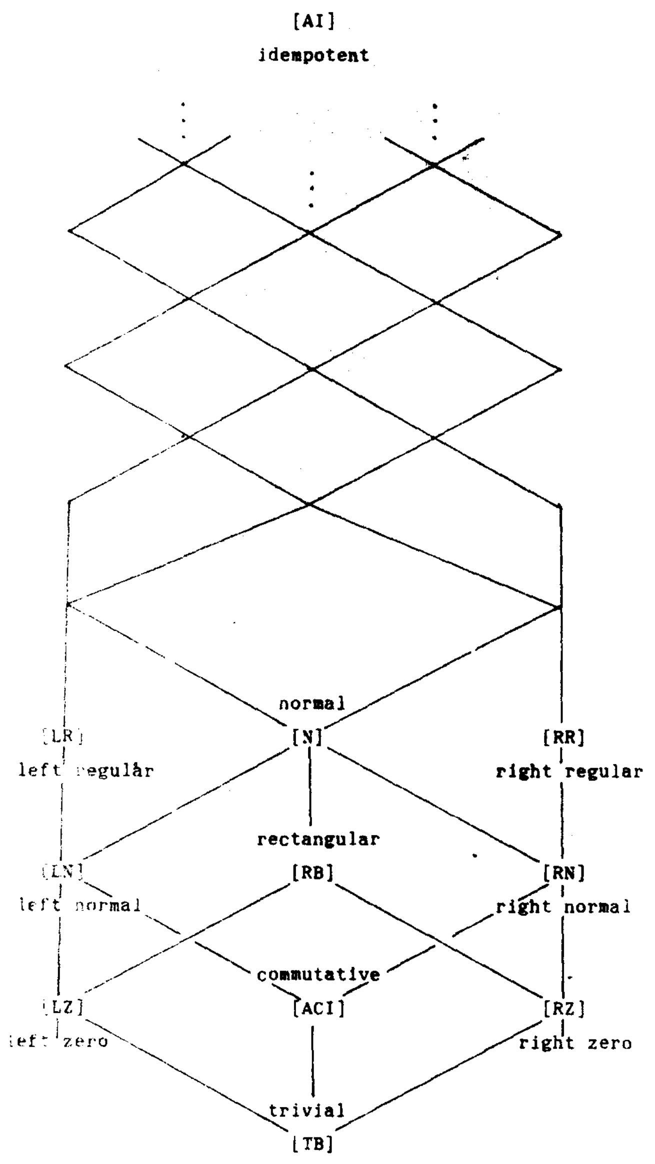
class of all idempotent semigroups satisfying each identity of E, i.e. the variety of idempotent semigroups defined by E.

Gerhard, Fennemore, Birjukov :

(1) Any variety of idempotent semigroups may be defined by exactly one identity.

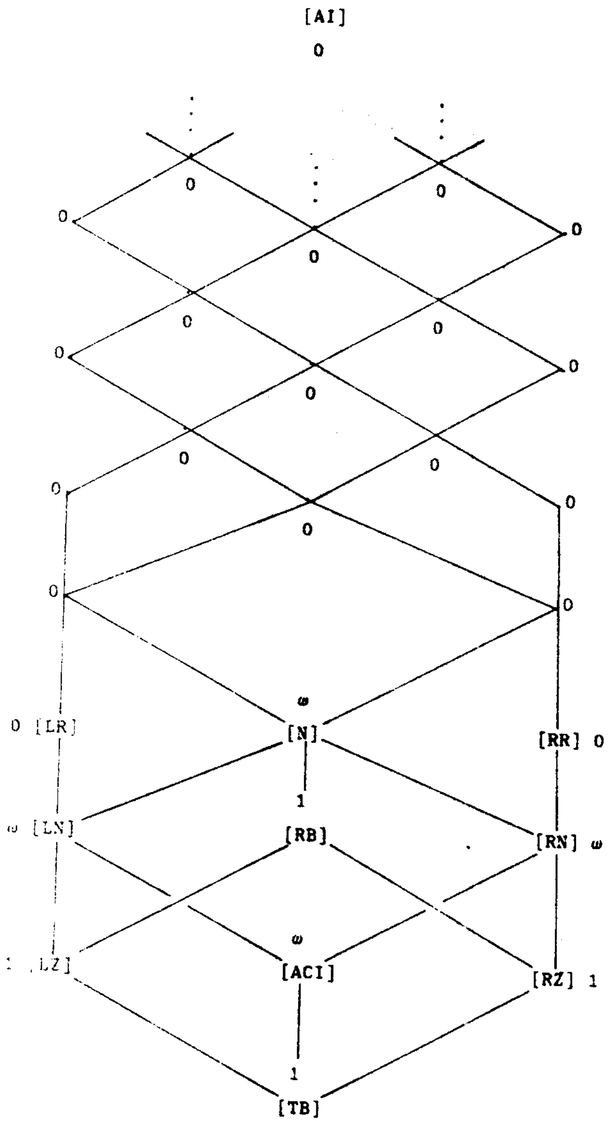
(2) Determination of the lattice $\mathcal{B}$ of all varieties of idempotent semigroups.

FIGURE 3.1



The lattice B of all varieties of bands.

FIGURE 7.1



The unification types of all varieties of bands.

1 ≙ unitary, ω ≙ finitary, 0 ≙ type zerc

.2 Unitary and Finitary Theories

The eight varieties defined by ~~the~~ the unitary and finitary theories constitute a sublattice of $\mathcal{B}$, i.e. the boolean lattice ~~of~~ generated by the three atoms of $\mathcal{B}$.

- We have treated the atoms [LZ], [RZ], [ACI] directly
- The other theories are "joins of the atoms".

We have to consider the following situation :

$[E_1] \vee [E_2] = [E]$, i.e. $=_{E_1} \cap =_{E_2} = =_E$.

Provided E_1, E_2 are finitary, is E finitary too ?

Sufficient condition on which the answer is yes :

Let V_0 be a finite set of variables; E, E_1, E_2 as above.

Condition:

For all substitutions μ there exist finite sets substitutions $\Sigma_1(\mu)$, $\Sigma_2(\mu)$ such that

(1) $\forall \mu' \in \Sigma_i(\mu) : \quad \mu' \leq_{E_i} \mu \langle V_0 \rangle \qquad (i = 1,2)$

(2) $\forall \sigma, \lambda$ such that $\sigma =_{E_i} \mu \circ \lambda \langle V_0 \rangle$ there is $\mu' \in \Sigma_i(\mu)$ and λ' with $\sigma =_E \mu' \circ \lambda' \langle V_0 \rangle$

Proposition

Let E_1, E_2 be finitary and $[E] = [E_1] \vee [E_2]$.
If for all finite sets of variables V_0 the condition holds then E is finitary too.

Corollary

If E_1, E_2 are unitary and the sets $\Sigma_i(\mu)$ are singletons , then E is also unitary.

