Building Decision Procedures for Data Structures

Silvio Ranise and Christophe Ringeissen

LORIA

Lecture 4
Outline

1. Use of Superposition
   - Equality
   - Extensions of Equality

2. When the superposition-based approach works...

3. References
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 $= \equiv$ written infix satisfying the following axioms:

  $\forall x.(x = x)$ \hspace{1cm} reflexivity

  $\forall x, y.(x = y \Rightarrow y = x)$ \hspace{1cm} symmetry

  $\forall x, y, z.(x = y \land y = z \Rightarrow x = z)$ \hspace{1cm} transitivity

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

<table>
<thead>
<tr>
<th>Superpos$_{1}$</th>
<th>[ \begin{align*} c &amp;= c' \ c &amp;= d \end{align*} ]</th>
<th>[ c' = d ] if $c \succ c'$, $c \succ d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Superpos$_{2}$</td>
<td>[ \begin{align*} c_j &amp;= c'<em>j \ f(c_1, \ldots, c_j, \ldots, c_n) &amp;= c</em>{n+1} \end{align*} ]</td>
<td>[ f(c_1, \ldots, c'<em>j, \ldots, c_n) = c</em>{n+1} ] if $c_j \succ c'_j$</td>
</tr>
<tr>
<td>Superpos$_{3}$</td>
<td>[ \begin{align*} f(c_1, \ldots, c_n) &amp;= c'<em>{n+1} \ f(c_1, \ldots, c_n) &amp;= c</em>{n+1} \end{align*} ]</td>
<td>[ c_{n+1} = c'<em>{n+1} ] if $c</em>{n+1} \succ c'_{n+1}$</td>
</tr>
<tr>
<td>Paramodul</td>
<td>[ \begin{align*} c &amp;= c' \ c &amp;\neq d \end{align*} ]</td>
<td>[ c' \neq d ] if $c \succ c'$, $c \succ d$</td>
</tr>
</tbody>
</table>
| Eq. Res.       | \[ c \neq c \] | \[ \bot \] 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 ⊥ is derived, then unsatisfiability is reported
  - if ⊥ 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 arrays:

\[
\begin{align*}
\text{read}(\text{write}(A, I, E), I) &= E \\
I &= J \lor \text{read}(\text{write}(A, I, E), J) &= \text{read}(A, J)
\end{align*}
\]

where \(A, I, J, E\) 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 $\mathcal{SP}$

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 Calculus (in the case of unit clauses)

Superposition
\[
\frac{l[u'] = r \quad u = t}{(l[t] = r)\sigma} \quad (i), (ii), (iii), (iv)
\]

Paramodulation
\[
\frac{l[u'] \neq r \quad u = t}{(l[t] \neq r)\sigma} \quad (i), (ii), (iii), (iv)
\]

Reflection
\[
\frac{u' \neq u}{\Box} \quad (i)
\]

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 Inference Rules.

Replacement of equal by equal performed up to unification.
Rules controlled by a simplification ordering on terms.
### INFEERENCE RULES OF $SP(I)$

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

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

** $\sigma = mgu(u, u')$ implicitly applied to consequents and conditions

---

Silvio Ranise and Christophe Ringeissen | Building Decision Procedures for Data Structures
### CONTRACTION RULES OF $SP$ (II)

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

- Requirement: $f(c_1, \ldots, c_n) \succ c_0$
  for each non-constant symbol $f$ and constant $c_i$ ($i = 0, 1, \ldots, n$)
- [Definition:] $(a = b) \succ (c = d)$ iff $\{a, b\} \succeq \{c, d\}$
  (where $\succeq$ 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
Definition (Well-founded Ordering)

> is *well-founded* if there is no infinite decreasing chain

\[ t_1 > t_2 > \ldots \]

Definition (Multiset Extension)

\[ M \succ^\text{mult} N \text{ if } M \neq N \text{ and } \\
N(t) > M(t) \Rightarrow \exists t' : t' > t \text{ and } M(t') > N(t') \]

