Decision procedures for equality

Silvio Ranise & Christophe Ringeissen

INRIA-Lorraine, Nancy (France)  U. degli Studi di Milano, Milano (Italia)

Decision Procedures for Software Verification: Lecture 1
December 5, 2013
GOAL: design decision procedures for the satisfiability problem of arbitrary Boolean combinations of ground atoms whose only main symbol is equality

Two techniques

1. By translation to the Boolean satisfiability problem (via Herbrand-method)
2. By reduction (i.e. using oriented equalities)
1 A motivating example

2 The satisfiability problem for equational formulae (SATEQ)
   - A fundamental tool in automated reasoning: Herbrand theorem
   - Decidability of SATEQ by bounding Herbrand universe

3 A decision procedure for a class of equational formulae
   - Equality as a graph
   - Convexity: its role in designing a dec proc for equality

4 A better decision procedure based on reduction
   - Reduction relations: formal preliminaries
   - Convergent reduction relations as dec. proc’s for equality
What is an (optimizing) compiler?

**Definition (Compilers)**
Special programs that take instructions written in a high level language (e.g., C, Pascal) and convert it into machine language or code the computer can understand.

**Example**
Consider the following simple program fragment in C:

```c
int x, y, z;
s0: ...
    /* y and z are initialized */
s1: x = (y+z) * (y+z) * (z+y) * (z+y);
...
```

**Problem**: sub-expressions are needlessly re-computed!
An (optimizing) compiler: an example

Example (cont’d)

By exploiting only the syntactic structure of sub-expressions, transform

```
int x, y, z;
int aux1, aux2;
s0: ... /* y and z are initialized */
s1: x = (y+z) * (y+z) * (z+y) * (z+y);
t0: ... /* y and z are initialized */
t1: aux1 = (y+z);
t2: aux2 = (z+y);
t3: x = aux1 * aux1 * aux2 * aux2;
```

into

```
int x, y, z;
s0: ... /* y and z are initialized */
s1: x = (y+z) * (y+z) * (z+y) * (z+y);
```

which avoids the re-computation of sub-expressions!
Example (cont’d)

**QUESTION:** how can we guarantee that the value stored in $x$ after the computation of the transformed program is equal to that in $x$ after the computation of the source?

**ANSWER:** ignore the arithmetic properties of all arithmetic operations and consider them as uninterpreted functions (i.e. $+ \mapsto f$ and $\ast \mapsto g$). Then, prove the validity of the following proof obligation:

\[
\begin{align*}
&y_{s0} = y_{t0} \land z_{s0} = z_{t0} \\
x_{s1} = g(g(f(y_{s0}, z_{s0}), f(y_{s0}, z_{s0}))), g(f(z_{s0}, y_{s0}), f(z_{s0}, y_{s0}))) \land \\
aux1_{t1} = f(y_{t0}, z_{t0}) \land \\
aux2_{t2} = f(z_{t0}, y_{t0}) \land \\
x_{t3} = g(g(aux1_{t1}, aux1_{t1}), g(aux2_{t2}, aux2_{t2})) \land \\& \Rightarrow x_{s1} = x_{t3}
\end{align*}
\]
The satisfiability problem for equational formulae (SATEQ)

Definition (SATEQ)

Let $\Sigma$ be a set of function and constant symbols. Let $A_\Sigma$ be the set of **equational atoms** of the form $t_1 = t_2$, where $t_i$ is a term built out of the symbols in $\Sigma$. An **equational formula** is a Boolean combination of equational atoms in $A_\Sigma$.

The satisfiability problem for equational formulae is the problem of checking the satisfiability of equational formulae.