Theories of Type Zero

The elements of $\mathcal{B} \setminus \mathcal{U}$ satisfy

$[LR] \subseteq [E] \subseteq [AI]$ or

$[RR] \subseteq [E] \subseteq [AI]$

Let $[LR] \subseteq [E] \subseteq [AI]$, i.e. $=_{AI} \subseteq =_E \subseteq =_{LR}$

Thus $u =_{AI} v$ is a sufficient condition for $u =_E v$,

$u =_{LR} v$ is a necessary condition for $u =_E v$.

Sketch of the Proof

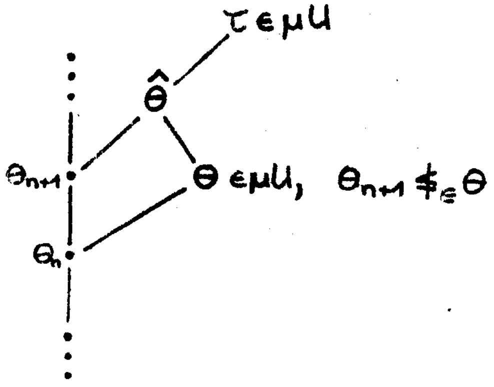
We construct a set $\{\Theta_n ; n \geq 1\}$ of unifiers such that

(1) $\Theta_n \leq_E \Theta_{n+1}$ but Θ_{n+1} [illegible] Θ_n

- Θ_n introduces $2n$ variables
- Any substitution Θ such that $\Theta_n \leq_E \Theta$ introduces at least $2n$ variables.

(2) For any Θ such that $\Theta_n \leq_E \Theta$ we construct a unifier $\hat{\Theta}$: $\Theta_{n+1} \leq_E \hat{\Theta}$
$\Theta \leq_E \hat{\Theta}$

Assume $\mu\mathcal{U}_E$ exists. Then



$\tau \in \mu\mathcal{U}$
$\hat{\Theta}$
Θ_{n+1}
$\Theta \in \mu\mathcal{U}$, $\Theta_{n+1} \not\leq_E \Theta$
Θ_n

Now $\Theta \leq_E \tau$ but $\tau \not\leq_E \Theta$.

Applications of Boolean Unification in Logic Programming

Helmut Simonis
ECRC
Arabellastr. 17
8000 München 81

Abstract

The boolean unification algorithm of Büttner and Simonis has been incorporated into a PROLOG system. Since boolean unification forms a unitary theory, only one mgu has to be computed. The system has been used on various applications in the hardware domain :

- Simulation
- Verification
- Simplification
- Synthesis
- Debugging

of digital hardware. Examples include the proof of correctness for a complete 16 bit microprocessor described at the logic gate level.

ECRC

Application of Boolean Unification in Logic Programming

H. Simonis

**European Computer-Industry
Research Centre GmbH
Arabellastr. 17
D-8000 Muenchen 81
West Germany**

Boolean Unification

Unitary Theory (only one mgu)

Possibly exponential growth of terms

Normal form of terms as sum of products (XOR, AND)

Stepwise elimination of variables

Implementation in C inside a Prolog-System

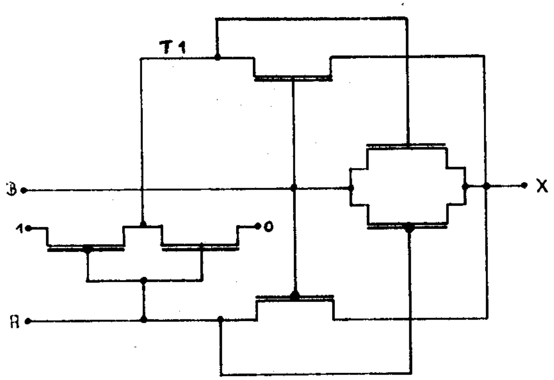
Application Areas

Digital Hardware Design

- Simulation (Executable Specifications)
- Verification (16 bit microprocessor)
- Synthesis (PAL equations from truth tables)
- Simplification (Design Rule Change: ECL --> MOS)
- Specialisation (Adder --> Increment)
- Algorithmic Debugging (Shapiro/Lloyd)

Hardware Description in Prolog

XOR-Gate as a Network of CMOS Transistors



```
xor(N,A,B,X) :-
        p_switch([1|N],1,A,T1),
        n_switch([2|N],0,A,T1),
        p_switch([3|N],B,A,X),
        n_switch([4|N],B,T1,X),
        p_switch([5|N],A,B,X),
        n_switch([6|N],T1,B,X).
```

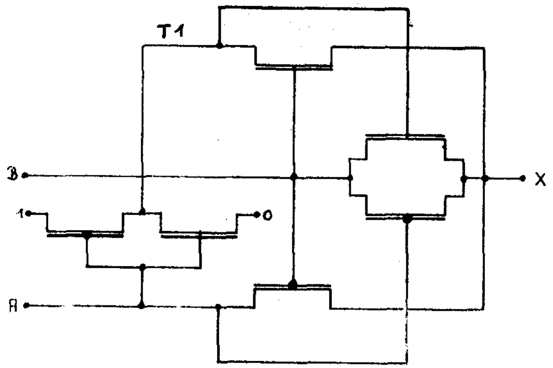
Hardware Description in Prolog

CMOS Transistors modeled as ideal switches

```
p_switch(N,D,G,S) :-
        eq(D # D & G , S # S & G).
        /*  G' => D = S  */

n_switch(N,D,G,S) :-
        eq(D & G , S & G).
        /*  G  => D = S  */
```

- Predicates Describing Components
- Wires Modeled with Shared Logical Variables
- Hierarchical Description of Modules
- Multiple Outputs
- Bidirectional Components
- Tristate Busses

Example: Verification of XOR-Gate



```
p_switch(N,D,G,S) :-
        eq(D # D & G,S # S & G).
        /*  G' => D=S  */

n_switch(N,D,G,S) :-
        eq(D & G,S & G).
        /*  G  => D=S  */
```

Values of the variables after each step

1) T1 = 1 # a # _A&a

2) T1 = 1 # a

3) X = b # _C&a # a&b

4) X = b # _C&a # a&b

5) X = a # $\overline{b}$ # _D&a&b

6) X = a # b

Conclusion

Advantages of Prolog with Boolean Unification

- Relational Form and Logical Variables allowing

  Multiple Outputs

  Bidirectional Switches

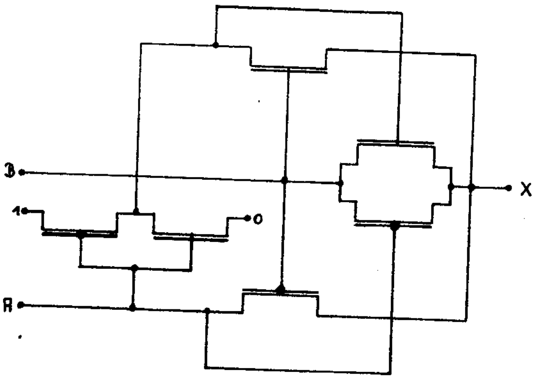
  Tristate-Busses

- Boolean Unification allowing

  Simulation

  Simplification

  Equation Solving


B
1
0
A
X

SIEMENS AG
Corporate Laboratories for Information Technology
Munich
Dr. W. Büttner

Abstract:

ON NEW UNITARY UNIFICATION THEORIES AND RELATED APPLICATIONS

In the past unification theory has closely investigated basic theories i.e. empty theory, associativity, commutativity, idempotence etc., which represent basic features of general domains. If the size of the set of most general unifiers serves as a quality measure, most of these theories behave poorly. Complex theories, modelling domains more faithfully, can be constructed from basic theories. In general, these complex theories will enherit the drawbacks of their building blocks.

