Decision Procedures for Logical Theories

Harald Ganzinger

Winter Term 2002/2003

1 Introduction

- decidable logics (fragments) and theories and their relation to verification tasks
- treating concrete and abstract data structures simultaneously

The State Explosion Problem

Question: What is the difference between MC and TP?

Answer: The size of the models.

Number of states: BDDs: 2^{100}, SAT: 2^{10^7}, in practice (conceptually): ∞

Needed: good (computable) abstract descriptions of ∞

Approach: exploit the structure of the data structures and try to achieve decidability

Practical experience: PVS, STeP, SVC, CVC

Example

Theory 1: $\mathbb{N} = (\mathbb{N}, +, <, \leq)$

Theory 2: $\mathbb{L} = (X^*, \mathsf{nil}, \mathsf{cons}, \mathsf{car}, \mathsf{cdr})$

Theory 3: $\mathbb{B} = (\mathbb{B}, \mathsf{t}, \mathsf{f})$

Enrichment S additional "free" symbols

$\Pi_S = \emptyset$, $\Omega_S = \{sorted/1\}$

with axioms (implicitly universally quantified)

$$\begin{aligned} & sorted(\mathsf{nil}) \approx \mathsf{t} \\ & sorted(\mathsf{cons}(x, \mathsf{nil})) \approx \mathsf{t} \\ x \leq y \;\rightarrow\; & sorted(\mathsf{cons}(x, \mathsf{cons}(y, l))) \approx sorted(\mathsf{cons}(y, l)) \end{aligned}$$

Sample problem: $\forall l\, (sorted(l) \approx \mathsf{t} \;\rightarrow\; sorted(\mathsf{cdr}(l)) \approx \mathsf{t} \;\vee\; l \approx \mathsf{nil})$

That means prove or refute

$$(\mathbb{N}{+}\mathbb{L}{+}\mathbb{B})^{sorted} \models \forall S \;\rightarrow\; \forall l\, (sorted(l) \approx \mathsf{t} \;\rightarrow\; sorted(\mathsf{cdr}(l)) \approx \mathsf{t} \vee l \approx \mathsf{nil})$$

The Setting

- Inference problems: check validity, find solution, find counterexample
- Which theories have decidable inference problems? Which class of problems is decidable?
- Having theories with decidable inference problems, consider their combination and extensions by free functions, possibly with additional axioms.
  Which inference problems remain decidable?
- Design of good (non-naive, complete) inference system if undecidable
- Black box vs. glass box approach

Complications

primitive theories: undecidable or too complex

combinations: sharing of sorts and/or functions between theories

extensions: the enrichment alone can be undecidable

extreme cases: the combination might not even be recursively enumerable

conclusion: we must be modest and consider restricted cases:

sharing, fragments such as $\forall$ or $\exists$

Contents

first-order logic: syntax, semantics, entailment, theories

basic decision methods: automata theory, model theory, universal calculi (tableau, resolution, superposition), complexity analysis

theories: congruence closure, syntactic unification, AC-unification, Presburger arithmetic, linear rational arithmetic, first-order theory of the real numbers with, lists, arrays

combination methods: Nelson/Oppen, Shostak

Literature

General: Schöning: Logik für Informatiker, Spektrum
Fitting: First-Order Logic and Automated Theorem Proving, Springer
Huth, Ryan: Logic in Computer Science: Modelling and reasoning about systems, Cambridge University Press
Baader, Nipkow: Term rewriting and all that. Cambridge U. Press, 1998, Chapter 2.

Specific: various journal and conference papers, cf. below

2 First-Order Logic with Equality

- formalizes fundamental mathematical concepts
- expressive (Turing-complete)
- not too expressive (not axiomatizable: natural numbers, uncountable sets)
- rich structure of decidable fragments
- rich model and proof theory

First-order logic is also called (first-order) predicate logic.

2.1 Syntax

- a signature Σ defines the non-logical symbols (domain-specific)
  $\Rightarrow$ terms, atomic formulas
- fixed are the logical symbols (domain-independent):
  equality ($\approx$), Boolean combinations ($\neg$, $\vee$, $\wedge$, $\rightarrow$, $\leftrightarrow$), quantifiers ($\forall$, $\exists$)

Example: Peano Arithmetic

$$\begin{aligned}\Sigma_{PA} &= (\Omega_{PA},\ \Pi_{PA})\\ \Omega_{PA} &= \{0/0,\ +/2,\ */2,\ s/1\}\\ \Pi_{PA} &= \{\leq/2,\ </2\}\end{aligned}$$

$+, *, <, \leq$ infix; $* >_p + >_p < >_p \leq$

Exampes of formulas over this signature are:

$\forall x, y(x \leq y \leftrightarrow \exists z(x + z \approx y))$
$\exists x \forall y(x + y \approx y)$
$\forall x, y(x * s(y) \approx x * y + x)$
$\forall x, y(s(x) \approx s(y) \rightarrow x \approx y)$
$\forall x \exists y\ (x < y \wedge \neg \exists z(x < z \wedge z < y))$

Signature

Usage: fixing the alphabet of non-logical symbols

$$\Sigma = (\Omega, \Pi),$$

where

- Ω a set of function symbols f with arity $n \geq 0$, written f/n,
- Π a set of predicate symbols p with arity $m \geq 0$, written p/m.

If $n = 0$ then f is also called a constant (symbol). If $m = 0$ then p is also called a propositional variable. We use letters P, Q, R, S, to denote propositional variables.

Refined concept for practical applications: many-sorted signatures (corresponds to simple type systems in programming languages); not so interesting from a logical point of view

Variables

Predicate logic admits the formulation of abstract, schematic assertions. (Object) variables are the technical tool for schematization.

We assume that

$$X$$

is a given countably infinite set of symbols which we use for (the denotation of) variables.

Terms

Terms over Σ (resp., Σ-terms) are formed according to these syntactic rules:

$$\begin{array}{llll} s,t,u,v & ::= & x & ,\ x \in X \quad \text{(variable)} \\ & \mid & f(s_1,...,s_n) & ,\ f/n \in \Omega \quad \text{(functional term)} \end{array}$$

By $T_\Sigma(X)$ we denote the set of Σ-terms (over X). A term not containing any variable is called a ground term. By T_Σ we denote the set of Σ-ground terms.

In other words, terms are formal expressions with well-balanced brackets which we may also view as marked, ordered trees. The markings are function symbols or variables. The nodes correspond to the subterms of the term. A node v that is marked with a function symbol f of arity n has exactly n subtrees representing the n immediate subterms of v.

Atoms

Atoms (also called atomic formulas) over Σ are formed according to this syntax:

$$\begin{array}{llll} A,B & ::= & p(s_1,...,s_m) & ,\ p/m \in \Pi \\ & \mid & (s \approx t) & \text{(equation)} \end{array}$$

Whenever we admit equations as atomic formulas we are in the realm of first-order logic with equality. Admitting equality does not really increase the expressiveness of first-order logic, (cf. exercises). But deductive systems where equality is treated specifically can be much more efficient.