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

Example (Multiset set extension of the ordering on Naturals)

\[ \{3, 3, 3, 2, 1\} \succ^\text{mult} \{3, 3, 2, 2, 2, 1\} \]

\[ \{3, 3, 1, 2\} \succ^\text{mult} \{1, 1, 2\} \]
### 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 σ(s) > σ(t),

### Example (Recursive Path Ordering)

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

1. $f = g$ and $\{s_1, \ldots, s_n\} >_{rpo}^{mult} \{t_1, \ldots, t_m\}$
2. $f \not> \overline{f} g$ and $\forall j \in \{1, \ldots, m\}$ $s >_{rpo} t_j$
3. $\exists i \in \{1, \ldots, n\}$ such that either $s_i >_{rpo} t$ or $s_i \sim t$ where $\sim$ means equivalent up to permutation of subterms
### Reduction Ordering: Another 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 \( (s_1, \ldots, s_n) >_{lpo} (t_1, \ldots, t_m) \) 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 of a well-founded ordering is well-founded
- LPO and RPO are simplification orderings: for any term \( s \) and any context \( u, u[s] > s \)
- LPO and RPO are 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 RPO or LPO?
which precedence to choose?
Rules for syntactic unification (computation of mgu)

Delete: \[ P \land s =? s \]
Conflict: \[ P \land f(s_1, \ldots, s_n) =? g(t_1, \ldots, t_p) \]
Decompose: \[ P \land f(s_1, \ldots, s_n) =? f(t_1, \ldots, t_n) \]

Coalesce: \[ P \land x =? y \]
Check*: \[ P \land x_1 =? s_1[x_2] \ldots \land x_n =? s_n[x_1] \]

Merge: \[ P \land x =? s \land x =? t \]
Check: \[ P \land x =? s \]
Eliminate: \[ P \land x =? s \]
Examples

\[
x = ? a
\]
\[
x = ? a \land y = ? f(x, a)
\]
\[
f(x, f(x, a)) = ? f(f(a, b), f(u, v))
\]
\[
x = ? a \land x = ? b
\]
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 $\forall i, x_i \in \mathcal{X}$ and:

(i) $\forall 1 \leq i \leq n, x_i \in \mathcal{V}ar(P)$,

(ii) $\forall 1 \leq i, j \leq n, i \neq j \Rightarrow x_i \neq x_j$,

(iii) $\forall 1 \leq i, j \leq n, x_i \notin \mathcal{V}ar(t_j)$.

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 \( \mathbf{F} \) 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 \( \text{mgu}(P) \).
Redundancy and Saturation

Definition

- A clause \( C \) is *redundant* with respect to a set \( S \) of clauses if either \( C \in S \) or \( S \) can be obtained from \( S \cup \{C\} \) by a sequence of application of contraction rules.
- An inference 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* with respect to \( SP \) if every inference of \( SP \) 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* with respect to \( SP \) if for every inference in \( SP \) with premises in \( S_\infty \), there is some \( j \geq 0 \) such that the inference is redundant with respect to \( S_j \).
Refutation Completeness

**Lemma (Nieuwenhuis-Rubio)**

Let $S_0, S_1, \ldots, S_n, \ldots$ be a derivation and let $C$ be a clause in $(\bigcup_i S_i) \setminus S_{\infty}$. Then $C$ is redundant with respect to $S_{\infty}$.

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 $SP$, then (i) its limit $S_{\infty}$ is saturated with respect to $SP$, (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

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

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

- 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 $Ax(L)$
  This is not very interesting since there are no possible inferences between the two axioms in $Ax(L)$
- Case 3: inferences between a clause in $Ax(L)$ and a clause in $S$
  - a superposition between $\text{car(cons}(X, Y)) = X$ and $\text{cons}(c_1, c_2) = d$ yielding $\text{car}(d) = c_1$ and
  - a superposition between $\text{cdr(cons}(X, Y)) = Y$ and $\text{cons}(c_1, c_2) = d$ yielding $\text{cdr}(d) = c_2$
• We are almost done, it is sufficient to notice that
  ▶ only finitely many equalities of the form (a) and (b) 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
Extensions of the theory of Lists