There are however exceptional cases, where the building blocks mutually cancel out their unpleasant features and produce a "tamed", unitary theory. Examples of such theories of considerable practical relevance are provided by the algebra of functions: $A^m \rightarrow A$, where A is the 2-element boolean algebra or - more general - a finite chain or - generalizing boolean rings - a finite field.

Putting the corresponding unification algorithms into a Prolog system provides an extended Prolog suitable to deal with

- digital circuits, sets, formulas of propositional logic, linear algebra over GF(2) (in the boolean algebra case)
- applications of multivalued logic (in case of a chain)
- applications of linear algebra over finite fields i.e. error correcting codes, finite Fourier transformation (in case of a finite field).

$(F_n, +, \cdot, -) := (\{0,1\}^{\{0,1\}^n}, \max, \min, 1-(\cdot))$, $F_n \overset{\text{free}}{=} \langle x_1, \ldots, x_n \rangle$

Atom in F_n = product of length n of free generators or their conjugates

Every $f \in F_n$ is (unique!) sum of atoms; $x_1 \in F_2 \Rightarrow x_1 = x_1x_2 + x_1\overline{x_2}$;

$$x_1 \in F_3 \Rightarrow x_1 = x_1x_2x_3 + x_1\overline{x_2}x_3 + x_1x_2\overline{x_3} + x_1\overline{x_2}\overline{x_3}$$

$f \in F_n$, $A_n(f) :=$ set of atoms generating f

$A_2(x_1) = \{x_1x_2, x_1\overline{x}\}$

$A_n(1) = \{\text{Atoms in } F_n\}$

$F_n \overset{\text{nonfree}}{=} \langle A_n(1) \rangle$

Which functions from $A_n(1)$ to F_m extend to homomorphisms from F_n to F_m ?

- I Any function which maps the elements of $A_n(1)$ onto the elements of a partition of F_m extends to a homomorphism.

  Conversely all homomorphism from F_n to F_m arise this way (Note: At this point F_n, F_m may be replaced by arbitary boolean algebras)

  Fix r constants among the free generators of F_n generating $F_r \simeq A \subseteq F_n$.

  " " " F_m " $F_r \simeq A \subseteq F_m$.

- II Any function which satisfies I and maps $A_n(a)$ onto $A_m(a)$ for all $a \in A$ extends to a constant preserving homomorphism from F_n to F_m.

  Conversely all constant preserving homomorphism arise this way.

Given $t \in F_n$, find all homomorphisms $\zeta : F_n \to F_m$ ($m = ?$) s. th. $\zeta(t) = 0$

- t contains no constants (for $t \neq 1$ always solvable)

  Choose N minimal w. r. t. $\#(A_n(1) \setminus A_n(t)) \leq 2^N$. Then:

  Any function $\zeta : A_n(1) \to F_k (k \geq N)$ satisfying ■

  and $\zeta(A_n(t)) = 0$ extends to a homomorphism

  $\zeta : F_n \to F_k$ with $\zeta(t) = 0$.

  For k = N we obtain mgu's introducing a minimal number of ~~unifiers~~ variables if we require

  that elements in $A_N(1) \setminus A_N(t)$ be mapped injectively to ~~$A_N(1)$.~~ a partition of F_m.

- t contains constants generating a subalgebra $A \subseteq F_n$.

  $t = 0$ is not solvable iff there is $a \in A$ s. th. $A_n(a) \subseteq A_n(t)$.

  Hence, complexity of decision problem is #A.

  Assume $t = 0$ is solvable.

  choose $a_0 \in A$ s.th. $\#(A_n(a_0) \setminus A_n(t))$ is maximal

  choose N minimal s.th. $\#(A_n(a_0) \setminus A_n(t) \leq 2N$.

  Now construct unifiers as above.

A a set

A^{A^m} = set of functions f: $A^m \to A$

A^{A^m} inherits algebraic structure from algebraic structure on A (pointwise)

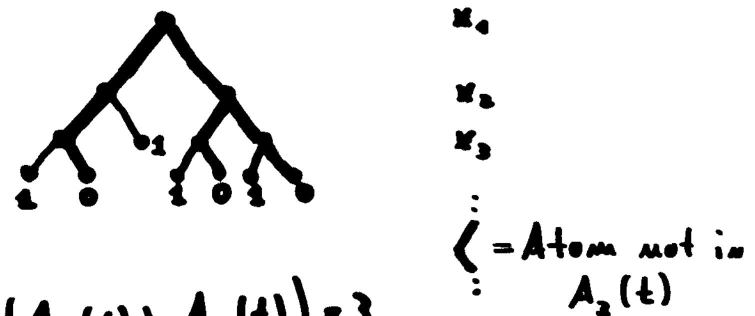
Special cases: A = 2-Element boolean algebra or
finite chain or
finite field

MAINRESULT: UNIFICATION IN A^{A^m} IS UNITARY
(A as above)

APPROACH: PROBLEM REDUCTION TO EQUATION SOLVING IN ARBITRARY BOOLEAN ALGEBRAS

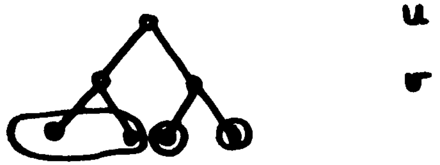
$$t = x_1 \overline{x_2} + x_3$$
$$= x_1 \overline{x_2} + (x_1 + \overline{x_1})\, x_3$$
$$= x_1 (\overline{x_2} + x_3) + \overline{x_1}\, x_3$$

⋮



$$\#(A_3(1) \setminus A_3(t)) = 3$$

=> 2 Variables needed



$$\delta(x_1) = uv + u\overline{v} = u$$
$$\delta(x_2) = u + \overline{u}v$$
$$\delta(x_3) = 0$$

Unification algorithm

C a chain of length u, C: $0 < c_1 < \ldots < c_u = 1$

$G_n := \{f : C^n \to C\}$, (G_n, max, min) = distributive lattice with 0 and 1 (Stepfunctions).

Note: For u = 2 we obtain the boolean algebra of switching functions.