Literals

$$\begin{array}{llll} L & ::= & A & \text{(positive literal)} \\ & \mid & \neg A & \text{(negative literal)} \end{array}$$

Clauses

$$
\begin{array}{llll}
C, D & ::= & \bot & \text{(empty clause)} \\
 & \mid & L_1 \vee \ldots \vee L_k, \;\; k \geq 1 & \text{(non-empty clause)}
\end{array}
$$

General First-Order Formulas

$F_\Sigma(X)$ is the set of first-order formulas over Σ defined as follows:

$$
\begin{array}{llll}
F, G, H & ::= & \bot & \text{(falsum)} \\
 & \mid & \top & \text{(verum)} \\
 & \mid & A & \text{(atomic formula)} \\
 & \mid & \neg F & \text{(negation)} \\
 & \mid & (F \wedge G) & \text{(conjunction)} \\
 & \mid & (F \vee G) & \text{(disjunction)} \\
 & \mid & (F \rightarrow G) & \text{(implication)} \\
 & \mid & (F \leftrightarrow G) & \text{(equivalence)} \\
 & \mid & \forall x F & \text{(universal quantification)} \\
 & \mid & \exists x F & \text{(existential quantification)}
\end{array}
$$

Notational Conventions

- We omit brackets according to the following rules:
  - $\neg \;>_p\; \vee \;>_p\; \wedge \;>_p\; \rightarrow \;>_p\; \leftrightarrow$
    (binding precedences)
  - $\vee$ and $\wedge$ are associative and commutative
  - $\rightarrow$ is right-associative
- $Qx_1, \ldots, x_n\, F$ abbreviates $Qx_1 \ldots Qx_n\, F$.
- infix-, prefix-, postfix-, or mixfix-notation with the usual operator precedences; examples:

$$
\begin{array}{ccc}
s + t * u & \text{for} & +(s, *(t, u)) \\
s * u \leq t + v & \text{for} & \leq (*(s, u), +(t, v)) \\
-s & \text{for} & -(s) \\
0 & \text{for} & 0()
\end{array}
$$

The Concept of Substitution

Substitution is a fundamental operation on terms and formulas that occurs in all inference systems for first-order logic. Substitution models on the syntactical level the assignment of values to variables. Since we are on the syntactical level, values are represented by terms.

In the presence of quantification it is surprisingly complex.

We need to make sure that the (free) variables in s are not captured upon placing s into the scope of a quantifier, hence the renaming of the bound variable y into a “fresh”, that is, previously unused, variable z.

Why this definition of substitution is well-defined will be discussed below.

Substitutions Formally

Substitutions are mappings

$$\sigma : X \to T_\Sigma(X)$$

such that the domain of σ, that is, the set

$$dom(\sigma) = \{x \in X \mid \sigma(x) \neq x\},$$

is finite. The set of variables introduced by σ, that is, the set of variables occurring in one of the terms $\sigma(x)$, with $x \in dom(\sigma)$, is denoted by $codom(\sigma)$.

Substitutions are often written as $[s_1/x_1, \dots, s_n/x_n]$, with x_i pairwise distinct, and then denote the mapping

$$[s_1/x_1, \dots, s_n/x_n](y) = \begin{cases} s_i, & \text{if } y = x_i \\ y, & \text{otherwise} \end{cases}$$

We also write $x\sigma$ for $\sigma(x)$.

Modifying a Substitution

The modification of a substitution σ at x is defined as follows:

$$\sigma[x \mapsto t](y) = \begin{cases} t, & \text{if } y = x \\ \sigma(y), & \text{otherwise} \end{cases}$$

Application of a Substitution

"Homomorphic" extension of σ to terms and formulas:

$$\begin{aligned}
f(s_1, \dots, s_n)\sigma &= f(s_1\sigma, \dots, s_n\sigma) \\
\bot\sigma &= \bot \\
\top\sigma &= \top \\
p(s_1, \dots, s_n)\sigma &= p(s_1\sigma, \dots, s_n\sigma) \\
(u \approx v)\sigma &= (u\sigma \approx v\sigma) \\
\neg F\sigma &= \neg(F\sigma) \\
(F \rho G)\sigma &= (F\sigma \, \rho \, G\sigma) \ ; \quad \text{for each binary connective } \rho \\
(Qx\, F)\sigma &= Qz\, (F\, \sigma[x \mapsto z]) \ ; \quad \text{with } z \text{ a fresh variable}
\end{aligned}$$

2.2. Semantics

To give semantics to a logical system means to define a notion of truth for the formulas. The concept of truth that we will now define for first-order logic goes back to Tarski.

In classical logic (dating back to Aristoteles) there are "only" two truth values "true" and "false" which we shall denote, respectively, by 1 and 0.

There are multi-valued logics having more than two truth values.

Structures

A Σ-algebra (also called Σ-interpretation or Σ-structure) is a triple

$$\mathcal{A} = (U,\ (f_{\mathcal{A}} : U^n \to U)_{f/n \in \Omega},\ (p_{\mathcal{A}} \subseteq U^m)_{p/m \in \Pi})$$

where $U \neq \emptyset$ is a set, called the universe of $\mathcal{A}$.

Normally, by abuse of notation, we will have $\mathcal{A}$ denote both the algebra and its universe.

By Σ-Alg we denote the class of all Σ-algebras.

Assignments

A variable has no intrinsic meaning. The meaning of a variable has to be defined externally (explicitly or implicitly in a given context) by an assignment.

A (variable) assignment, also called a valuation (over a given Σ-algebra $\mathcal{A}$), is a map $\alpha : X \to \mathcal{A}$.

Variable assignments are the semantic counterparts of substitutions.

Value of a Term in $\mathcal{A}$ with Respect to α

By structural induction we define

$$\mathcal{A}(\alpha) : T_\Sigma(X) \to \mathcal{A}$$

as follows:

$$\begin{aligned} \mathcal{A}(\alpha)(x) &= \alpha(x), && x \in X \\ \mathcal{A}(\alpha)(f(s_1,\dots,s_n)) &= f_{\mathcal{A}}(\mathcal{A}(\alpha)(s_1),\dots,\mathcal{A}(\alpha)(s_n)), && f/n \in \Omega \end{aligned}$$

In the scope of a quantifier we need to evaluate terms with respect to modified assigments. To that end, let $\alpha[x \mapsto a] : X \to \mathcal{A}$, for $x \in X$ and $a \in \mathcal{A}$, denote the assignment

$$\alpha[x \mapsto a](y) := \begin{cases} a & \text{if } x = y \\ \alpha(y) & \text{otherwise} \end{cases}$$

Truth Value of a Formula in $\mathcal{A}$ with Respect to α

The set of truth values is given as $\{0, 1\}$.

