Building Decision Procedures for Data Structures

Silvio Ranise and Christophe Ringeissen

LORIA

Lecture 4

Outline

Satisfiability Procedures for Equality

- Aka theory of uninterpreted function (UF) symbols
- Useful in virtually any verification problem
  - ▷ uninterpreted function symbols provide a natural means for abstracting data and data operations
  - ▷ hardware, software, safety checking, ...

Axiom schemas for the theory of UF

- Equality can be defined as a binary predicate $=$ written infix satisfying the following axioms:

$$\forall x.(x = x) \qquad \textit{reflexivity}$$

$$\forall x, y.(x = y \Rightarrow y = x) \qquad \textit{symmetry}$$

$$\forall x, y, z.(x = y \wedge y = z \Rightarrow x = z) \qquad \textit{transitivity}$$

$$\forall x_1, y_1, \ldots, x_n, y_n.(\bigwedge_{i=1}^{n} x_i = y_i \Rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)) \qquad \textit{congruence}$$

- Note: congruence is an axiom schema since it must be instantiated for each function symbol f in the formula

Decision Procedure for the full theory of UF

Superpos$_1$	$$\frac{c = c' \qquad c = d}{c' = d}$$	if $c \succ c'$, $c \succ d$
Superpos$_2$	$$\frac{c_j = c'_j \qquad f(c_1, \ldots, c_j, \ldots, c_n) = c_{n+1}}{f(c_1, \ldots, c'_j, \ldots, c_n) = c_{n+1}}$$	if $c_j \succ c'_j$
Superpos$_3$	$$\frac{f(c_1, \ldots, c_n) = c'_{n+1} \qquad f(c_1, \ldots, c_n) = c_{n+1}}{c_{n+1} = c'_{n+1}}$$	
Paramodul	$$\frac{c = c' \qquad c \neq d}{c' \neq d}$$	if $c \succ c'$, $c \succ d$
Eq. Res.	$$\frac{c \neq c}{\bot}$$	

Notice that we only need to compare constants!

Decision Procedure for the full theory of UF: Summary

- Flatten literals
- Exhaustive application of the rules in the previous slide
  - ▷ if $\perp$ is derived, then unsatisfiability is reported
  - ▷ if $\perp$ is not derived and no more rule can be applied, then satisfiability is reported

Can we extend the approach to other theories?

- Yes, but using more general concepts:
  - ▷ rewriting on arbitrary terms (not only constants)
  - ▷ considering arbitrary clauses since many interesting

theories are axiomatized by formulae which are more complex than simple equalities or disequalities, e.g. the theory of lists:

$$\begin{aligned} \text{car}(\text{cons}(X, Y)) &= X \\ \text{cdr}(\text{cons}(X, Y)) &= Y \end{aligned}$$

where X, Y are implicitly universally quantified variables

Our goal

- **Given**
  - ▷ a presentation of a theory T extending UF
    (Notice that T is **not restricted** to equations*!*)
- **We want to derive**
  - ▷ a satisfiability decision procedure capable of establishing whether S is T-satisfiable, i.e. $S \cup T$ is satisfiable (where S is a set of *ground literals*)

Our approach to the problem

- Based on the **rewriting approach**
  - ▷ uniform and simple
  - ▷ efficient alternative to the congruence closure approach
- **Tune** a general (off-the-shelf)

refutation complete superposition inference system

(from, e.g. [Rus91,BacGan94]) in order to obtain

termination

on some interesting theories

First step: flatten

- The first step is to flatten all the input literals by extending the signature introducing “fresh” constants
- ***Example***: $\{f(c, c') = h(h(a)), h(h(h(a))) \neq a\}$ is flattened to

$$\{f(c, c') = h(c_1), c_3 \neq a\} \cup \{c_1 = h(a), c_3 = h(c_2), c_2 = h(c_1)\}$$

Fact

Let S be a finite set of Σ-literals. Then there exists a finite set of flat Σ'-literals S' (where Σ' is obtained from Σ by adding a finite number of constants) such that S' is T-satisfiable iff S is.