Any $f \in G_n$ is superposition of functions with values 0, 1

- $$f = \sum_{i=1}^{u} g_i \cdot c_i, \quad g_i(x_1, \ldots, x_n) = \begin{cases} 1 \text{ iff } f(x_1, \ldots, x_n) = C_i \\ 0 \text{ else} \end{cases}$$

(disjoint representation)

[illegible] $f = \sum_{i=1}^{u} D_i(f) \cdot c_i$ where $D_i(f) = \begin{cases} 1 \text{ iff } D_i(f)\ (x_1, \ldots\ldots, x_n) \not\leq C_1^{\bullet} \\ 0 \text{ else} \end{cases}$

hence $i \leq j \Rightarrow D_i(f) \leq D_j(f)$ (monotoneous representation)

Note: The functions $D_i(f)$ map into $\{0, 1\}$ and therefore form a (nonfree) boolean algebra $C(G_n)$.

Unary operators on G_n:

D_i $(1 \leq i \leq n)$ and

$C(f) := \overline{D_n(f)}$

$(G_n, \max, \min, C, D_1, \ldots, D_n)$ = free Postalgebra in n generators.

Unification in G_n can be reduced to unification in $C(G_n)$.

(Any homomorphism ζ: $C(G_n) \rightarrow C(G_m)$ extends to a homomorphism from G_n to G_m, and conversely.)

$$t \in G_n \Rightarrow t = \sum_{i=1}^{u} D_i(t) \cdot C_i;$$

Unificationproblem: Find $\zeta: G_n \rightarrow G_m$ with $\zeta(t) = 0$;

$\zeta(t) = 0 \Rightarrow \; D_i(t)\,) = 0 \Rightarrow$ all atoms in $D_i(t)$ are mapped under ζ onto 0 $(1 \leq i \leq n)$.

Now the unification problem can be pursued as for boolean algebras.

Unification in Functionally Complete Algebras

Tobias Nipkow
Department of Computer Science
University of Manchester
Manchester M13 9PL

January 18, 1987

Abstract

Unification in functionally complete algebras is shown to be unary. Three different unification algorithms are investigated. The simplest one consists of computing all solutions and coding them up in a single vector of polynomials. The other two methods are derived from unification algorithms for boolean algebras.

There are two applications which are studied in more detail: Post algebras and matrix rings over finite fields. The former are algebraic models for many-valued logics, the latter cover in particular modular arithmetic.

Unification in Functionally Complete Algebras and Their Products

Tobias Nipkow
Manchester

JANET: tobias@uk.ac.man.cs.ux

Signature = finite set of finitary function symbols

An algebra A is *functionally complete* iff every finitary function can be expressed as a *polynomial* (term with values from A as constants).

$\Rightarrow$ functionally complete algebras are *finite*

Examples : the 2-element boolean algebra
m-element Post-algebras of order m
finite simple non-abelian groups
finite fields
matrix rings over finite fields

All functionally complete algebras are *simple*
The converse does not hold.

Post's Characterization :

An algebra A, $|A| \geq 2$, is functionally complete iff there are $0, 1 \in A$, $0 \neq 1$, two binomials $+, * : A^2 \to A$ such that

$$0 + x = x + 0 = x$$
$$0 * x = 0$$
$$1 * x = x$$

and for every $a \in A$ there is a monomial $\chi_a : A \to \{0,1\}$ such that

$$\chi_a(x) = \begin{cases} 1 & \text{if } x = a \\ 0 & \text{if } x \neq a \end{cases}$$

$$f(\underline{x}) = \sum_{\underline{a} \in A^n} f(\underline{a}) * \chi_{\underline{a}}(\underline{x})$$

$$\chi_{\underline{a}}(\underline{x}) = \prod_{i=1}^{n} \chi_{a_i}(x_i)$$

where Σ and Π are iterations of + and

Alternative : Sheffer-functions

Unification = In/Un/Dis/Anti-Unification
= Matching

$s = t \iff neq(s,t) = 0$

$s \neq t \iff eq(s,t) = 0$

$$neq(x,y) = \begin{cases} 1 & \text{if } x \neq y \\ 0 & \text{if } x = y \end{cases}$$

$$eq(x,y) = \begin{cases} 0 & \text{if } x \neq y \\ 1 & \text{if } x = y \end{cases}$$

$\Rightarrow$ W.l.o.g. $t(\underline{x}) = 0$

Method I - Computing all Solutions

$$S = \{ \underline{a} \in A^n \mid t(\underline{a}) = 0 \}$$

Principle: Find a vector of polynomials that enumerate S.

A <u>reproductive</u> solution:

$F : A^n \to S \quad , \underline{b} \in S \neq \{\}$

$$F(\underline{x}) = \sum_{\underline{a} \in S} \underline{a} * \chi_{\underline{a}}(\underline{x}) + \underline{b} * \sum_{\underline{a} \in A^n \setminus S} \chi_{\underline{a}}(\underline{x})$$

$$F(\underline{x}) = \begin{cases} \underline{x} & \text{if } \underline{x} \in S \\ \underline{b} & \text{if } x \notin S \end{cases}$$

F is a <u>most general unifier</u>

Minimal number of parameters in an mgu

$$\log_{|A|} |S|$$

Method II - Finding one Solution

Let $\underline{b} \in A^n$ such that $t(\underline{b}) = 0$

$$F(\underline{x}) = \underline{\text{if}}\ t(\underline{x}) = 0\ \underline{\text{then}}\ \underline{x}\ \underline{\text{else}}\ \underline{b}$$

$$= \chi_0(t(\underline{x})) * \underline{x} + \chi_0(\chi_0(t(\underline{x}))) * \underline{b}$$

F is a reproductive solution with exactly n parameters.

Special case: Löwenheim's formula for solving boolean equations.

Remark: $(N, +, *, \dot{-})$ is of unification type 1 (unitary) but is undecidable. Use Method II with

$$\chi_0(x) = (x+1) \dot{-} 2*x$$

Method III - Successive Variable Elimination

$$(*)\quad t(x_1, .., x_n) = 0 \iff \exists a \in A : t(a, x_2, .., x_n) =$$

$$\iff \bigcap_{a \in A} t(a, x_2 ... x_n) = 0 \quad (**$$

where $x \cap y = 0 \iff (x = 0 \text{ or } y = 0)$

If $G : A^{n-1} \rightarrow A^{n-1}$ is a mgu of $(**)$, $F : A^n \rightarrow A^n$ is a mgu of $(*)$.

$$F(x_1 .. x_n) = (f(x_1, .., x_n), G(x_2 ... x_n))$$

$$f(x_1 ... x_n) = \underline{\text{if}}\ t(x_1, G(x_2 ... x_n)) = 0\ \underline{\text{then}}\ x_1$$
$$\underline{\text{else}}\ s(G(x_2 ... x_n))$$

$$s(x_2 ... x_n) = \bigcup_{a \in A} a * \chi_0(t(a, x_2 ... x_n))$$

where $x \cup 0 = 0 \cup x = x$ and $x \cup y \in \{x, y\}$

Special case: Boole's / Schröders method of solving boolean equations

Applications

Boolean algebra, reasoning about hardware

Post algebras, multiple valued logics

Finite fields, modular arithmetic in $\mathbb{Z}_p$

Matrix rings over $\mathbb{Z}_p$

Products (direct sum) of algebras

Let $P = A_1 \times \dots \times A_n$ where unification in each A_i is unitary.
To solve $s = t$ in P $(*)$
solve all $s_i = t_i$ in A_i $(**)$
and combine the solutions of $(**)$ into a solution of $(*)$.

