Building Decision Procedures for Data Structures

Silvio Ranise and Christophe Ringeissen

LORIA

Lecture 4
Outline

1. Use of Superposition
   - Equality
   - Extensions of Equality

2. Superposition: Unit Clauses
   - Orderings
   - Unification
   - Saturation

3. Superposition: Arbitrary Clauses

4. References
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, ...

Silvio Ranise and Christophe Ringeissen

Building Decision Procedures for Data Structures
Axiom schemas for the theory of UF

- **Equality** can be defined as a binary predicate $= \text{ written infix}$ satisfying the following axioms:

\[
\forall x. (x = x) \quad \text{reflexivity}
\]
\[
\forall x, y. (x = y \Rightarrow y = x) \quad \text{symmetry}
\]
\[
\forall x, y, z. (x = y \land y = z \Rightarrow x = z) \quad \text{transitivity}
\]
\[
\forall x_1, y_1, \ldots, x_n, y_n. (\bigwedge_{i=1}^{n} x_i = y_i \Rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \quad \text{congruence}
\]

- **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$_1$
\[
\frac{c = c'}{c' = d} \quad \frac{c = d}{c' = d}
\]
if $c \succ c'$, $c \succ d$

Superpos$_2$
\[
\frac{c_j = c_j'}{f(c_1, \ldots, c_j, \ldots, c_n) = c_{n+1}}
\]
\[
f(c_1, \ldots, c_j', \ldots, c_n) = c_{n+1}
\]
if $c_j \succ c_j'$

Superpos$_3$
\[
\frac{f(c_1, \ldots, c_n) = c_{n+1}}{c_{n+1} = c'_{n+1}}
\]
\[
\frac{f(c_1, \ldots, c_n) = c_{n+1}}{c_{n+1} = c'_{n+1}}
\]

Paramodul
\[
\frac{c = c'}{c' \neq d}
\]
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 \( \bot \) is derived, then unsatisfiability is reported
  - if \( \bot \) 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:

\[
\begin{align*}
\text{car}(\text{cons}(X, Y)) &= X \\
\text{cdr}(\text{cons}(X, Y)) &= Y
\end{align*}
\]

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 \( \Sigma \)-literals. Then there exists a finite set of flat \( \Sigma' \)-literals \( S' \) (where \( \Sigma' \) is obtained from \( \Sigma \) by adding a finite number of constants) such that \( S' \) is \( T \)-satisfiable iff \( S \) is.
Second step: apply superposition calculus $SP$

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

- **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: \[ l[u'] = r \quad u = t \]
\[
\sigma(l[t] = r) 
\]

(i), (ii), (iii), (iv)

Paramodulation: \[ l[u'] \neq r \quad u = t \]
\[
\sigma(l[t] \neq r) 
\]

(i), (ii), (iii), (iv)

Reflection: \[ u' \neq u \]
\[
\Box 
\]

(i)

where (i) \( \sigma \) is the most general unifier of \( u \) and \( u' \), (ii) \( u' \) is not a variable, (iii) \( u \sigma \not\preceq t \sigma \), (iv) \( l[u'] \sigma \not\preceq r \sigma \).

**Figure:** Expansion Rules of \( SP \)

Replacement of equal by equal performed up to **unification**
Rules controlled by a **simplification ordering** on terms
### Superposition Calc. (Unit Clauses, Contraction Rules)

<table>
<thead>
<tr>
<th>Name</th>
<th>Rule</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subsumption</strong></td>
<td>$S \cup {L, L'}$</td>
<td>$S \cup {L}$ for some $\theta$, $\theta(L) = L'$</td>
</tr>
<tr>
<td></td>
<td>$S \cup {L}$</td>
<td></td>
</tr>
<tr>
<td><strong>Simplification</strong></td>
<td>$S \cup {L[\theta(l)], l = r}$</td>
<td>$\theta(l) \succ \theta(r)$, $L[\theta(l)] \succ (\theta(l) = \theta(r))$</td>
</tr>
<tr>
<td></td>
<td>$S \cup {L[\theta(r)], l = r}$</td>
<td></td>
</tr>
<tr>
<td><strong>Deletion</strong></td>
<td>$S \cup {t = t}$</td>
<td>$S$</td>
</tr>
</tbody>
</table>

The table above summarizes the rules for Superposition Calculus (Unit Clauses) and their associated conditions.
• Requirement: \( f(c_1, \ldots, c_n) \succ c_0 \)
for each non-constant symbol \( f \) and constant \( c_i \) \((i = 0, 1, \ldots, n)\)
Ordering: Definitions

**Definition (Well-founded Ordering)**

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

**Definition (Multiset Extension)**

$M >^{\text{mult}} N$ 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)**

\[
\{3, 3, 3, 2, 1\} >^{\text{mult}} \{3, 3, 2, 2, 2, 1\} \\
\{3, 3, 1, 2\} >^{\text{mult}} \{1, 1, 2\}
\]
Definition (Reduction Ordering)