$\mathcal{A}(\alpha) : F_\Sigma(X) \to \{0, 1\}$ is defined inductively over the structure of F:

$$\begin{aligned} \mathcal{A}(\alpha)(\bot) &= 0 \\ \mathcal{A}(\alpha)(\top) &= 1 \\ \mathcal{A}(\alpha)(p(s_1,\dots,s_n)) = 1 &\Leftrightarrow (\mathcal{A}(\alpha)(s_1),\dots,\mathcal{A}(\alpha)(s_n)) \in p_{\mathcal{A}} \\ \mathcal{A}(\alpha)(s \approx t) = 1 &\Leftrightarrow \mathcal{A}(\alpha)(s) = \mathcal{A}(\alpha)(t) \\ \mathcal{A}(\alpha)(\neg F) = 1 &\Leftrightarrow \mathcal{A}(\alpha)(F) = 0 \\ \mathcal{A}(\alpha)(F \rho G) &= \mathsf{B}_\rho(\mathcal{A}(\alpha)(F), \mathcal{A}(\alpha)(G)) \\ &\quad \text{with } \mathsf{B}_\rho \text{ the Boolean function associated with } \rho \\ \mathcal{A}(\alpha)(\forall x F) &= \min_{a \in \mathcal{A}} \{\mathcal{A}(\alpha[x \mapsto a])(F)\} \\ \mathcal{A}(\alpha)(\exists x F) &= \max_{a \in \mathcal{A}} \{\mathcal{A}(\alpha[x \mapsto a])(F)\} \end{aligned}$$

Ex: “Standard” Interpretation $\mathbb{N}$ for Peano Arithmetic

$$\begin{array}{rcl}
U_{\mathbb{N}} & = & \{0, 1, 2, \ldots\} \\
0_{\mathbb{N}} & = & 0 \\
s_{\mathbb{N}} & : & n \mapsto n+1 \\
+_{\mathbb{N}} & : & (n,m) \mapsto n+m \\
*_{\mathbb{N}} & : & (n,m) \mapsto n*m \\
\leq_{\mathbb{N}} & = & \{(n,m) \mid n \text{ less than or equal to } m\} \\
<_{\mathbb{N}} & = & \{(n,m) \mid n \text{ less than } m\}
\end{array}$$

Note that $\mathbb{N}$ is just one out of many possible Σ_{PA}-interpretations.

Values over $\mathbb{N}$ for Sample Terms and Formulas

Under the assignment $\alpha : x \mapsto 1, y \mapsto 3$ we obtain

$$\begin{array}{lcl}
\mathbb{N}(\alpha)(s(x) + s(0)) & = & 3 \\
\mathbb{N}(\alpha)(x + y \approx s(y)) & = & 1 \\
\mathbb{N}(\alpha)(\forall x, y(x + y \approx y + x)) & = & 1 \\
\mathbb{N}(\alpha)(\forall z\ z \leq y) & = & 0 \\
\mathbb{N}(\alpha)(\forall x \exists y\ x < y) & = & 1
\end{array}$$

2.3 Models, Validity, and Satisfiability

F is valid in $\mathcal{A} \in \Sigma\text{-alg}$ under assigment α:

$$\mathcal{A}, \alpha \models F \ :\Leftrightarrow\ \mathcal{A}(\alpha)(F) = 1$$

F is valid in $\mathcal{A} \in \Sigma\text{-alg}$ ($\mathcal{A}$ is a model of F):

$$\mathcal{A} \models F \ :\Leftrightarrow\ \mathcal{A}, \alpha \models F, \text{ for all } \alpha \in X \to \mathcal{A}$$

F is valid in a class of structures $\mathcal{M} \subseteq \Sigma\text{-alg}$:

$$\mathcal{M} \models F \ :\Leftrightarrow\ \mathcal{A} \models F, \text{ for all } \mathcal{A} \in \mathcal{M}$$

F is (universally) valid (or is a tautology):

$$\models F \ :\Leftrightarrow\ \mathcal{A} \models F, \text{ for all } \mathcal{A} \in \Sigma\text{-alg}$$

F is called satisfiable [in $\mathcal{M}$] iff there exist $\mathcal{A}$ [$\in \mathcal{M}$] and α such that $\mathcal{A}, \alpha \models F$. Otherwise F is called unsatisfiable [in $\mathcal{M}$].

Substitution Lemma

The following theorems, to be proved by structural induction, hold for all Σ-algebras $\mathcal{A}$, assignments α, and substitutions σ.

THEOREM 2.1 For any Σ-term t,

$$\mathcal{A}(\alpha)(t\sigma) = \mathcal{A}(\alpha \circ \sigma)(t),$$

where $\alpha \circ \sigma : X \to \mathcal{A}$ is the assignment $\alpha \circ \sigma(x) = \mathcal{A}(\alpha)(x\sigma)$.

THEOREM 2.2 For any Σ-formula F,

$$\mathcal{A}(\alpha)(F\sigma) = \mathcal{A}(\alpha \circ \sigma)(F).$$

COROLLARY 2.3 $\mathcal{A}, \alpha \models F\sigma \ \Leftrightarrow\ \mathcal{A}, \alpha \circ \sigma \models F$

These theorems basically express that the syntactic concept of substitution corresponds to the semantic concept of an assignment.

Entailment and Equivalence

In this course we will use the notion of entailment only for closed formulas. Let F and G be closed formulas.

F entails (implies) G (or G is entailed by F), written $F \models G$
:⇔ for all $\mathcal{A} \in \Sigma\text{-alg}$, whenever $\mathcal{A} \models F$ then $\mathcal{A} \models G$.

F and G are called equivalent
:⇔ for all $\mathcal{A} \in \Sigma\text{-alg}$ we have $\mathcal{A} \models F \;\Leftrightarrow\; \mathcal{A} \models G$.

Convention: Whenever we write $F \models G$, where F or G are not necessarily closed, we consider this as shorthand notation for $\forall F \models \forall G$.

Proposition 2.4 F entails G iff $(F \rightarrow G)$ is valid

Proposition 2.5 F and G are equivalent iff $(F \leftrightarrow G)$ is valid.

Extension to sets of closed formulas $\mathcal{F}$ in the "natural way", e.g., $\mathcal{F} \models G$
:⇔ for all $\mathcal{A} \in \Sigma\text{-alg}$: if $\mathcal{A} \models F$, for all $F \in \mathcal{F}$, then $\mathcal{A} \models G$.

Validity vs. Unsatisfiability

Validity and unsatisfiability are just two sides of the same medal as explained by the following proposition.

Proposition 2.6

$$F \text{ valid} \;\Leftrightarrow\; \neg F \text{ unsatisfiable}$$

Hence in order to design a theorem prover (validity checker) it is sufficient to design a checker for unsatisfiability.

Q: In a similar way, entailment $\mathcal{F} \models F$ can be reduced to unsatisfiability. How?

2.4 Theories

Theories have a syntactic and a semantic aspect.

In the syntactic or axiomatic view, a (first-order) theory is given by a set $\mathcal{F}$ of (closed) first-order Σ-formulas, and then one is interested in the models $\mathsf{Mod}(\mathcal{F})$ of $\mathcal{F}$, that is, the set of Σ-algebras satisfying $\mathcal{F}$:

$$\mathsf{Mod}(\mathcal{F}) = \{\mathcal{A} \in \Sigma\text{-alg} \mid \mathcal{A} \models G, \text{ for all } G \text{ in } \mathcal{F}\}$$

Dually, in the semantic view, when given a class $\mathcal{M} \subseteq \Sigma\text{-alg}$ of structures, one is interested in the (first-order) theory $\mathsf{Th}(\mathcal{M})$ of $\mathcal{M}$

$$\mathsf{Th}(\mathcal{M}) = \{G \in F_\Sigma(X) \text{ closed} \mid \mathcal{M} \models G\}$$

which is the set of Σ-formulas that are satisfied in all structures $\mathcal{A}$ in $\mathcal{M}$.