In general the combination is not possible.

A sufficient condition is the existence of $0, 1, +$ and $*$ in each A_i.

$\Rightarrow$ Products of functionally complete algebras have a unitary unification problem.

Examples: All finite boolean & Post algebras
Semisimple Artinian rings

An Example

$P = B_4 = \{0, a, a+1, 1\} = B_2 \times B_2 = \{\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}0\\1\end{pmatrix}, \begin{pmatrix}1\\1\end{pmatrix}$

$$a*x*y + y = 0 \qquad \text{in } P$$

$$\Leftrightarrow \begin{pmatrix}1\\0\end{pmatrix} * \begin{pmatrix}x_1\\x_2\end{pmatrix} * \begin{pmatrix}y_1\\y_2\end{pmatrix} + \begin{pmatrix}y_1\\y_2\end{pmatrix} = 0 \qquad \text{in } B_2 \times B_2$$

$$\Leftrightarrow x_1 * y_1 + y_1 = 0 \quad \text{and} \quad y_2 = 0 \qquad \text{in } B_2$$

Mgu in B_2: $\begin{array}{ll} x_1 \to x_1 & x_2 \to x_2 \\ y_1 \to x_1 * y_1 & y_2 \to 0 \end{array}$

$$x = \begin{pmatrix}x_1\\x_2\end{pmatrix} \to \begin{pmatrix}x_1\\x_2\end{pmatrix} = x$$

$$y = \begin{pmatrix}y_1\\y_2\end{pmatrix} \to \begin{pmatrix}x_1 * y_1\\y_2\end{pmatrix} = \begin{pmatrix}x_1\\x_2\end{pmatrix} * \begin{pmatrix}y_1\\y_2\end{pmatrix} * \begin{pmatrix}1\\0\end{pmatrix} + \begin{pmatrix}y_1\\y_2\end{pmatrix} \begin{pmatrix}0\\1\end{pmatrix}$$

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

Mgu in P: $\begin{array}{l} x \to x \\ y \to a * x * y + (a+1) * y \end{array}$

Ursula Martin

Tobias Nipkow

UNIFICATION IN BOOLEAN RINGS

In this talk we describe a basic algorithm for unification in Boolean rings, expressed in terms of exclusive or, and, 0 and 1. We explain how the algorithm can be transalted to give a unification algorithm for any other complete set of operators describing a Boolean ring; such as nand for example. We then go on to describe how our methods extend to give a reformulation of Herbrands therorem for the first order predicate calculus, which leads to the formulation of a semi-decision procedure expressed in terms of unification alone.

UNIFICATION
IN
BOOLEAN RINGS

Ursula Martin
Tobias Nipkow

Computer Science Dept
University of Manchester

② Boolean rings

$$a + b = b + a \qquad a * b = b * a$$

$$(a+b) + c = a + (b + c) \qquad (a * b) * c = a * (b * c)$$

$$a + 0 = a \qquad a * (b + c) = a * b + a * c$$

$$a + (-a) = 0 \qquad a * a = a$$

$$1 * a = a \qquad a + a = 0$$

Examples

a) Propositional calculus under <u>excl. or</u> (+) and <u>and</u> (.)
$T = 1 \quad F = 0$

b) Power set of a set under <u>Sym. diff</u> (+) and <u>intsen</u> (.)
$S = 1, \; \emptyset = 0$

c) Quantifier free formulae of FOPC under <u>excl. or</u> and <u>and</u>

Skolem's Theorem

let P be a

2) To solve

$$f(x_1, x_2, \ldots, x_n) = 0$$

Suppose we have one solution $x_i \rightarrow a_i$

The most general unifier is

$$x_i \rightarrow x_i + f(x_1, \ldots, x_n)(x_i + a_i)$$

To find one solution

$f(x_1 \ldots x_n) = 0$ has a solution if and only if $f(0, x_2, \ldots, x_n) * f(1, x_2, \ldots x_n) = 0$ has a soldution.

Repeat the process, solve for $x_n, x_{n-1}, \ldots, x_1$.

xample

$$xy + y + a = 0$$

ne solution $x \to 0$
$y \to a$

1G U

$$x \to x + (xy + y + a)(x + 0) = x + xa$$

$$y \to y + (xy + y + a)(y + a) = xya + xy + a$$

$$1) \cdot xy + y + a = x\underbrace{a}_{f(1,y)} + (x+1)\underbrace{(y+a)}_{f(0,y)} = 0$$

$$y \quad f(x,y) = 0 \iff \exists y \quad a(y+a) = 0$$

$$\iff a(1+a) = 0$$

hence $y \to a$
$x \to 0$

③ $f(x_1 \cdots x_n) = f \qquad f(1, x_2, \cdots x_n) = A \qquad f(0, x_2, \cdots x_n) = B$

Operators	MGU	discriminant
$+ \; * \; 0 \; 1$	$x_i \to x_i * (1+f) + a_i * f$	$A * B$
$\wedge \; \vee \; \neg$	$x_i \to (x_i \wedge \neg f) \vee (a_i \wedge f)$	$A * B$
ite , 0, 1	$x_i \to \text{ite}(f, a_i, x_i)$	$\text{ite}(A, B, 0)$
nand |	$x_i \to (x_i \mid (f \mid f)) \mid (a_i \mid f)$	$(A \mid B) \mid (A \mid B)$

Dis unification

$B = \{0, 1, a, 1+a\}$

$ax + a = 0 \qquad x \to z(1+a) + a$

$ax + a \neq 0 \qquad x \to z(1+a)$

Booleam ring + free function symbols

Unification is not unitary!

$f(x) * f(y) = f(a) * f(b)$

GU's are $\begin{cases} x \to a \\ y \to b \end{cases}$ $\qquad x + \cancel{0} = \cancel{x}$

$\begin{cases} x \to b \\ y \to a \end{cases}$

Conjecture : Unification is finitary

$f(x * y) = x \qquad MGU \begin{cases} x \to f(0) \\ y \to z * (1 + f(0)) \end{cases}$

(7)

A semi decision procedure for the FOPC

Theorem

Let P be a wff of the FOPC, and let $Q(x_1, \ldots, x_n)$ be its Skolem form (expressed in terms of + and .)

Then P is unsatisfiable if and only if one of

$Q(x_1, \ldots x_n)$

$Q(x_1, \ldots x_n) . Q(x_{n+1}, \ldots, x_{2n})$

$\vdots$

$Q(x_1 \ldots x_n) . \ldots . Q(x_{mn+1}, \ldots, x_{(m+1)n})$

$\vdots$

can be unified with 0.