- Recall that in the proof of termination of $SP$ on $Ax(L) \cup S$, we have observed that inferences between clauses in $S$ were already considered for the ground case.
- So, if we consider a signature $\Sigma := \{\text{cons, car, cdr}\} \cup \Sigma_{UF}$, where $\Sigma_{UF}$ is a finite set of function symbols, the proof of termination above continues to hold.
- In other words, we are capable of solving the satisfiability problem for $L \cup UF \cup S$, where $S$ is a set of ground literals built out of the interpreted function symbols cons, car, cdr and arbitrary uninterpreted function symbols.
- The above holds for all satisfiability procedure built by the rewriting approach described here.
SP for lists: summary

- Analysis of the possible inferences in $\mathcal{SP}$

**Lemma**

Let $S$ be a finite set of flat $\Sigma_L$-literals. The clauses occurring in the saturations of $S \cup \text{Ax}(\mathcal{L})$ by $\mathcal{SP}$ can only be the empty clause, ground flat literals, or the equalities in $\text{Ax}(\mathcal{L})$.

- Termination follows

**Lemma**

Let $S$ be a finite set of flat $\Sigma_L$-literals. All the saturations of $S \cup \text{Ax}(\mathcal{L})$ by $\mathcal{SP}$ are finite.

- From termination, fairness, and refutation completeness...

**Theorem**

$\mathcal{SP}$ is a decision procedure for $\mathcal{L}$.
Deriving a Decision Procedure for Arrays (I)

**Sorts:** to avoid problematic terms such as

\[ \text{rd}(a, \underbrace{\text{wr}(a, i, e)}_{\text{not an index!}}) \]

**Presentation of \( \mathcal{A}_e^s \):**

\[
\begin{align*}
\text{Ax}(\mathcal{A}_e^s) & := \text{Ax}(\mathcal{A}_e^s) \bigcup \forall A, B. (\forall I. (\text{rd}(A, I) = \text{rd}(B, I)) \implies A = B) \\
& \quad \text{extensionality}
\end{align*}
\]

\[
\text{Ax}(\mathcal{A}_e^s) := \begin{cases} 
\text{rd(\text{wr}(A, I, E), I)} = E \\
I = J \lor \text{rd(\text{wr}(A, I, E), J)} = \text{rd}(A, J)
\end{cases}
\]
Problem: how to handle the axiom of extensionality?

Lemma

Let $S$ be a set of ground literals and let $S'$ be obtained from $S$ by replacing all the inequalities of the form $t \neq t'$ with $\exists i. \text{rd}(t, i) \neq \text{rd}(t', i)$, where $t$ and $t'$ are terms of sort ARRAY. Then $S$ is $\mathcal{A}_e^s$-satisfiable iff $S'$ is $\mathcal{A}^s$-satisfiable.

Then... simply Skolemize, i.e. $\exists i. \text{rd}(t, i) \neq \text{rd}(t', i)$ becomes

$$\text{rd}(t, i_{sk(t,t')}) \neq \text{rd}(t', i_{sk(t,t')})$$

Finally apply the two step methodology previously described
Deriving a Decision Procedure for Arrays (III)

**Lemma**

Let $S$ be a finite set of flat literals. The clauses occurring in the saturations of $S \cup \text{Ax}(A)$ 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 $S \cup \text{Ax}(A)$ are finite.
A rewriting approach: **lists**