$\mathsf{Th}(\mathsf{Mod}(\mathcal{F}))$ is the set of formulas true in all models of $\mathcal{F}$. It represents exactly the set of consequences of $\mathcal{F}$. Clearly, $\mathcal{F} \subseteq \mathsf{Th}(\mathsf{Mod}(\mathcal{F}))$, and also $\mathcal{M} \subseteq \mathsf{Mod}(\mathsf{Th}(\mathcal{M}))$. Typically these inclusions are strict.

Two Interesting Theories

Let $\Sigma_{Pres} = (\{0/0, s/1, +/2\}, \emptyset)$ and $\mathbb{Z}_+ = (\mathbb{Z}, 0, s, +)$ its standard interpretation on the integers.[a] $\mathsf{Th}(\mathbb{Z}_+)$ is called Presburger arithmetic.[b] Presburger arithmetic is decidable in 3EXPTIME[c] (and there is a constant $c \geq 0$ such that $\mathsf{Th}(\mathbb{Z}_+) \notin \text{NTIME}(2^{2^{cn}})$) and in 2EXPSPACE; usage of automata-theoretic methods, cf. below.

However, $\mathbb{N}_* = (\mathbb{N}, 0, s, +, *)$, the standard interpretation of $\Sigma_{PA} = (\{0/0, s/1, +/2, */2\}, \emptyset)$, has as theory the so-called Peano arithmetic which is undedidable, not even recursively enumerable.

Note: The choice of signature can make a big difference with regard to the compational complexity of theories.

[a] There is no essential difference when one, instead of $\mathbb{Z}$, considers the natural numbers $\mathbb{N}$ as standard interpretation.

[b] M. Presburger (1929)

[c] D. Oppen: A $2^{2^{2^n}}$ upper bound on the complexity of Presburger arithmetic. Journal of Computer and System Sciences, 16(3):323–332, July 1978

Example: Lists

Lists (binary trees, rather) are an equational theory, axiomatized by these (implicitly universally quantified) equations:

$$\mathsf{car}(\mathsf{cons}(x,y)) \approx x$$
$$\mathsf{cdr}(\mathsf{cons}(x,y)) \approx y$$
$$\mathsf{cons}(\mathsf{car}(x),\mathsf{cdr}(x)) \approx x$$

over the signature Σ_{list} with $\Pi_{\mathsf{list}} = \emptyset$ and $\Omega_{\mathsf{list}} = \{\mathsf{cons}/2,\ \mathsf{car}/1,\ \mathsf{cdr}/1\}$.

To explain how typical models of these axioms look like, viewing the equations as rewrite rules (from left to right) is useful. We need to explain the fundamental concepts of term rewriting.

Combination of Theories and Models

Forgetting symbols. Suppose $\Sigma \subseteq \Sigma'$, that is, $\Omega \subseteq \Omega'$ and $\Pi \subseteq \Pi'$.
For $\mathcal{A} \in \Sigma'$-alg, by $\mathcal{A}|_\Sigma$ we denote the Σ-structure for which

$$\begin{aligned} U_{\mathcal{A}|_\Sigma} &= U_{\mathcal{A}} \\ f_{\mathcal{A}|_\Sigma} &= f_{\mathcal{A}}, \text{ for } f \text{ in } \Omega \\ p_{\mathcal{A}|_\Sigma} &= p_{\mathcal{A}}, \text{ for } p \text{ in } \Pi \end{aligned}$$

That is, we simply ignore the functions and predicates associated with symbols in $\Sigma' \setminus \Sigma$. $\mathcal{A}|_\Sigma$ is called the restriction of $\mathcal{A}$ to Σ.

Amalgamating algebras. Let $\Sigma = \Sigma_1 + \Sigma_2$ (formed by uniting the respective symbol sets). A Σ-Algebra $\mathcal{A}$ is called an amalgamation of Σ_i-algebras $\mathcal{A}_i$ iff, for $i = 1, 2$, we have that $\mathcal{A}|_{\Sigma_i} = \mathcal{A}_i$.

On model classes: Let $\Sigma \subseteq \Sigma'$, $\mathcal{M} \subseteq \Sigma'$-$\mathsf{alg}$, $\mathcal{M}_i \subseteq \Sigma_i$-$\mathsf{alg}$.

$$\begin{aligned} \mathcal{M}|_\Sigma &= \{\mathcal{A}|_\Sigma \mid \mathcal{A} \in \mathcal{M}\} \\ \mathcal{M}_1 + \mathcal{M}_2 &= \{\mathcal{A} \in (\Sigma_1{+}\Sigma_2)\text{-}\mathsf{alg} \mid \mathcal{A}|_{\Sigma_i} \in \mathcal{M}_i, \text{ for } i = 1, 2\} \end{aligned}$$

Canonical List Model

Let L be the set of list axioms oriented as rewrite rules from left to right. (Below we will introduce the basics of term rewriting formally.) We claim (w/o proof) that L is uniquely normalizing.

Define the Σ_{list}-structure $\mathbb{L}$ as follows:

$$\begin{aligned} \mathbb{L} &= \{t \in T_{\Sigma_{\mathsf{list}}}(X) \mid t \text{ is irreducible by } L\} \\ \mathsf{cons}_{\mathbb{L}}(l_1, l_2) &= \mathsf{cons}(l_1, l_2){\downarrow_L} \\ \mathsf{car}_{\mathbb{L}}(l) &= \mathsf{car}(l){\downarrow_L} \\ \mathsf{cdr}_{\mathbb{L}}(l) &= \mathsf{cdr}(l){\downarrow_L} \end{aligned}$$

Regard $s{\downarrow_L}$ as the result of evaluating the “input” s with respect to the “functional program” L.

E: convince yourself that L is uniquely normalizing.
E: show that $\mathbb{L}$ in fact satisfies the list axioms.
Q: why is this a canonical model of L?

3 Rewrite-Based Methods

3.1 Essentials of Term Rewriting

Subterm replacement: If p is a position in a term s, by s/p we denote the subterm of s at position p. If also v is a term, $s[v]_p$ denotes the term resulting from replacing the subterm of s at p by v.

Rewrite rule: pair of terms (over a given signature Σ), written $s \Rightarrow t$

Rewrite system: set R of rewrite rules

One-step rewrite relation: $s \Rightarrow_R t$ iff there exists $l \Rightarrow r$ in R, a position p in s and a substitution σ such that $s/p = l\sigma$ and $t = s[r\sigma]_p$

(Ir-)reducibility: s is called reducible (by R) if $s \Rightarrow_R t$ for some term t, and is called irreducible (by R), otherwise.