Second step: apply superposition calculus $\mathcal{SP}$

A calculus manipulating clauses (disjunctions of literals):
$(s_1 \neq t_1 \vee \cdots \vee s_k \neq t_k) \vee (s_{k+1} = t_{k+1} \vee \cdots \vee s_m = t_m)$
also written $s_1 = t_1, \ldots, s_k = t_k \to s_{k+1} = t_{k+1}, \ldots, s_m = t_m$

- **Inference rules:** Superposition, Paramodulation, Reflection, Factoring
- **Simplification rules:** Subsumption, Simplification, Deletion
- **Reduction ordering** $\succ$ (total on ground terms)
- **Refutation complete**: any fair application of the rules to an unsatisfiable set of clauses will derive the empty clause
- **Saturation** of a set of clauses is the final set of clauses generated by a fair derivation
- A derivation is **fair** when all possible inferences are performed

See below for formal definitions of all these concepts!

Superposition Calc. (Unit Clauses, Expansion Rules)

Superposition	$\dfrac{l[u'] = r \quad u = t}{\sigma(l[t] = r)}$	$(i), (ii)$
Paramodulation	$\dfrac{l[u'] \neq r \quad u = t}{\sigma(l[t] \neq r)}$	$(i), (ii)$
Reflection	$\dfrac{u' \neq u}{\square}$	

where the substitution σ is the most general unifier of u and u' (i.e., $\sigma(u') = \sigma(u)$), u' is not a variable and the following conditions hold:

(i) $\sigma(u) \not\preceq \sigma(t)$

(ii) $\sigma(l[u']) \not\preceq \sigma(r)$

Figure: Expansion Rules of $\mathcal{SP}$

Replacement of equal by equal performed up to **unification**
Rules controlled by a **simplification ordering on terms**

Superposition Calc. (Unit Clauses, Contraction Rules)

Name	Rule	Conditions
Subsumption	$\dfrac{S \cup \{L, L'\}}{S \cup \{L\}}$	for some θ, $\theta(L) = L'$
Simplification	$\dfrac{S \cup \{L[\theta(l)], l = r\}}{S \cup \{L[\theta(r)], l = r\}}$	$\theta(l) \succ \theta(r)$, $L[\theta(l)] \succ (\theta(l) = \theta(r))$
Deletion	$\dfrac{S \cup \{t = t\}}{S}$	

Orderings

- Requirement: $f(c_1, \ldots, c_n) \succ c_0$
for each non-constant symbol f and constant c_i ($i = 0, 1, ..., n$)
- [Definition:] $(a = b) \succ (c = d)$ iff $\{a, b\} \succ\!\succ \{c, d\}$
(where $\succ\!\succ$ is the multiset extension of $\succ$ on terms)
- multisets of literals are compared by the multiset extension of $\succ$ on literals
- clauses are considered as multisets of literals
- **Intuition**: the ordering $\succ$ is such that only maximal sides of maximal instances of literals are involved in inferences

Ordering: Definitions

Definition (Well-founded Ordering)

$>$ is *well-founded* if there is no infinite decreasing chain $t_1 > t_2 > \ldots$

Definition (Reduction Ordering)

$>$ is a *reduction ordering* if

- $>$ is *well-founded*,
- For any terms s, t and context u, $s > t$ implies $u[s] > u[t]$,
- For any terms s, t and substitution σ, $s > t$ implies $\sigma(s) > \sigma(t)$,

Well-Founded Ordering: Multiset Extension

Definition (Multiset Extension)

$M >^{mult} N$ if $M \neq N$ and
$N(t) > M(t) \Rightarrow \exists t' : t' > t$ and $M(t') > N(t')$

Fact: The multiset extension of a well-founded ordering is well-founded.

Example (Multiset set extension of the ordering on Naturals)

$\{3, 3, 3, 2, 1\} >^{mult} \{3, 3, 2, 2, 2, 1\}$
$\{3, 3, 1, 2\} >^{mult} \{1, 1, 2\}$

Well-Founded Ordering: LPO

Example (Lexicographic Path Ordering)

$s = f(s_1, \ldots, s_n) >_{lpo} g(t_1, \ldots, t_m) = t$ if

1. $f = g$ and $(s_1, \ldots, s_n) >^{lex}_{lpo} (t_1, \ldots, t_m)$ and $\forall j \in \{1, \ldots, m\}\ \ s >_{lpo} t_j$
2. $f >_{\mathcal{F}} g$ and $\forall j \in \{1, \ldots, m\}\ \ s >_{lpo} t_j$
3. $\exists i \in \{1, \ldots, n\}$ such that either $s_i >_{lpo} t$ or $s_i = t$

Remarks:

- The lexicographic extension $>^{lex}$ is defined as follows: $(s_1, \ldots, s_n) >^{lex} (t_1, \ldots, t_n)$ if there exists some $i \in [1, n]$ such that $s_i > t_i$ and for any j smaller than i, $s_j = t_j$. The ordering $>^{lex}$ is well-founded if $>$ is well-founded.
- LPO is a simplification ordering: for any term s and any context u, $u[s] > s$
- LPO is total on ground terms

Reduction Ordering: Exercise

Termination of Ackermann Function

$$\begin{array}{lcl} Ack(0,y) & \to & s(y) \\ Ack(s(x),0) & \to & Ack(x,s(0)) \\ Ack(s(x),s(y)) & \to & Ack(x,Ack(s(x),y)) \end{array}$$

With LPO? Which precedence to choose?
Let $Ack > s > 0$
$Ack(0,y) > s(y)$ since $Ack(0,y) > y$

$Ack(s(x),0) > Ack(x,s(0))$ since $s(x) > x$ and $Ack(s(x),0) > x$ and ($Ack(s(x),0) > s(0)$ since $Ack(s(x),0) > 0$)

$Ack(s(x),s(y)) > Ack(x,Ack(s(x),y))$ since $s(x) > x$ and
$Ack(s(x),s(y)) > x$ and ($Ack(s(x),s(y)) > Ack(s(x),y)$ since $s(y) > y$ and
$Ack(s(x),s(y)) > s(x)$ and $Ack(s(x),s(y)) > y$)

Syntactic Unification

Problem

Given two terms s and t, is there a substitution σ such that $\sigma(s)$ and $\sigma(t)$ are identical?

The substitution σ is called a **unifier** of s and t, equivalently it is a solution of the unification problem $s =^? t$.

In general, a unification problem P is a conjunction of equations $s_1 =^? t_1 \wedge \cdots \wedge s_n =^? t_n$, and a unifier σ of P is a substitution such that $\sigma(s_i)$ and $\sigma(t_i)$ are identical for all $i = 1, \ldots, n$.

Fact

If a unification problem admits a solution, then there exists a **most general unifier** μ such that any unifier σ is an instance of μ.

Example

$x =^? f(a, y)$ has a unifier $\sigma = \{x \mapsto f(a, a), y \mapsto a\}$ but σ is an instance of $\mu = \{x \mapsto f(a, y)\}$.

Rules for syntactic unification (computation of mgu)

Delete	$P \wedge s =^? s$	$\mapsto\!\!\!\twoheadrightarrow$	P
Decompose	$P \wedge f(s_1, \ldots, s_n) =^? f(t_1, \ldots, t_n)$	$\mapsto\!\!\!\twoheadrightarrow$	$P \wedge s_1 =^? t_1 \wedge \ldots \wedge s_n =^? t_n$
Conflict	$P \wedge f(s_1, \ldots, s_n) =^? g(t_1, \ldots, t_p)$	$\mapsto\!\!\!\twoheadrightarrow$	$\bot$ if $f \neq g$
Coalesce	$P \wedge x =^? y$	$\mapsto\!\!\!\twoheadrightarrow$	$\{x \mapsto y\}(P) \wedge x =^? y$ if $x, y \in Var(P)$ *and* $x \neq y$
Check*	$P \wedge x_1 =^? s_1[x_2] \ldots \wedge x_n =^? s_n[x_1]$	$\mapsto\!\!\!\twoheadrightarrow$	$\bot$ if $s_i \notin Var$ for some $i \in [1..n]$
Merge	$P \wedge x =^? s \wedge x =^? t$	$\mapsto\!\!\!\twoheadrightarrow$	$P \wedge x =^? s \wedge s =^? t$ if $0 < \lvert s \rvert \leq \lvert t \rvert$
Check	$P \wedge x =^? s$	$\mapsto\!\!\!\twoheadrightarrow$	$\bot$ if $x \in Var(s)$ and $s \notin Var$
Eliminate	$P \wedge x =^? s$	$\mapsto\!\!\!\twoheadrightarrow$	$\{x \mapsto s\}(P) \wedge x =^? s$ if $x \notin Var(s), s \notin Var, x \in Var(P)$

Examples

$$x =^? a$$
$$x =^? a \wedge y =^? f(x, a)$$
$$f(x, f(x, a)) =^? f(f(a, b), f(u, v))$$
$$x =^? a \wedge x =^? b$$

Tree Solved form

A *tree solved form* for P is any conjunction Q of equations

$$x_1 =^? t_1 \wedge \cdots \wedge x_n =^? t_n$$

equivalent to P such that for any $i = 1, \ldots, n$, x_i is a variable occurring only once in Q.
Example: $x =^? f(f(y)) \wedge z =^? g(a)$

Computation of mgu

Theorem

Starting with a unification problem P and using the above rules repeatedly until none is applicable
— results in $\perp$ iff P has no solution, or else it
— results in a tree solved form $x_1 =^? t_1 \wedge \cdots \wedge x_n =^? t_n$ for P, with the same set of solutions than P.
Moreover

$$\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$$

is a most general unifier of P, denoted by mgu(P).