\( > \) is a \textit{reduction ordering} if

- \( > \) is \textit{well-founded},
- For any terms \( s, t \) and context \( u \), \( s > t \) implies \( u[s] > u[t] \),
- For any terms \( s, t \) and substitution \( \sigma \), \( s > t \) implies \( \sigma(s) > \sigma(t) \),
### Reduction Ordering: Example

**Example (Lexicographic Path Ordering)**

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

1. \( f = g \) and \( (s_1, \ldots, s_n) >_{lpo}^{lex} (t_1, \ldots, t_m) \) and \( \forall j \in \{1, \ldots, m\} \ s >_{lpo} t_j \)
2. \( f >_F g \) 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 of a well-founded ordering 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
Termination of Ackermann Function

\[
\begin{align*}
    \text{Ack}(0, y) & \rightarrow s(y) \\
    \text{Ack}(s(x), 0) & \rightarrow \text{Ack}(x, s(0)) \\
    \text{Ack}(s(x), s(y)) & \rightarrow \text{Ack}(x, \text{Ack}(s(x), y))
\end{align*}
\]

with LPO? which precedence to choose?
## Rules for syntactic unification (computation of mgu)

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete</td>
<td>( P \land s =? s ) [\mapsto] ( P )</td>
<td>( P \land s_1 =? t_1 \land \ldots \land s_n =? t_n ) [\mapsto] ( P )</td>
</tr>
<tr>
<td>Decompose</td>
<td>( P \land f(s_1, \ldots, s_n) =? f(t_1, \ldots, t_n) ) [\mapsto] ( P \land s_1 =? t_1 \land \ldots \land s_n =? t_n )</td>
<td>( P \land f(s_1, \ldots, s_n) =? g(t_1, \ldots, t_p) ) [\mapsto] ( \bot ) if ( f \neq g )</td>
</tr>
<tr>
<td>Conflict</td>
<td>( P \land f(s_1, \ldots, s_n) =? g(t_1, \ldots, t_p) ) [\mapsto] ( \bot ) if ( f \neq g )</td>
<td>( P \land x =? y ) [\mapsto] ( {x \mapsto y}(P) \land x =? y ) if ( x, y \in \text{Var}(P) ) and ( x \neq y )</td>
</tr>
<tr>
<td>Coalesce</td>
<td>( P \land x =? y ) [\mapsto] ( {x \mapsto y}(P) \land x =? y ) if ( x, y \in \text{Var}(P) ) and ( x \neq y )</td>
<td>( P \land x =? y ) [\mapsto] ( {x \mapsto y}(P) \land x =? y ) if ( x, y \in \text{Var}(P) ) and ( x \neq y )</td>
</tr>
<tr>
<td>Check*</td>
<td>( P \land x_1 =? s_1[x_2] \ldots \land x_n =? s_n[x_1] ) [\mapsto] ( \bot ) if ( s_i \notin \text{Var} ) for some ( i \in [1..n] )</td>
<td>( P \land x =? s ) [\mapsto] ( \bot ) if ( x \in \text{Var}(s) ) and ( s \notin \text{Var} )</td>
</tr>
<tr>
<td>Merge</td>
<td>( P \land x =? s \land x =? t ) [\mapsto] ( P \land x =? s \land s =? t ) if ( 0 &lt;</td>
<td>s</td>
</tr>
<tr>
<td>Check</td>
<td>( P \land x =? s ) [\mapsto] ( \bot ) if ( x \in \text{Var}(s) ) and ( s \notin \text{Var} )</td>
<td>( P \land x =? s ) [\mapsto] ( \bot ) if ( x \in \text{Var}(s) ) and ( s \notin \text{Var} )</td>
</tr>
<tr>
<td>Eliminate</td>
<td>( P \land x =? s ) [\mapsto] ( {x \mapsto s}(P) \land x =? s ) if ( x \notin \text{Var}(s), s \notin \text{Var}, x \in \text{Var}(P) )</td>
<td>( P \land x =? s ) [\mapsto] ( {x \mapsto s}(P) \land x =? s ) if ( x \notin \text{Var}(s), s \notin \text{Var}, x \in \text{Var}(P) )</td>
</tr>
</tbody>
</table>
Examples

\[
\begin{align*}
x &= a \\
x &= a \land y &= f(x, a) \\
f(x, f(x, a)) &= f(f(a, b), f(u, v)) \\
x &= a \land x &= b
\end{align*}
\]
A *tree solved form* for $P$ is any conjunction of equations

$$x_1 = ? \ t_1 \land \cdots \land x_n = ? \ t_n$$

equivalent to $P$ such that $\forall i, x_i$ is a variable and

(i) $\forall 1 \leq i \leq n, x_i \in \text{Var}(P)$,
(ii) $\forall 1 \leq i, j \leq n, i \neq j \Rightarrow x_i \neq x_j$,
(iii) $\forall 1 \leq i, j \leq n, x_i \notin \text{Var}(t_j)$.