Normal form: t is called a normal form of s (by R), if $s \Rightarrow_R^* t$ and t is irreducible by R. R is called uniquely normalizing if every term has exactly one normal form wrt. R. In that case by $s{\downarrow}_R$ we denote that unique normal form of s by R.

Term Rewriting: Basic Definitions II

Termination: A rewrite system R is called terminating if $\Rightarrow_R$ has no infinite chains.

Confluence: R is called confluent if whenever $s \Rightarrow_R^* t$ and $s \Rightarrow_R^* u$, for any three terms s, t, and u, then there exists a term v such that $t \Rightarrow_R^* v$ and $u \Rightarrow_R^* v$.

Local confluence: R is called locally confluent if whenever $s \Rightarrow_R t$ and $s \Rightarrow_R u$, for any three terms s, t, and u, then there exists a term v such that $t \Rightarrow_R^* v$ and $u \Rightarrow_R^* v$.

Joinability: Two terms s and t are called joinable by R, written $s \Downarrow_R t$, if there exists a term u such that $s \Rightarrow_R^* u \Leftarrow_R^* t$.

Bidirectional replacement: We define $\Leftrightarrow_R$ to be the relation $\Rightarrow_R \cup \Leftarrow_R$, formalizing the notion of replacement via R where rewrite rules can be applied both directions.

Term Rewriting: Basic Properties

Theorem 3.1 (Newman 1942) Let R be terminating. Then R is confluent if, and only if, R is locally confluent.

Theorem 3.2 R is confluent if, and only if, the relations $\Leftrightarrow_R^*$ and $\Downarrow_R$ coincide.

Word Problems

Word problems: Given a set E of Σ-equations, the word problem for E is to decide, for any two Σ-terms s and t, whether or not $E \models s \approx t$.[a]

Uniform word problems: Given a set E of (implicitly) universally quantified Σ-equations, the uniform word problem for E is to decide, for any sequence of pairs of Σ-terms s_i and t_i, $0 \leq i \leq n$, $n \geq 0$, whether or not $E \models (s_1 \approx t_1 \wedge \ldots \wedge s_n \approx t_n \rightarrow s_0 \approx t_0)$.[b]

Theory: In general WPs (hence UWPs, too) are undecidable. (Exercise) There are theories E for which the WP is decidable but the UWP is not.

Practice: For many theories WPs and UWPs are efficiently decidable.

[a] According to our definition of entailment on p. 31, what we mean is $\forall E \models \forall(s \approx t)$.

[b] In full notation $\forall E \models \forall(s_1 \approx t_1 \wedge \ldots \wedge s_n \approx t_n \rightarrow s_0 \approx t_0)$.

Equations vs. Rules

If E is a set of equations we may identify it with the set of rewrite rules obtained by orienting the equations from left to right. Conversely, if R is a set of rewrite rules we may identify R with the set of equations obtained by forgetting the orientation of the rules.

In other words, for us rewrite rules and equations have the same logical meaning (both are equations) and we freely convert one into the other.

When we convert equations into rules we need to give them an orientation. Unless specified otherwise we assume an orientation from left to right.

Birkhoff's Theorem

Theorem 3.3 (Birkhoff) Let E be a set of Σ-equations. For any two Σ-terms s and t the following two properties are equivalent:

1. $E \models s \approx t$ 2. $s \Leftrightarrow_E^* t$

Proof. Baader/Nipkow book, chapter 3 □

Corollary 3.4 If R is a confluent rewrite system then the following two properties are equivalent:

1. $R \models s \approx t$ 2. $s \Downarrow_R t$

Theorem 3.5 If R is a finite terminating and confluent rewrite system then the word problem for R is decidable.

Proof. If R is finite and terminating, the relation $\Downarrow_R$ is decidable. □

This theorem represents one of the fundamental methods of obtaining decision procedures for [uniform] word problems.

Term Rewriting: Basic Definitions III

Critical pairs: Let $s \Rightarrow t$ and $u \Rightarrow v$ be two rules do not share any variables (possibly after suitable renaming). If s is unifiable (via a most general unifier σ) with a non-variable subterm u/p of u then we call the pair of terms

$$((u[t]_p)\sigma, v\sigma)$$

a critical pair between these two rules. We often write critical pairs as equations as they are equational consequences of the two rules. The pair is called joinable (in R) if $(u[t]_p)\sigma \Downarrow_R v\sigma$.

Example: Consider the two rules

$$\mathsf{car}(\mathsf{cons}(x, y)) \Rightarrow x$$
$$\mathsf{cons}(\mathsf{car}(x'), \mathsf{cdr}(x')) \Rightarrow x'$$

Then $(\mathsf{car}(x'), \mathsf{car}(x'))$ and $(\mathsf{cons}(x, \mathsf{cdr}(x')), \mathsf{cons}(x, \mathsf{cdr}(\mathsf{cons}(x, y))))$ are the critical pairs between the two rules (both joinable).

Confluence Test

Theorem 3.6 Then R is locally confluent if, and only if, all critical pairs between any two rules in R are joinable in R.

Proof. Baader/Nipkow, chapter 6.1 □

Corollary 3.7 If R is finite and terminating, confluence of R is decidable.

Proof. For terminating systems, local confluence and confluence coincide. Check the joinability of the (finitely many) critical pairs. □

3.2 Knuth/Bendix Completion

Knuth/Bendix completion (1970) attempts at converting a given set of equations E into a logically equivalent, canonical rewrite system R. We compute with equations E (initially given) and rewrite rules R (initially empty). If the process terminates, E is empty and R is canonical and equivalent with the initial equations. For the KB inference rules below use matching modulo the symmetry of $\approx$. This is relevant for Orient and Reduce-E. In particular we may orient an equation in any direction that preserves the termination of the rewrite rules.

The order in which the KB rules are applied is in principle irrelevant as long as we apply them exhaustively. In practice, however, they should be tried in the order as listed, so as to give preference to the simplification of the equations and rewrite rules.

The procedure fails when an equation persists that cannot be oriented. Also, the procedure might not be terminating. Moreover, there is an extra technical condition to be placed on Reduce-R in order to avoid certain trivial cycles in the procedure when applying the KB rules strictly with the given precedence.

KB Inference Rules

$E \cup \{s \approx t\}, R$	$\rhd_{\mathsf{KB}}$	$E \cup \{s' \approx t\}, R$ if $s \Rightarrow_R^+ s'$	(Reduce-E)
$E \cup \{s \approx s\}, R$	$\rhd_{\mathsf{KB}}$	E, R	(Triv)
$E, R \cup \{s \Rightarrow t\}$	$\rhd_{\mathsf{KB}}$	$E \cup \{s' \approx t\}, R$ if $s \Rightarrow_R^+ s'$	(Reduce-R)
$E \cup \{s \approx t\}, R$	$\rhd_{\mathsf{KB}}$	$E, R \cup \{s \Rightarrow t\}$ if $R \cup \{s \Rightarrow t\}$ is terminating	(Orient)
E, R	$\rhd_{\mathsf{KB}}$	$E \cup \{s \approx t\}, R$ if $s \approx t$ is a non-joinable c.p. between two rules in R	(CP)

Knuth/Bendix Completion: Properties

THEOREM 3.8 If $E, R \rhd_{\mathsf{KB}} E', R'$ then $E \cup R \models E' \cup R'$ and $E' \cup R' \models E \cup R$.