D

Example

P $\quad \forall x \{ [p(x) \wedge \neg (p(a) \wedge p(b))] \vee [\neg p(x) \wedge p(a) \wedge p(b)] \}$

Q $\quad p(x) + p(a).\,p(b)$

$p(x) + p(a).\,p(b) = 0$ – cant unify

$\{p(x) + p(a)\,p(b)\}\{p(y) + p(a).\,p(b)\} = 0$

can unify – take $x = a$, $y = b$.

Thus P is unsatisfiable.

method

$$p(x) + p(a).\,p(b)$$

$$= (\overline{p(x)}+1)\,\overline{p(a)}\,p(b) \qquad x \to a \text{ or } x \to b$$

$$+\ p(x)\,p(a)\,(1+p(b)) \qquad x \to b$$

$$+\ p(x)\,(1+p(a))\,p(b) \qquad x \to a$$

$$+\ p(x)\,(1+p(a))\,(1+p(b))$$

This is 0 ⟺ each summand is 0.
No substitution makes all summands 0.
∴ can't be unified with 0.

A GENERAL COMPLETE E-UNIFICATION PROCEDURE

Jean H. Gallier and Wayne Snyder[1]
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, Pa 19104

In this paper, a general unification procedure that enumerates a complete set of E-unifiers of two terms for any *arbitrary* set E of equations is presented. It is more efficient than the brute force approach using paramodulation, because many redundant E-unifiers arising by rewriting at or below variable occurrences are pruned out by our procedure, still retaining a complete set. This procedure can be viewed as a nondeterministic implementation of a generalization of the Martelli-Montanari method of transformations on systems of terms [13], which has its roots in Herbrand's thesis [7]. Remarkably, only two new transformations need to be added to the transformations used for standard unification. This approach differs from previous work based on transformations because, rather than sticking rather closely to the Martelli-Montanari approach using multi-equations [13] as in Kirchner [10,11], we introduce transformations dealing directly with rewrite rules.

As an example of the flexibility of this approach, we apply it to the problem of higher-order unification, and find an improved version of Huet's procedure [8]. Our major new result is the presentation and justification of a method for enumerating (relatively minimal) complete sets of unifiers modulo arbitrary sets of equations.

Hans-Jürgen BURCKERT

Usually matching is considered as a special form of unification. Hence most research in unification theory does not consider the problems arising in matching. After discussing the various definitions of matching in the literature we compare matching and unification in the more general framework of restricted unification. Restricted unification is unification of terms where not all variables are allowed for substitution. Matching and unification are special cases of restricted unification. We give some examples where matching and unification behave different especially we present an equational theory where unification is decidable, however matching is undecidable in this theory. There are also certain results in similar behaviour of matching and unification with respect to the cardinalities of minimal and complete solution sets (unification hierarchy), if we restrict us to socalled collapse free equational theories.

Matching – A Special Case of Unification?

Hans-Jürgen Bürckert*

Matching

(1) semi-unification:

μ matches t to s

$\iff$

$s = \mu s =_E \mu t$

(2) filtering:

μ matches t to s

$\iff$

$s =_E \mu t$

(1) $\iff$ (2) if $Var(s) \cap Var(t) = \emptyset$

Example:

→ $\{x \leftarrow g(x)\}$ matches $f(x)$ to $f(g(x))$ (2)

→ no match of $f(x)$ to $f(g(x))$ (1)

We choose (1), since (2) can be reduced to (1).

7

σ filters t to s $(s =_E \sigma t)$

$\Longleftrightarrow$

$\sigma = \tau \cdot \varrho$ and

τ semi-unifies ϱt to s

$(s = \tau s =_E \tau \varrho t = \sigma t)$

where ϱ renames the common variables of s and t by new variables

=

most general matchers?

minimal & complete sets of matchers w.r.t. some instance relation

$\geq [V]$

- represent exactly the set of all matchers
- contain no redundant matchers

6. Matching Problem

+

$\langle t_i \ll s_i : 1 \leq i \leq n \rangle_E$

matching t_i to s_i

Solutions

$M_E(t_i \ll s_i) = \{\mu : s_i = \mu s_i =_E \mu t_i\}$

i.e. $Dom\,\mu \subseteq Var(t_i) \setminus Var(s_i)$

Example

Matching x to fy

$\{x \leftarrow fy\}$ is mgm

$\{x \leftarrow fa\}$ is an instance, but not a matcher

if we compare matchers by $\geq [\{x\}]$

"wrong" instances

minimal & complete E-matcher sets

(1) [illegible] $(t_i \ll s_i) \subseteq \Pi_E(t_i \ll s_i)$

(2) $\forall \delta \in \Pi'_E(t_i \ll s_i)$

$$\exists \mu \in \mu\Pi_E \ : \ \mu \leq_E \delta \ [\mathbf{W}]$$

(3) $\mu, \nu \in \mu\Pi_E, \ \mu \leq_E \nu \ [\mathbf{W}] \Rightarrow \mu = \nu$

$$\boxed{\mathbf{W} = Var(s_i) \cup Var(t_i)}$$

Representation Theorem

$\forall \delta$ with $Dom\,\delta \subseteq Var(t_i) \setminus Var(s_i)$

$$\boxed{\begin{array}{c} \delta \in \Pi_E(t_i \ll s_i) \\ \iff \\ \delta \geq_E \sigma \ [Var(s_i) \cup Var(t_i)] \\ \text{for some } \sigma \in \mu\Pi_E \end{array}}$$

Matching

=

Unification, where

the blocked variables

are considered

as constants

NEW SIGNATURE!

Conjecture

$E \in \mathcal{U}_1 \Rightarrow E \in \mathcal{M}_1$

$E \in \mathcal{U}_\omega \Rightarrow E \in \mathcal{M}_\omega$

$E \in \mathcal{M}_\infty \Rightarrow E \in \mathcal{U}_\infty$

$E \in \mathcal{M}_0 \Rightarrow E \in \mathcal{U}_0$

Wrong in general, but

Theorem:

E collapse free, then

$\mathcal{U}_1 \subseteq \mathcal{M}_1$

$\mathcal{U}_\omega \subseteq \mathcal{M}_\omega$

Counter example

signature: $f \in FUN_2$

$1 \in FUN_0$

theory: AC1

$$f(f(x\ y)\ z) = f(x\ f(y\ z))$$
$$f(x\ y) = f(y\ x)$$
$$f(x\ 1) = x$$

AC1-unification problems

$\langle x_1 \dots x_n = y_1 \dots y_m \rangle_{AC1}$ or $\langle x_1 \dots x_n = 1 \rangle_{AC1}$

are all unitary

$\Longrightarrow AC1 \in \mathcal{U}_1$

However $AC1 \in \mathcal{M}_\omega$

$\langle x\,y = z\,z \rangle_{AC1}$

$\{x \leftarrow z, y \leftarrow z\}, \{x \leftarrow zz, y \leftarrow 1\}, \{x \leftarrow 1, y \leftarrow zz\}$