Example: $x = ? f(f(y)) \land z = ? g(a)$
Computation of mgu

**Theorem**

_Starting with a unification problem P and using the above rules repeatedly until none is applicable_

— results in ⊥ iff P has no solution, or else it

— results in a tree solved form \( x_1 = ? t_1 \land \cdots \land x_n = ? t_n \) 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 $SP$.

- An inference in $SP$ 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 $SP$ 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 $S_i \Rightarrow_{SP} S_{i+1}$ via the application of expansion rules or contraction rules in $SP$.

- 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 $S_0, S_1, \ldots, S_i, \ldots$ with limit $S_\infty$ is *fair* if every inference in $SP$ with premises in $S_\infty$ is redundant with respect to some $S_j$. 
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 $S_\infty$ 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

\[ \text{Ax}(\mathcal{L}) := \{ \text{car}(\text{cons}(X, Y)) = X, \text{cdr}(\text{cons}(X, Y)) = Y \} \]

- Recall that a literal in \( S \) has one of the four possible forms:
  1. \( \text{car}(c) = d \),
  2. \( \text{cdr}(c) = d \),
  3. \( \text{cons}(c_1, c_2) = d \),
  4. \( c \neq d \).

- There are three cases to consider:
  1. inferences between two clauses in \( S \)
  2. inferences between two clauses in \( \text{Ax}(\mathcal{L}) \)
  3. inferences between a clause in \( \text{Ax}(\mathcal{L}) \) 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 \( SP \) 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 \( SP \) after flattening the input set of literals
Ax(\mathcal{A}) := \left\{ \begin{array}{l}
\text{rd}(\text{wr}(A, I, E), I) = E \\
\top \rightarrow I = J, \text{rd}(\text{wr}(A, I, E), J) = \text{rd}(A, J)
\end{array} \right\}

Apply the methodology previously described using a superposition calculus handling arbitrary clauses
### SP (Expansion Rules)

<table>
<thead>
<tr>
<th>Name</th>
<th>Rule</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sup.</strong></td>
<td>[ \Gamma \rightarrow \Delta, l[u'] = r \quad \Pi \rightarrow \Sigma, u = v ] [ \Gamma, \Pi \rightarrow \Delta, \Sigma, l[v] = r ]</td>
<td>*, **</td>
</tr>
<tr>
<td><strong>Par.</strong></td>
<td>[ \Gamma, l[u'] = r \rightarrow \Delta \quad \Pi \rightarrow \Sigma, u = v ] [ l[v] = r, \Gamma, \Pi \rightarrow \Delta, \Sigma ]</td>
<td>*, **</td>
</tr>
<tr>
<td><strong>Ref.</strong></td>
<td>[ \Gamma, u' = u \rightarrow \Delta ]</td>
<td>( (u' = u) \not&lt; (\Gamma \cup \Delta), ** )</td>
</tr>
<tr>
<td><strong>Fac.</strong></td>
<td>[ \Gamma \rightarrow \Delta, u = v, u' = v' ] [ \Gamma, v = v' \rightarrow \Delta, u = v' ]</td>
<td>( u \not&lt; v, u \not&lt; \Gamma, (u = v) \not&lt; {u' = v'} \cup \Delta, ** )</td>
</tr>
</tbody>
</table>

* \( u' \) is a not a variable, \( u \not< v, l[u'] \not< r, (u = v) \not< (\Pi \cup \Sigma) \), \( (l[u'] = r) \not< (\Gamma \cup \Delta) \)

** \( \sigma = \text{mgu}(u, u') \) implicitly applied to consequents and conditions
### SP (Contraction Rules)

<table>
<thead>
<tr>
<th>Name</th>
<th>Rule</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subsumption</td>
<td>$S \cup {C, C'}$</td>
<td>for some $\theta$, $\theta(C) \subseteq C'$, and there is no $\rho$ s.t. $\rho(C') = C$</td>
</tr>
<tr>
<td>Simplification</td>
<td>$S \cup {C[\theta(l)], l = r}$</td>
<td>$\theta(l) \succ \theta(r)$, $C[\theta(l)] \succ (\theta(l) = \theta(r))$</td>
</tr>
<tr>
<td>Deletion</td>
<td>$S \cup {\Gamma \rightarrow \Delta, t = t}$</td>
<td>$S$</td>
</tr>
</tbody>
</table>
Lemma

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

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

iv) clauses of type $t \otimes t' \lor c_1 = c'_1 \lor \cdots \lor c_n = c'_n$  
with $t \otimes t' \in \{ c \neq c', \text{rd}(c, i) = c', \text{rd}(c, i) = \text{rd}(c', i') \}$

v) clauses of type $\text{rd}(c, x) = \text{rd}(c', x) \lor c_1 = k_1 \lor \cdots \lor c_n = k_n,$  
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 $S \cup \text{Ax}(\mathcal{A})$ 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 $SP$ with a satisfiability procedure for the theory of arithmetic
References on a rewriting approach to sat proc

- Armando, Bonacina, Ranise, Schulz. “On a rewriting approach to satisfiability procedures: extension, combination of theories, and an experimental appraisal,” presented at FroCos’05, Vienna. (Experimental evaluation.)