THEOREM 3.9 If $E, \emptyset \rhd_{\mathsf{KB}}^* \emptyset, R$ with no KB inference rule applicable to $\emptyset, R$ then R is canonical.

Altogether, if the KB procedure terminates with an empty set of equations the the final set R of rules is canonical and logically equivalent to the initial set E of equations.

In that case the word problem for E is decidable by testing for R-joinability of equations.

Practical problems:
How to prove termination of rewrite systems? (Requires the implementation of so-called rewrite orderings.)

How to make the procedure non-failing? (Requires an extended concept of rewriting with orientable instances of equations. Unfailing KB completion is a semi-decision procedure for any kind of word problems.)

The Overall Picture

Goal: we want to decide the word problem for an equational theory E.

- Birkhoff's theorem links entailment with equational proofs in $\Leftrightarrow_E^*$.
- Confluence allows one to restrict proof search for $\Leftrightarrow_E^*$ to rewrite proofs (relation $\Downarrow_E$).
- Canonicity (confluence+termination) makes searching for rewrite proofs in $\Downarrow_E$ don't-care nondeterministic and terminating.
- KB-completion is targeted at transforming a given E into an equivalent canonical E'.
- An extended procedure can be used to check/achieve decidability of the UWP for E. [C. Lynch, Proc. IEEE Conf. on Logic in Computer Science, 2002]

3.3 Congruence Closure

Problem: Decide (universal) validity of equational Horn clauses

$$\models_\Sigma \forall(s_1 \approx t_1 \land \ldots \land s_n \approx t_n \to s_0 \approx t_0) \qquad (1)$$

over Σ. This is the UWP of the theory of all Σ-algebras.

Skolemization: Consider the variables as constants and thus consider the extended signature $\Sigma(X) = \Sigma \cup X$. Then (1) holds if, and only if,

$$s_1 \approx t_1 \land \ldots \land s_n \approx t_n \models_{\Sigma(X)} s_0 \approx t_0.$$

Prove this as an exercise.

With this we have reduced the congruence closure problem to a word problem for the theory consisting of the antecedent (ground) equations. (That theory, however, changes with each clause!)

Method: Apply a ground version of KB completion to the antecedent such that it always terminates.

Congruence Closure: Preparations

Flattening: Make all (antecedent) equations flat, that is, bring them into the form $a \approx b$ or $f(a_1, \ldots, a_k) \approx b$ with constants a, a_i, b, without changing the given congruence closure problem. This can be achieved in linear time via the use of auxiliary constants/variables to name subterms of non-flat terms.

Example: The congruence closure problem

$$g(x) \approx x,\ f(f(f(x))) \approx x,\ f(f(x)) \approx x \to f(x) \approx g(x)$$

has an equivalent flattened form

$$g(x) \approx x,\ f(x) \approx x_1,\ f(x_1) \approx x_2,\ f(x_2) \approx x,\ f(x_1) \approx x \to f(x) \approx x.$$

Constant ordering: Choose some total ordering $\succ$ on the constants.

Congruence Closure: Inference Rules

Given: CC problem $\quad E \to u \approx v \quad$ such that E is flat.

Orient E into rules R : Orient $a \approx b$ into $a \Rightarrow b$ iff $a \succ b$. Orient $f(a_1, \ldots, a_k) \approx b$ from left to right. Then E becomes a terminating rewrite system R.

Inferences on R: Let a, b, c denote constants, and s arbitrary flat terms:

$$R \cup \{s \Rightarrow b,\ s \Rightarrow c\} \quad \rhd_{\mathrm{CC}} \quad R \cup \{b \Rightarrow c,\ s \Rightarrow c\} \quad \text{if } s \succ b \succ c \qquad \text{(CP-1)}$$

$$R \cup \{a \Rightarrow b,\ f(\ldots, a, \ldots) \Rightarrow c\} \quad \rhd_{\mathrm{CC}} \quad R \cup \{a \Rightarrow b,\ f(\ldots, b, \ldots) \Rightarrow c\} \qquad \text{(CP-2)}$$

$$R \cup \{a \approx a\} \quad \rhd_{\mathrm{CC}} \quad R \qquad \text{(Triv)}$$

Example of a Run

Congruence Closure Procedure: Properties

Theorem 3.10 $\rhd_{CC}$ is a terminating derivation relation.

Proof. At each step R becomes smaller in the two-fold multiset extension of $\succ$ (first to rules and then to sets of rules). □

Theorem 3.11 If $R \rhd_{CC} R'$ then R and R' are equivalent.

Theorem 3.12 Let $R \rhd_{CC}{}^{*} R'$ be such that R' irreducible by $\rhd_{CC}$.

(i) R' is canonical, called the congruence closure of R.

(ii) If $u \approx v$ is a ground equation over $\Sigma(X)$ then

$$\models_{\Sigma(X)} R \rightarrow u \approx v \quad \Leftrightarrow \quad u \Downarrow_{R'} v.$$

Proof. (i) Inferences (CP-1) and (CP-2) compute all critical pairs between the rules. Termination is clear by construction.

(ii) R' is canonical and equivalent to R. □

Congruence Closure Procedure: Complexity

Theorem 3.13 Congruence closure is decidable in time $O(n^2)$.

Proof. Flattening can be done in linear time. We generate at most quadratically many new rules (why?). Every rule can be generated in constant time. □

The upper bound can be improved to $O(n \log n)$ by using union-find for the representation of congruence classes.

Literature:

P. J. Downey, R. Sethi, and R. E. Tarjan. Variations on the common subexpressions problem. J. Association for Computing Machinery, 27(4):771–785, 1980.

Leo Bachmair and Ashish Tiwari. Abstract congruence closure and specializations. LNCS, vol. 1831, 64–78, 2000.

H. Ganzinger, and D. McAllester. A new meta-complexity theorem for bottom-up logic programs. LNCS, vol. 2083, 514—528, 2001.

Application: Compiler Validation

Problem: prove equivalence of source and target program

Example:

```
1:  y := 1                    1: y := 1
2:  if z = x*x*x              2: R1 := x*x
3:    then y := x*x + y       3: R2 := R1*x
4:  endif                     4: jmpNE(z,R2,6)
                              5: y := R1+1
```

To prove: (indexes refer to values at line numbers; index 0 = initial values)

$$y_1 \approx 1 \wedge z_0 \approx x_0 * x_0 * x_0 \wedge y_3 \approx x_0 * x_0 + y_1$$
$$y_1' \approx 1 \wedge R1_2 \approx x_0' * x_0' \wedge R2_3 \approx R1_2 * x_0' \wedge z_0' \approx R2_3 \wedge y_5' \approx R1_2 + 1$$
$$\wedge\, x_0 \approx x_0' \wedge y_0 \approx y_0' \wedge z_0 \approx z_0' \quad \models \quad y_3 \approx y_5'$$

Compiler Validation: Open Problems

- handle large programs
- exploit algebraic properties of operations
- compiler optimization depends on non-local program analysis; verify its correctness
- loop/recursion structure source-target not isomorphic

Literature: various papers by Pnueli, Zuck et al (CAV conferences)

4 Automata-Theoretic Methods

Atomic constraints: solutions as regular sets (over tuples of characters)

Boolean combinations: boolean combinations of automata

Quantifier(s): projection (and cylindrification)

Satisfiability: emptiness test

Literature:
Hubert Comon and Claude Kirchner: Constraint Solving on Terms, LNCS 2002, 47–103, 2001.
The entire LNCS volume is recommended reading.