Redundancy and Saturation

Definition

- A clause C is *redundant* with respect to a set S of clauses if S can be obtained from $S \cup \{C\}$ by a sequence of applications of contraction rules in $\mathcal{SP}$.
- An inference in $\mathcal{SP}$ is *redundant* with respect to a set S of clauses if its conclusion is redundant with respect to S.
- A set S of clauses is *saturated* if every inference in $\mathcal{SP}$ with premises in S is redundant with respect to S.

Fair derivation

Definition

- A *derivation* is a sequence $S_0, S_1, \ldots, S_i, \ldots$ of sets of clauses where $S_i \Rightarrow_{\mathcal{SP}} S_{i+1}$ via the application of expansion rules or contraction rules in $\mathcal{SP}$.
- The *limit* of a derivation is defined as the set of persistent clauses $S_\infty = \bigcup_{j \geq 0} \bigcap_{i > j} S_i$.
- A derivation $S_0, S_1, ..., S_i, ...$ with limit S_∞ is *fair* if every inference in $\mathcal{SP}$ with premises in S_∞ is redundant with respect to some S_j.

Refutation Completeness

Fair derivations compute saturated sets and generate the empty clause iff the initial set is unsatisfiable.

Theorem (Nieuwenhuis-Rubio)

If $S_0, S_1, \ldots$ is a fair derivation of $\mathcal{SP}$, then (i) its limit S_∞ is saturated with respect to $\mathcal{SP}$, (ii) S_0 is unsatisfiable iff the empty clause is in S_j for some j, and (iii) if such a fair derivation is finite, i.e. it is of the form $S_0, \ldots, S_n$, then S_n is saturated and logically equivalent to S_0.

Problem: For which theories do we have finite fair derivations?

Example: SP for lists (I)

- Consider the following (simplified) theory of lists

$$Ax(\mathcal{L}) := \{\mathrm{car}(\mathrm{cons}(X, Y)) = X, \mathrm{cdr}(\mathrm{cons}(X, Y)) = Y\}$$

- Recall that a literal in S has one of the four possible forms:
  - (i) $\mathrm{car}(c) = d$,
  - (ii) $\mathrm{cdr}(c) = d$,
  - (iii) $\mathrm{cons}(c_1, c_2) = d$,
  - (iv) $c \neq d$.
- There are three cases to consider:

1. inferences between two clauses in S
2. inferences between two clauses in $Ax(\mathcal{L})$
3. inferences between a clause in $Ax(\mathcal{L})$ and a clause in S

Example: SP for lists (II)

- Case 1: inferences between two clauses in S

It has already been considered when considering equality only (please, keep in mind this point)

- Case 2: inferences between two clauses in $Ax(\mathcal{L})$

This is not very interesting since there are no possible inferences between the two axioms in $Ax(\mathcal{L})$

- Case 3: inferences between a clause in $Ax(\mathcal{L})$ and a clause in S

  ▷ a superposition between $\text{car}(\text{cons}(X, Y)) = X$ and $\text{cons}(c_1, c_2) = d$ yielding $\text{car}(d) = c_1$ and

  ▷ a superposition between $\text{cdr}(\text{cons}(X, Y)) = Y$ and $\text{cons}(c_1, c_2) = d$ yielding $\text{cdr}(d) = c_2$

Example: SP for lists (III)

- We are almost done, it is sufficient to notice that
  - ▷ only finitely many equalities of the form (i) and (ii) can be generated this way out of a set of clauses built on a finite signature
  - ▷ so, we are entitled to conclude that $\mathcal{SP}$ can only generate finitely many clauses on set of clauses of the form $Ax(\mathcal{L}) \cup S$
- A decision procedure for the satisfiability problem of $\mathcal{L}$ can be built by simply using $\mathcal{SP}$ after flattening the input set of literals