Example

$f, g \in FUN_3$, $k \in FUN_2$, $h \in FUN_1$, $a \in FUN_0$

E:

$$f(x\ g(y\ z\ v)\ v) = g(f(x\ y\ v)\ f(x\ z\ v)\ v)$$
$$f(g(x\ y\ v)\ z\ v) = g(f(x\ z\ v)\ f(y\ z\ v)\ v)$$
$$f(f(x\ y\ v)\ z\ v) = f(x\ f(y\ z\ v)\ v)$$

$f(x\ y\ a) = a$	$g(x\ y\ a) = a$
$f(x\ y\ f(u\ v\ w)) = a$	$g(x\ y\ f(u\ v\ w)) = a$
$f(x\ y\ g(u\ v\ w)) = a$	$g(x\ y\ g(u\ v\ w)) = a$
$f(x\ y\ k(u\ v)) = a$	$g(x\ y\ k(u\ v)) = a$
$f(x\ y\ h(v)) = a$	$g(x\ y\ h(w)) = a$

$$k(x\ x) = h(x)$$

E-Unification decidable

E-Matching undecidable

Matching by Rewriting

Jalel MZALI

C R I N
Centre de Recherche en Informatique de Nancy
BP 239 54506 Vandoeuvre-les-Nancy
France

ABSTRACT

Our goal is

- First, to axiomatise the problem of matching in an equational theory
- Second, to orient the axioms obtained into rules
- Third, to compute the system of rules by using Knuth-Bendix algorithm
- Finally, to compute matchers by normalisation.

We give two examples of computing matchers with the empty theory and with a commutative theory.

EQUATION

$$M \ll_E N$$

$$EF(M \ll_E N) = \{\sigma \,/\, \sigma M =_E N\}$$

Normalized equation $x \ll_E N$

SYSTEM

$$S = M_1 \ll_{E_1} N_1 \wedge \cdots \wedge M_n \ll_{E_n} N_n$$

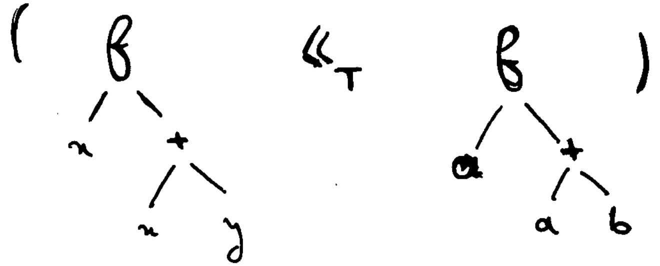
$$EF(S) = \{\sigma \,/\, \forall i = 1,n \quad \sigma \in EF(M_i \ll N_i)$$

DISJONCTION

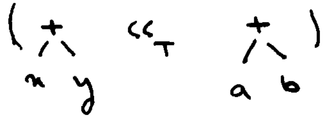
$$Q = S_1 \vee \cdots \vee S_n$$

$$EF(Q) = \bigcup_{i=1,n} EF(S_i)$$

$$T = \left\{ \begin{array}{ll} f & \emptyset\text{-} \\ + & C\text{-} \end{array} \right.$$

- 

$\downarrow$

- $(x \ll_T a) \wedge$ 

$\downarrow$

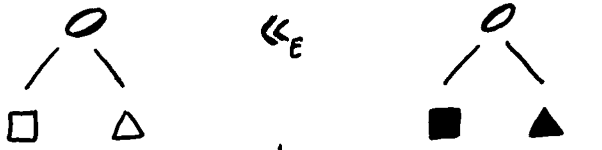
- $(x \ll_T a) \wedge \Big[\big((x \ll_T a) \wedge (y \ll_T b) \big)$
  $\vee \big((x \ll_T b) \wedge (y \ll_T a) \big) \Big]$

- $\downarrow$
- $\vdots$

$\downarrow$

- $\big((x \ll_T a) \wedge (y \ll_T b) \big) \vee \emptyset$

DECOMPOSITION



$\downarrow$

□ $\ll_E$ ■ $\wedge$ △ $\ll_E$ ▲

$Fd \subseteq F$

$t = f(t_1, \dots, t_n)$

$t' = g(t'_1, \dots, t'_{m'})$

$f \in Fd, \quad f = g \Rightarrow \left[t \ll_E t' \Leftrightarrow \bigwedge_{i=1}^{n} (t_i \ll_E t'_i) \right]$

$f_d \in Fd$

$f_d(t_1, \dots, t_n) \ll_E f_d(t'_1, \dots, t'_n) \rightarrow t_1 \ll_E t'_1 \wedge \dots \wedge t_n \ll_E t'_n$

NON EXISTANCE OF SOLUTIONS

$F_{ex} \subseteq F \times F$ Exclusif Symbol

$\forall\ (f, g) \in F_{ex}$

$EF\left(f(\dots t_i \dots) \ll_\varepsilon g(\dots t'_i \dots)\right) = \emptyset$

$(f_{ex}, g_{ex}) \in F_{ex}$

$f_{ex}(t_1, \dots, t_n) \ll_\varepsilon g_{ex}(t'_1, \dots, t'_\ell)$ → **Echec**

MERGING

$$\begin{cases} t \ll_\varepsilon M \\ x \ll_\varepsilon \blacksquare \\ y \ll_\varepsilon \blacktriangle \\ x \ll_\varepsilon \bullet \end{cases} \rightarrow \begin{cases} t \ll_\varepsilon M \\ \blacksquare =_\varepsilon \bullet \\ y \ll_\varepsilon \blacktriangle \\ x \ll_\varepsilon \blacksquare \end{cases}$$

$$x \ll_\varepsilon t_1 \wedge x \ll_\varepsilon t_2 \rightarrow x \ll_\varepsilon t_1 \wedge t_1 =_\varepsilon t_2$$

MUTATION

$$G \longrightarrow D$$

$$EF(G) = EF(D)$$

Matching Algorithm

$$\left.\begin{array}{l}f_d(\ldots t_i \ldots) \ll_\varepsilon f_d(\ldots t'_i \ldots) \to \wedge(t_i \ll_\varepsilon t'_i\\ h_d(\;) \ll_\varepsilon h_d(\ldots t'_i \ldots) \to \wedge(t_i \ll_\varepsilon t'_i)\end{array}\right\} \text{DEC}$$

$$\left.\begin{array}{l}\vdots\\ f_{e_1}(\ldots) \ll_\varepsilon f_{e_2}(\ldots) \to \text{Echec}\end{array}\right\} \text{NON Exi. Sol}$$

$$\left.\begin{array}{l}\vdots \quad \to\end{array}\right\} \text{MER}$$

$$\left.\begin{array}{l}\vdots \quad \to\\ \vdots \quad \to\end{array}\right\} \text{MUT}$$

$$\Big\downarrow \text{ KB Completion}$$

$$\left.\begin{array}{l}\to\\ \to\\ \to\\ \to\\ \to\\ \to\\ \to\end{array}\right\} \text{Matching Algorithm}$$