Example: Presburger (1929) Arithmetic

Primitive constraints: linear [in]equations $b - \sum_{i=1}^{n} a_i \cdot x_i = [\leq]\ 0$

representation of numbers: binary; we read them from right to left

representation of tuples $(x_1, \ldots, x_n)$ of numbers: words of bit tuples $t_1 \ldots t_k$ where x_i is given in binary by the bit string $t_1[j] \ldots t_k[j]$.

recursive characterization of solutions:

$$\boxed{\mathrm{b}} - \sum_{i=1}^{n} a_i \cdot \sum_{j=0}^{m} x_{ij} \cdot 2^j = 0$$

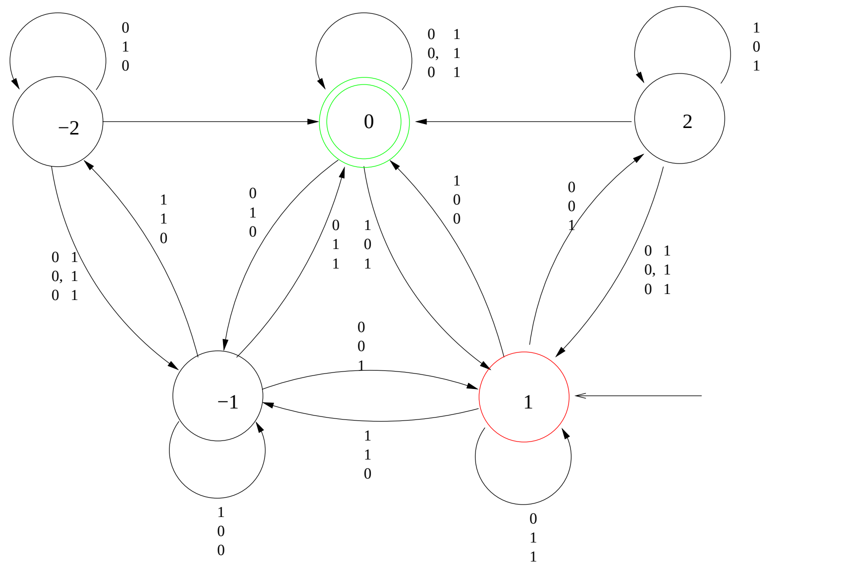
iff

$$(b - \sum_{i=1}^{n} a_i x_{i0}) = 0 \mod 2$$

and

$$\boxed{(b - \textstyle\sum_{i=1}^{n} a_i x_{i0})/2} - \sum_{i=1}^{n} a_i \cdot \sum_{j=0}^{m} x_{i(j+1)} \cdot 2^j = 0$$

Solutions of $1 - x - 2y + 3z = 0$



Automata for Linear Equations

Given an equation of the form

$$b - \sum_{i=1}^{n} a_i \cdot x_i = 0$$

we assume that q_b is the initial state, and we generate the following additional states and transition rules (until no more can be added):

$$\frac{q_c \in Q}{q_d \in Q, \qquad (q_c, \theta, q_d) \in \delta} \text{ if } \begin{cases} a_1\theta_1 + \ldots + a_n\theta_n = c \mod 2 \\ d = (c - a_1\theta_1 - \ldots - a_n\theta_n)/2 \\ \theta \in \{0,1\}^n \text{ encodes } (\theta_1 \ldots \theta_n) \end{cases}$$

q_0 is the final state of the automaton.

Automata for Linear Inequations

Given an inequation of the form

$$b - \sum_{i=1}^{n} a_i \cdot x_i \geq 0$$

we assume that q_b is the initial state, and we generate the following additional states and transition rules (until no more can be added):

$$\frac{q_c \in Q}{q_d \in Q, \qquad (q_c, \theta, q_d) \in \delta} \text{ if } \begin{cases} d = \lfloor (c - a_1\theta_1 - \ldots - a_n\theta_n)/2 \rfloor \\ \theta \in \{0,1\}^n \text{ encodes } (\theta_1 \ldots \theta_n) \end{cases}$$

The final states are $Q_f = \{q_c \mid c \geq 0\}$.

Result

Theorem 4.1 Presburger arithmetic, that is, the first-order theory of the integers with addition, subtraction, scalar multiplication, integer constants, equality and inequality, is decidable.

Different methods show decidability in triple-exponential time, cf. p 34.

Other Decision Procedures based on Automata

- order-sorted term membership constraints — bottom-up tree automata
- weak second-order monadic logic — Rabin automata
- set constraints — term set automata

5 Constraint Simplification Methods

Literature:
Hubert Comon and Claude Kirchner: Constraint Solving on Terms, LNCS 2002, 47–103, 2001.

Unification

Let $E = \{s_1 \approx t_1, \ldots, s_n \approx t_n\}$ (s_i, t_i terms or atoms) a multi-set of equality problems. A substitution σ is called a unifier of E $:\Leftrightarrow$

$$\forall 1 \leq i \leq n : s_i\sigma = t_i\sigma.$$

If a unifier exists, E is called unifiable. If a unifier of E is more general than any other unifier of E, then we speak of a most general unifier (mgu) of E. Hereby a substitution σ is called more general than a substitution τ

$$\sigma \leq \tau \ :\Leftrightarrow \text{ there exists a substitution } \varrho \text{ s.t. } \varrho \circ \sigma = \tau$$

where $(\varrho \circ \sigma)(x) := (x\sigma)\varrho$ is the composition of σ and ϱ als mappings.

Proposition 5.1 (Exercise)

(i) $\leq$ is a quasi-ordering on substitutions, and $\circ$ is associative.

(ii) If $\sigma \leq \tau$ and $\tau \leq \sigma$ (we write $\sigma \sim \tau$ in this case), then $x\sigma$ and $x\tau$ are equal up to (bijective) variable renaming, for any x in X.

Unification after Martelli/Montanari

$$t \approx t, E \quad \rhd_{\mathsf{MM}} \quad E$$

$$f(s_1, \ldots, s_n) \approx f(t_1, \ldots, t_n), E \quad \rhd_{\mathsf{MM}} \quad s_1 \approx t_1, \ldots, s_n \approx t_n, E$$

$$f(\ldots) \approx g(\ldots), E \quad \rhd_{\mathsf{MM}} \quad \bot$$

$$x \approx t, E \quad \rhd_{\mathsf{MM}} \quad x \approx t, E[t/x]$$
$$\text{if } x \in var(E), x \notin var(t)$$

$$x \approx t, E \quad \rhd_{\mathsf{MM}} \quad \bot$$
$$\text{if } x \neq t, x \in var(t)$$

$$t \approx x, E \quad \rhd_{\mathsf{MM}} \quad x \approx t, E$$
$$\text{if } t \notin X$$

MM: Main Properties

A substitution σ is called idempotent, if $\sigma \circ \sigma = \sigma$.

