Automated Reasoning II*

Uwe Waldmann

Winter Term 2011/2012

Topics of the Course

Decision procedures

equality (congruence closure)
algebraic theories
combinations

Satisfiability modulo theories (SMT)

DPLL(T)
dealing with universal quantification

Superposition

combining ordered resolution and completion
optimizations
integrating theories

1 Decision Procedures

In general, validity (or unsatisfiability) of first-order formulas is undecidable.

To get decidability results, we have to consider restricted classes of formulas.

*This document contains the text of the lecture slides (almost verbatim) plus some additional information, mostly proofs of theorems that are presented on the blackboard during the course. It is not a full script and does not contain the examples and additional explanations given during the lecture. Moreover it should not be taken as an example how to write a research paper – neither stylistically nor typographically.

1.1 Theories and Fragments

So far, we have considered the validity or satisfiability of "unstructured" sets of formulas.

We will now split these sets of formulas into two parts: a theory (which we keep fixed) and a set of formulas that we consider relative to the theory.

A *first-order theory* $\mathcal{T}$ is defined by

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

its axioms, that is, a set of closed Σ-formulas.

(We often use the same symbol $\mathcal{T}$ for a theory and its set of axioms.)

Note: This is the *syntactic view* of theories. There is also a *semantic view*, where one specifies a class of Σ-algebras $\mathcal{M}$ and considers $Th(\mathcal{M})$, that is, all closed Σ-formulas that hold the algebras of $\mathcal{M}$.

A Σ-algebra that satisfies all axioms of $\mathcal{T}$ is called a *$\mathcal{T}$-algebra* (or *$\mathcal{T}$-interpretation*).

$\mathcal{T}$ is called *consistent* if there is at least one $\mathcal{T}$-algebra. (We will only consider consistent theories.)

We can define models, validity, satisfiability, entailment, equivalence, etc., relative to a theory $\mathcal{T}$:

A $\mathcal{T}$-algebra that is a model of a Σ-formula F is also called a $\mathcal{T}$-model of F.

A Σ-formula F is called $\mathcal{T}$-valid, if $\mathcal{A}, \beta \models F$ for all $\mathcal{T}$-algebras $\mathcal{A}$ and assignments β.

A Σ-formula F is called $\mathcal{T}$-satisfiable, if $\mathcal{A}, \beta \models F$ for some $\mathcal{T}$-algebra and assignment β (and otherwise $\mathcal{T}$-unsatisfiable).

($\mathcal{T}$-satisfiability of sets of formulas, $\mathcal{T}$-entailment, $\mathcal{T}$-equivalence: analogously.)

A *fragment* is some syntactically restricted class of Σ-formulas.

Typical restriction: only certain quantifier prefixes are permitted.

1.2 Equality

Theory of equality:

Signature: arbitrary

Axioms: none
(but the equality predicate $\approx$ has a fixed interpretation)

Alternatively:

Signature contains a binary predicate symbol $\sim$ instead of the built-in $\approx$

Axioms: reflexivity, symmetry, transitivity, congruence for $\sim$

In general, satisfiability of first-order formulas w. r. t. equality is undecidable.

However, we will show that it is decidable for *ground* first-order formulas.

Note: It suffices to consider conjunctions of literals. Arbitrary ground formulas can be converted into DNF; a formula in DNF is satisfiable if and only if one of its conjunctions is satisfiable.

Note that our problem can be written in several ways:

An equational clause
$\forall \vec{x}\,(A_1 \vee \ldots \vee A_n \vee \neg B_1 \vee \ldots \vee \neg B_k)$ is $\mathcal{T}$-valid

iff

$\exists \vec{x}\,(\neg A_1 \wedge \ldots \wedge \neg A_n \wedge B_1 \wedge \ldots \wedge B_k)$ is $\mathcal{T}$-unsatisfiable

iff

the Skolemized (ground!) formula
$(\neg A_1 \wedge \ldots \wedge \neg A_n \wedge B_1 \wedge \ldots \wedge B_k)\{\vec{x} \mapsto \vec{c}\}$ is $\mathcal{T}$-unsatisfiable

iff

$(A_1 \vee \ldots \vee A_n \vee \neg B_1 \vee \ldots \vee \neg B_k)\{\vec{x} \mapsto \vec{c}\}$ is $\mathcal{T}$-valid

Other names:

The theory is also known as *EUF* (equality with uninterpreted function symbols).

The decision procedures for the ground fragment are called *congruence closure* algorithms.

Congruence Closure

Goal: check (un-)satisfiability of a ground conjunction

$$u_1 \approx v_1 \wedge \ldots \wedge u_n \approx v_n \wedge \neg\, s_1 \approx t_1 \wedge \ldots \wedge \neg\, s_k \approx t_k$$

Idea:

transform $E = \{u_1 \approx v_1,\, \ldots,\, u_n \approx v_n\}$ into an equivalent convergent TRS R and check whether $s_i{\downarrow_R} = t_i{\downarrow_R}$.

if $s_i{\downarrow_R} = t_i{\downarrow_R}$ for some i:
$s_i{\downarrow_R} = t_i{\downarrow_R} \;\Leftrightarrow\; s_i \leftrightarrow_E^* t_i \;\Leftrightarrow\; E \models s_i \approx t_i \;\Rightarrow\;$ unsat.

if $s_i{\downarrow_R} = t_i{\downarrow_R}$ for no i:
$\mathrm{T}_\Sigma(X)/R = \mathrm{T}_\Sigma(X)/E$ is a model of the conjunction $\Rightarrow$ sat.