Deriving a Decision Procedure for Arrays (I)

$$Ax(\mathcal{A}) := \left\{ \begin{array}{l} \mathrm{rd}(\mathrm{wr}(A, I, E), I) = E \\ I = J \vee \mathrm{rd}(\mathrm{wr}(A, I, E), J) = \mathrm{rd}(A, J) \end{array} \right\}$$

Apply the methodology previously described using a superposition calculus handling arbitrary clauses

$\mathcal{SP}$ (Arbitrary Clauses, Expansion Rules)

Sup.	$\dfrac{C \vee l[u'] = r \quad D \vee u = t}{\sigma(C \vee D \vee l[t] = r)}$	$(i), (ii), (iii), (iv)$
Par.	$\dfrac{C \vee l[u'] \neq r \quad D \vee u = t}{\sigma(C \vee D \vee l[t] \neq r)}$	$(i), (ii), (iii), (iv)$
Ref.	$\dfrac{C \vee u' \neq u}{\sigma(C)}$	$\forall L \in C.\ \sigma(u' = u) \not\prec \sigma(L)$
Fac.	$\dfrac{C \vee u = t \vee u' = t'}{\sigma(C \vee t \neq t' \vee u = t')}$	$(i), \forall L \in \{u' = t'\} \cup C.\ \sigma(u = t) \not\prec \sigma(L)$

where the substitution $\sigma = mgu(u =^? u')$, u' is not a variable, and the following conditions hold:

(i) $\sigma(u) \not\preceq \sigma(t)$

(ii) $\forall L \in D.\ \sigma(u = t) \not\preceq \sigma(L)$

(iii) $\sigma(l[u']) \not\preceq \sigma(r)$

(iv) $\forall L \in C.\ \sigma(l[u'] \bowtie r) \not\preceq \sigma(L)$

Figure: Expansion Rules of $\mathcal{SP}$

$\mathcal{SP}$ (Arbitrary Clauses, Contraction Rules)

Name	Rule	Conditions
Subsumption	$\dfrac{S \cup \{C, C'\}}{S \cup \{C\}}$	for some θ, $\theta(C) \subseteq C'$, and there is no ρ s.t. $\rho(C') = C$
Simplification	$\dfrac{S \cup \{C[\theta(l)], l = r\}}{S \cup \{C[\theta(r)], l = r\}}$	$\theta(l) \succ \theta(r)$, $C[\theta(l)] \succ (\theta(l) = \theta(r))$
Deletion	$\dfrac{S \cup \{C \vee t = t\}}{S}$	

Deriving a Decision Procedure for Arrays (II)

Lemma

Let S be a finite set of flat literals. The clauses occurring in the saturations of $S \cup Ax(\mathcal{A})$ by $\mathcal{SP}$ can only be:

i) *the empty clause;* ii) *axioms* iii) *ground flat literals*

iv) *clauses of type* $t \bowtie t' \vee c_1 = c'_1 \vee \cdots \vee c_n = c'_n$ *with* $t \bowtie t' \in \{c \neq c', \mathrm{rd}(c, i) = c', \mathrm{rd}(c, i) = \mathrm{rd}(c', i')\}$

v) *clauses of type* $\mathrm{rd}(c, x) = \mathrm{rd}(c', x) \vee c_1 = k_1 \vee \cdots \vee c_n = k_n$, *where* k_i *is either* x *or a constant among* $c, c_1, \ldots, c_n$

where $i, c, c', c_1, c'_1, \ldots, c_n, c'_n$ *are constants, and* x *is a variable.*

Lemma

The saturations of $S \cup Ax(\mathcal{A})$ are finite

Rewriting approach: drawbacks

- Unfortunately not all theories are finitely axiomatized
  - ▷ Example: the usual theory of arithmetic does not admit a finite axiomatization
- Because of this and the ubiquity of arithmetic in practically any verification problem jointly with equational theories, we need to combine the satisfiability procedure provided by $\mathcal{SP}$ with a satisfiability procedure for the theory of arithmetic

References on a rewriting approach to sat proc

- Armando, Ranise, Rusinowitch. "*A Rewriting Approach to Satisfiability Procedures*," Information and Computation, 183(2):140–164, June 2003. (Extended version of CSL'01).
- Armando, Bonacina, Ranise, Schulz. "*On a rewriting approach to satisfiability procedures: extension, combination of theories, and an experimental apprisal*," presented at FroCos'05, Vienna. (Experimental evaluation.)
- Nieuwenhuis, Rubio. "*Paramodulation-based Theorem Proving*," Handbook of Automated Reasoning, Volume 1, Chapter 7, pages 371–444.