Proposition 5.2 σ is idempotent iff $dom(\sigma) \cap codom(\sigma) = \emptyset$.

If $E = x_1 \approx u_1, \ldots, x_k \approx u_k$, with x_i pw. distinct, $x_i \notin var(u_j)$, then E is called an (equational problem in) solved form representing the solution $\sigma_E = [u_1/x_1, \ldots, u_k/x_k]$.

Proposition 5.3 If E is a solved form then σ_E is am mgu of E.

Theorem 5.4 1. If $E \rhd_{\mathsf{MM}} E'$ then σ unifier of E iff σ unfier of E'

2. If $E \overset{*}{\rhd}_{\mathsf{MM}} \bot$ then E is not unifiable.

3. If $E \overset{*}{\rhd}_{\mathsf{MM}} E'$, with E' a solved form, then $\sigma_{E'}$ is an mgu of E.

Proof. (1) We have to show this for each of the rules. Let's treat the case for the 4th rule here. Suppose σ is a unifier of $x \approx t$, that is, $x\sigma = t\sigma$. Thus, $\sigma \circ [t/x] = \sigma[x \mapsto t\sigma] = \sigma[x \mapsto x\sigma] = \sigma$. Therefore, for any equation $u \approx v$ in E: $u\sigma = v\sigma$, iff $u[t/x]\sigma = v[t/x]\sigma$. (2) and (3) follow by induction from (1) using Proposition 5.3. □

Main Unification Theorem

Theorem 5.5 E unifiable $\Leftrightarrow$ there exists a most general unifier σ of E, such that σ is idempotent and $dom(\sigma) \cup codom(\sigma) \subseteq var(E)$.

Notation: $\sigma = \mathsf{mgu}(E)$

Problem: exponential growth of terms possible

Proof of the Unification Theorem

- Systems E irreducible wrt. $\rhd_{\mathsf{MM}}$ are either $\bot$ or a solved form.
- $\rhd_{\mathsf{MM}}$ is Noetherian. A suitable lexicographic ordering on the multisets E (with $\bot$ minimal) shows this. Compare in this order:
  1. the number of defined variables (d.h. variables x in equations $x \approx t$ with $x \notin var(t)$), which also occur outside their definition elsewhere in E;
  2. the multi-set ordering induced by (i) the size (number of symbols) in an equation; (ii) if sizes are equal consider $x \approx t$ smaller than $t \approx x$, if $t \notin X$.
- Therefore, reducing any E by MM with end (no matter what reduction strategy we apply) in an irreducible E' having the same unifiers as E, and we can read off the mgu (or non-unifiability) of E from E' (Theorem 5.4, Proposition 5.3).
- σ is idempotent because of the substitution in rule 4. $dom(\sigma) \cup codom(\sigma) \subseteq var(E)$, as no new variables are generated.

Deciding Clausal Validity for Finite Trees

The Problem: Let Σ be a signature (w/o predicates) such that T_Σ is infinite. For any given Σ-clause C, decide whether or not $T_\Sigma \models \forall C$

Convexity:

Theorem 5.6 T_Σ is convex, that is, for any Σ-clause $E \vee D$ with equations E and disequations D, whenever $T_\Sigma \models \forall C$ then there exists an equation e in E such that $T_\Sigma \models \forall (D \vee e)$

With this, our problem reduces to the uniform word problem for T_Σ.

Solution: Given $C = E \vee D$ partitioned as indicated into equations E and disequations D, use MM to either show unsatisfiability of $\neg D$ (which is a conjunction of equations) or compute the mgu σ of $\neg D$. Then check whether or not $T_\Sigma \models \forall e\sigma$, for one of the equations e in E. The latter is true if and only if $e\sigma$ is of the form $s \approx s$, for some term s.

Finite Lists

Validity Problem: Decide

$$\mathbb{L} \models \forall C$$

for clauses C over cons, car, cdr, where $\mathbb{L}$ is the canonical model of the axioms L as defined on page 37.

Unification Problem: Given a set E of equations between terms over cons, car, cdr, find a complete set S of solutions σ of E, that is, substitutions σ for which

$$\mathbb{L} \models \forall E\sigma.$$

S is called complete if for each solution σ of E there exists a solution τ in S and a substitution ρ such that for each variable x in E $L \models x\sigma \approx x\tau\rho$. (We say that τ is more general than σ modulo L.)

Unification for Lists

Apply these rules modulo symmetry of $\approx$, AC of ",", and with priority as given by the order. s, t, u, v denote arbitrary terms, x, y, z are variables, N are non-variable terms. Variables z on the right side of rules not occurring on the left are assumed to be fresh. L is the set of rewrite rules for lists (p. 35). We assume that simplification starts with given E and empty R.

Unification for Lists

$$x \approx t, E\,;R \quad \rhd_{\mathsf{L}} \quad E\,;R, x \approx t, \quad \text{if } x \notin \mathit{var}(t)$$

$$x \approx t, E\,;R \quad \rhd_{\mathsf{L}} \quad \bot$$

if $t \neq \mathsf{cons}(u, v)$,

or if x occurs as an argument to a cons in t

$$x \approx \mathsf{cons}(s_1, s_2), E\,;R \quad \rhd_{\mathsf{L}} \quad z_1 = s'_1, z_2 = s'_2, E\,;\ R, x \approx \mathsf{cons}(s'_1, s'_2)$$

if x occurs as an argument to car and/or cdr in $\mathsf{cons}(s_1, s_2)$,

with s'_i denoting the result of replacing $\mathsf{car}(x)$ by z_1 and $\mathsf{cdr}(x)$ by z_2, respectively, in s_i

$$t \approx t, E\,;R \quad \rhd_{\mathsf{L}} \quad E\,;R$$

$$s \approx t, E\,;R \quad \rhd_{\mathsf{L}} \quad u \approx t, E\,;R, \quad \text{if } s \Rightarrow_L u \text{ or } s \Rightarrow_R u$$

$$\mathsf{car}(s) \approx N, E\,;R \quad \rhd_{\mathsf{L}} \quad s \approx \mathsf{cons}(N, z), E\,;R$$

$$\mathsf{cdr}(s) \approx N, E\,;R \quad \rhd_{\mathsf{L}} \quad s \approx \mathsf{cons}(z, N), E\,;R$$

$$\mathsf{cons}(s, t) \approx \mathsf{cons}(u, v), E\,;R \quad \rhd_{\mathsf{L}} \quad s \approx u, t \approx v, E\,;R$$

Properties

Soundness: Each rule preserves the set of solutions. If there are fresh variables on the right side of a rule we mean that each solution of the left side can be extended to a solution of the right side.

Completeness: Upon termination E will be empty and R is a terminating rewrite system of rule of the form $x \approx t$ with just a single rule for each x. Such R represents a substitution and thus a most general solution.

Termination: ???