In principle, one could use Knuth-Bendix completion to convert E into an equivalent convergent TRS R.

If done properly (see exercises), Knuth-Bendix completion terminates for ground inputs.

However, for the ground case, one can optimize the general procedure.

First step:

Flatten terms: Introduce new constant symbols $c_1, c_2, \ldots$ for all subterms.

$g(a, h(h(b))) \approx h(a)$ is replaced by $a \approx c_1 \wedge b \approx c_2 \wedge h(c_2) \approx c_3 \wedge h(c_3) \approx c_4 \wedge g(c_1, c_4) \approx c_5 \wedge h(c_1) \approx c_6 \wedge c_5 \approx c_6$

Result: only two kinds of equations left.
D-equations: $f(c_{i_1}, \ldots, c_{i_n}) \approx c_{i_0}$ for $f/n \in \Omega$, $n \geq 0$.
C-equations: $c_i \approx c_j$.

$\Rightarrow$ efficient indexing (e. g., using hash tables),
obvious termination for D-equations.

Inference Rules

The congruence closure algorithm is presented as a set of inference rules working on a set of equations E and a set of rules R: $E_0, R_0 \vdash E_1, R_1 \vdash E_2, R_2 \vdash \ldots$

At the beginning, $E = E_0$ is the set of C-equations and $R = R_0$ is the set of D-equations oriented left-to-right. At the end, E should be empty; then R is the result.

Notation: The formula $s \mathrel{\dot{\approx}} t$ denotes either $s \approx t$ or $t \approx s$.

Simplify:

$$\frac{E \cup \{c \mathrel{\dot{\approx}} c'\}, \quad R \cup \{c \to c''\}}{E \cup \{c'' \mathrel{\dot{\approx}} c'\}, \quad R \cup \{c \to c''\}}$$

Delete:

$$\frac{E \cup \{c \approx c\}, \quad R}{E, \quad R}$$

Orient:

$$\frac{E \cup \{c \mathrel{\dot{\approx}} c'\}, \quad R}{E, \quad R \cup \{c \to c'\}} \qquad \text{if } c \succ c'$$

Deduce:

$$\frac{E, \quad R \cup \{t \to c,\, t \to c'\}}{E \cup \{c \approx c'\}, \quad R \cup \{t \to c\}}$$

Collapse:

$$\frac{E, \quad R \cup \{t[c] \to c',\, c \to c''\}}{E, \quad R \cup \{t[c''] \to c',\, c \to c''\}}$$

Note: for ground rewrite rules, critical pair computation does not involve substitution. Therefore, every critical pair computation can be replaced by a simplification, either using Deduce or Collapse.

Strategy

The inference rules are applied according to the following strategy:

(1) If there is an equations in E, use Simplify as long as possible for this equation, then use either Delete or Orient. Repeat until E is empty.

(2) If Collapse is applicable, apply it, if now Deduce is applicable, apply it as well. Repeat until Collapse is no longer applicable.

(3) If E is non-empty, go to (1), otherwise return R.

Implementation

Instead of fixing the ordering $\succ$ in advance, it is preferable to define it on the fly during the algorithm:

If we orient an equation $c \approx c'$ between two constant symbols, we try to make that constant symbol larger that occurs less often in $R \Rightarrow$ fewer Collapse steps.

Additionally:

Use various index data structures so that all the required operations can be performed efficiently.

Use a union-find data structure to represent the equivalence classes encoded by the C-rules.

Average runtime: $O(m \log m)$, where m is the number of edges in the graph representation of the initial C and D-equations.

Other Predicate Symbols

If the initial ground conjunction contains also non-equational literals $[\neg]\, P(t_1, \dots, t_n)$, treat these like equational literals $[\neg]\, P(t_1, \dots, t_n) \approx \mathit{true}$. Then use the same algorithm as before.

One Small Problem

The inference rules are sound in the usual sense: The conclusions are entailed by the premises, so every $\mathcal{T}$-model of the premises is a $\mathcal{T}$-model of the conclusions.

For the initial flattening, however, we get a weaker result: We have to *extend* the $\mathcal{T}$-models of the original equations to obtain models of the flattened equations. That is, we get a new algebra with the same universe as the old one, with the same interpretations for old functions and predicate symbols, but with appropriately chosen interpretations for the new constants.

Consequently, the relations $\approx_E$ and $\approx_R$ for the original E and the final R are not the same. For instance, $c_3 \approx_E c_7$ does not hold, but $c_3 \approx_R c_7$ may hold.

On the other hand, the model extension preserves the universe and the interpretations for old symbols. Therefore, if s and t are terms over the old symbols, we have $s \approx_E t$ iff $s \approx_R t$.

This is sufficient for our purposes: The terms s_i and t_i that we want to normalize using R do not contain new symbols.

History

Congruence closure algorithms have been published, among others, by Shostak (1978). by Nelson and Oppen (1980), and by Downey, Sethi and Tarjan (1980).

Kapur (1997) showed that Shostak's algorithm can be described as a completion procedure.

Bachmair and Tiwari (2000) did this also for the Nelson/Oppen and the Downey/Sethi/Tarjan algorithm.

The algorithm presented here is the Downey/Sethi/Tarjan algorithm in the presentation of Bachmair and Tiwari.