- **equality:** $\Sigma_{UF} := \text{finite set of function symbols, } Ax(UF) := \emptyset$
- **lists à la Shostak:** $\Sigma_{L_{Sh}} := \{\text{cons, car, cdr}\} \cup \Sigma_{UF}$,
  $$Ax(L_{Sh}) := \{\text{car}(\text{cons}(X, Y)) = X, \text{cdr}(\text{cons}(X, Y)) = Y, \text{cons}(\text{car}(X), \text{cdr}(X)) = X\}$$
- **lists à la Nelson-Oppen:**
  $\Sigma_{L_{NO}} := \{\text{cons, car, cdr, atom}\} \cup \Sigma_{UF}$,
  $$Ax(L_{NO}) := \{\text{car}(\text{cons}(X, Y)) = X, \text{cdr}(\text{cons}(X, Y)) = Y, \neg\text{atom}(\text{cons}(X, Y)), \text{atom}(X) \lor \text{cons}(\text{car}(X), \text{cdr}(X)) = X\}$$
A rewriting approach: arrays

arrays w/ extensionality: $\Sigma_{A^s} := \{\text{rd, wr}\} \cup \Sigma_{UF}$,

$$Ax(A^s) := \left\{ \begin{array}{l}
\text{rd}(\text{wr}(A, l, E), l) = E \\
I = J \lor \text{rd}(\text{wr}(A, l, E), J) = \text{rd}(A, J)
\end{array} \right\}$$

$$Ax(A^s_\varnothing) := Ax(A^s) \cup$$
$$\{\forall A, B. (\forall l. (\text{rd}(A, l) = \text{rd}(B, l)) \implies A = B)\}$$
A rewriting approach: \textbf{records}

- records w/ extensionality:
  \[
  \Sigma_{\mathcal{R}^s} := \{ \text{rsel}_i, \text{rst}_i | i = 1, \ldots, n \} \cup \Sigma_{\text{UF}},
  \]

  \[
  \text{Ax}(\mathcal{R}^s) := \left\{ \begin{array}{l}
  \text{rsel}_i(\text{rst}_i(X, V)) = V \quad \text{for all } i, 1 \leq i \leq n \\
  \text{rsel}_j(\text{rst}_i(X, V)) = \text{rsel}_j(X) \quad \text{for all } i, j, 1 \leq i \neq j \leq n
  \end{array} \right. 
  \]

  \[
  \text{Ax}(\mathcal{R}_\phi^s) := \text{Ax}(\mathcal{R}^s) \cup \{ \forall X, Y. (\bigwedge_{i=1}^{n} \text{rsel}_i(X) = \text{rsel}_i(Y) \implies X = Y) \}
  \]
A rewriting approach: **integer offsets**

- **integer offsets**: \( \Sigma_I := \{\text{succ}, \text{prec}\} \cup \Sigma_{UF} \),

  \[
  Ax(I) := \begin{cases} 
  \text{succ}(\text{prec}(X)) = X, \text{prec}(\text{succ}(X)) = X, \\
  \text{succ}^i(X) \neq X \\
  \text{acyclicity}
  \end{cases} 
  \]

  where \( \text{succ}^1(x) = \text{succ}(x), \ \text{succ}^{i+1}(x) = \text{succ}(\text{succ}^i(x)) \)
  for \( i \geq 1 \)

- **integer offsets modulo**: \( \Sigma_{I_k} := \{\text{succ}, \text{prec}\} \cup \Sigma_{UF} \),

  \[
  Ax(I_k) := \begin{cases} 
  \text{succ}(\text{prec}(X)) = X, \text{prec}(\text{succ}(X)) = X, \\
  \text{succ}^i(X) \neq X \\
  \text{k-acyclicity} \\
  \text{succ}^k(X) = X
  \end{cases} 
  \]

for \( 1 \leq i \leq k - 1 \)
Problems with the rewriting approach

- Unfortunately not all theories are equational...
  - for example, the usual theory of arithmetic is not equational, i.e. there does not exist a finite set of equalities which axiomatize the theory
- Because of this and the ubiquity of arithmetic in practically any verification problem *jointly* with equational theories, we need to combine satisfiability procedures
  - for such theories, the approach advocated above to build decision procedures, obviously cannot be used
  - fortunately, decision procedures for full arithmetic over rationals (or reals), over integers, for some of its fragments are widely studied
Because of all these reasons, it is important to combine satisfiability procedures to obtain a satisfiability procedure for the union of some theories.
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.)