- Matching ~ Normalisation

Exemple 0.1 -- Soit F l'ensemble des symbole de fonction {f,g,a,b}, f et g sont d'arité deux, a et b des constante. **Considérant le système de réécriture qui axiomatise les trois phase de l'algorithme de filtrage où** "echec" est la substitution vide, Id est l'identité.

```
V1: a « b == echec
V2: (x « b) ∧ (x « a)  ==  echec
V3: f(x, y) « g(z, u) == echec
```

No Solution

```
V4: f(x, y) « f(z, u) == (x « z) ∧ (y « u)
V5: g(x, y) « g(z, u) == (x « z) ∧ (y « u)
V6: a«a == Id
V7: b«b == Id
```

Dec

```
V8: echec ∧ x == echec
V9: x ∧ Id == x
V10: x ∧ x == x
```

Les trois premières règles (V1 V2 V3) décrivent l'exclusivité, les quatre suivante (V4 V5 V6 V7) la décomposition et les deux dernière (v8 V7) decrivent les propriété de l'opération ∧. La complétion de ce système avec ∧ AC engendre le système suivant:

```
   V1: a « b → echec
   V2: (x « b) ∧ (x « a)  →  echec
   V3: f(x, y) « g(z, u) → echec
  •V11: b « a → echec

   V4: f(x, y) « f(z, u) → (x « z) ∧ (y « u)
   V5: g(x, y) « g(z, u) → (x « z) ∧ (y « u)
   V6: a«a → Id
   V7: b«b → Id

   V8: echec ∧ x → echec
   V9: x ∧ Id → x
   V10: x ∧ x → x
```

Ce système convergent peut être utilisé comme un algorithme de filtrage dans la théorie considéré, il suffit de normaliser le terme M«N pour trouver le filtre de M vers N. donnons une sortie de la normatisation

utilisant le système REVE:

Résolvons l'équation f(x,g(a,x))«f(a,g(a,b))

```
Please enter the term for which you would like the normal form computed,
terminated by <ESC>:
f(x,g(a,x))«f(a,g(a,b))

The sequence of term réductions leading to the normal form of your term is:
f(x, g(a, x)) « f(a, g(a, b))
(g(a, x) « g(a, b)) ∧ (x « a)
(a « a) ∧ (x « a) ∧ (x « b)
(a « a) ∧ echec
echec
```

Considérons l'équation suivante

f(f(x,y),f(g(a,b),f(x,y)))«f(f(a,b),f(g(a,b),f(a,b)))

```
Please enter the term for which you would like the normal form computed,
terminated by <ESC>:
f(f(x,y),f(g(a,b),f(x,y)))«f(f(a,b),f(g(a,b),f(a,b)))

The sequence of term réductions leading to the normal form of your term is:
f(f(x, y), f(g(a, b), f(x, y))) « f(f(a, b), f(g(a, b), f(a, b)))
(f(g(a, b), f(x, y)) « f(g(a, b), f(a, b))) ∧ (f(x, y) « f(a, b))
(f(x, y) « f(a, b)) ∧ (f(x, y) « f(a, b)) ∧ (g(a, b) « g(a, b))
(f(x, y) « f(a, b)) ∧ (g(a, b) « g(a, b))
(g(a, b) « g(a, b)) ∧ (x « a) ∧ (y « b)
(a « a) ∧ (b « b) ∧ (x « a) ∧ (y « b)
(b « b) ∧ (x « a) ∧ (y « b) ∧ Id
(b « b) ∧ (x « a) ∧ (y « b)
(x « a) ∧ (y « b) ∧ Id
(x « a) ∧ (y « b)
```

Considérant deux termes égaux:

```
Please enter the term for which you would like the normal form computed,
terminated by <ESC>:
f(a,b)«f(a,b)

The sequence of term réductions leading to the normal form of your term is:
f(a, b) « f(a, b)
(a « a) ∧ (b « b)
(b « b) ∧ Id
b « b
Id
```

∇

Dans cette partie, nous spécifions le mécanisme de décomposition-fusion-mutation par des équations. Le symbole ∧ pour la conjonction d'équations de filtrage, et ∨ pour la disjonction, Id pour la substitution Identité et echec pour la substitution vide. Par exemple pour le cas où l'ensemble des symboles de fonctions est F = {a, b, +} avec + un symbole commutatif, nous avons le système d'équations suivant

(x+y)«(z+u) == ((x«z) ∧ (y « u)) ∨ ((x «u)∧(y«z))
(x « x) «= Id
x ∧ Id == x
a « b == echec
b « a == echec
x ∧ echec == echec
echec ∨ x == x

La première règle est l'axiomatisation de la mutation pour ce cas particulier. les autres règles sont des règle de simplifications. La complétion de ce système complété en considérant ∧ et ∨ AC, nous obtenant le système convergent suivant

(x « x) → Id
(x ∧ Id) → x
(a « b) → echec
(b « a) → echec
(x ∧ echec) → echec
(echec ∨ x) → x
(Id ∨ Id) → Id
(v_1 ∧ (x ∧ echec)) → echec
(v_1 ∧ (x ∧ Id)) → (v_1 ∧ x)
(v_1 ∨ (echec ∨ x)) → (v_1 ∨ x)
(v_1 ∨ (Id ∨ Id)) → (v_1 ∨ Id)
(Id ∨ ((x « y) ∧ (y « x))) → Id
(v_1 ∨ (Id ∨ ((x « y) ∧ (y « x)))) → (v_1 ∨ Id)
((x + y) « (z + u)) → (((x « z) ∧ (y « u)) ∨ ((x « u) ∧ (y « z)))

Ce sytème convergent va nous servir à résoudre des problèmes de filtrage (et d'unification) dans M(F,X)/=C, en normalisant des termes dans ce système.

Par example pour chercher l'unificateur de t_1 « (x+a) et t_2 « (y+b) il suffit de normaliser le terme (x+a)«(y+b), nous donnons une sortie de REVE 3

```
    -> normal-form
    Please enter the term for which you would like the normal form computed,
    terminated by <ESC>:
    (x+a)=(y+b)

    The normal form of your term is:
    ((x « b) ∧ (y « a))
```

qui correspond à la substitution {x←a,y←b}.

```
    -> nor
    Please enter the term for which you would like the normal form computed,
    terminated by <ESC>:
    (a+b) =(y+a)

    The normal form of your term is:
    (b « y)

    -> nor (a+b)=(b+a)

    The normal form of your term is:
    Id
```

T_1 R_1

T_2 R_2

$$T_1 \cup T_2 \quad \left\{ \begin{array}{l} R_1 \cup R_2 \\ \cup \left| \begin{array}{l} f_1(\ldots) \ll_E f_2(\ldots) \to \text{Echec} \\ h_1(\ldots) \ll_E h_2(\ldots) \to \text{Echec} \end{array} \right. \end{array} \right.$$

termination ?