**QUESTION:** is this problem decidable? I.e. does it exist a decision procedure for such a problem? I.e. does it exist an algorithm which takes an arbitrary equational formula and returns *satisfiable* when there exists a model of it and *unsatisfiable* when there is not structure satisfying the formula?
SATEQ: example

For our example, we should prove the unsatisfiability of (Why ?)

\[
\begin{align*}
y_{s0} &= y_{t0} \land z_{s0} = z_{t0} \\
x_{s1} &= g(g(f(y_{s0}, z_{s0}), f(y_{s0}, z_{s0})), g(f(z_{s0}, y_{s0}), f(z_{s0}, y_{s0}))) \\
aux1_{t1} &= f(y_{t0}, z_{t0}) \\
aux2_{t2} &= f(z_{t0}, y_{t0}) \\
x_{t3} &= g(g(aux1_{t1}, aux1_{t1}), g(aux2_{t2}, aux2_{t2}))
\end{align*}
\]  \land x_{s1} \neq x_{t3}

which is indeed an equational formula whose atoms are built out of the symbols in \( \Sigma := \{ f/2, g/2, x_{s0}/0, y_{s0}/0, x_{t0}/0, y_{t0}/0, \ldots \} \)
Validity and satisfiability

- $\Phi \models \varphi$ iff every model of $\Phi$ is also a model of $\varphi$
- This problem is quite difficult since we must consider all possible models (which are infinitely many) of $\Phi$ and then check that $\varphi$ is true in all such models...
- So, we prefer to reason by refutation:

  $$\Phi \models \varphi \iff \Phi \cup \{\neg \varphi\} \text{ is unsatisfiable}$$

- Recall that $\psi$ is **satisfiable** (or **consistent**) iff there exists a model of $\psi$
- Furthermore, as we will see in a moment, we know that it is sufficient to search for a model of $\psi$ in quite a particular class of models (**Herbrand models**)
Given a formula $\psi$, consider:

- the whole class of structures of $\psi$
- the class of **Herbrand structures** of $\psi$

Given a structure $\mathcal{M}$ of $\psi$ which is not Herbrand, we can always find a Herbrand structure $\mathcal{H}$ such that $\mathcal{M} \models \psi$ iff $\mathcal{H} \models \psi$
Herbrand universe: UH

**Assume** the following form $\forall x_1, \ldots, x_k. \psi$

where $\psi$ is a Boolean combination of atoms without quantifiers

- $UH_0 := \text{constants occurring in } \psi$
  - if there are no constants in $\psi$, then $UH_0 := \{a\}$ (for $a$ an arbitrary constant symbol)

- $UH_{i+1} := UH_i \cup \{f(t_1, \ldots, t_n) | f \text{ is in } \psi \text{ of arity } n \text{ and } t_1, \ldots, t_n \in UH_i\}$

- The **Herbrand universe** is defined as follows:

\[
UH := \bigcup_{i=0}^{\infty} UH_i
\]
Herbrand structures

Definition

The Herbrand structure \( \mathcal{H} = \langle D_\mathcal{H}, I_\mathcal{H} \rangle \) of \( \forall x_1, \ldots, x_k. \psi \) (where \( \psi \) is a Boolean combination of atoms without quantifiers) is such that

- \( D_\mathcal{H} \) is the Herbrand universe of \( \psi \)
- \( I_\mathcal{H} \) is defined on (ground) terms as follows:

\[
I_\mathcal{H}(c) := \begin{cases} 
\text{c if c is a constant in } \psi 
\end{cases} \\
I_\mathcal{H}(f(t_1, \ldots, t_n)) := \text{mapping the n-tuple of terms } (t_1, \ldots, t_n) \text{ to the term } f(t_1, \ldots, t_n)
\]
Herbrand theorem

Theorem

The formula $\forall x_1, \ldots, x_k. \psi$ is consistent iff it admits a Herbrand model, where $\psi$ is a quantifier-free Boolean combination of atoms.

Proof.

($\Leftarrow$): obvious.

($\Rightarrow$): Let $\mathcal{M}$ be a model of $\phi = (\forall x_1, \ldots, x_k. \psi)$. We can define an interpretation over atoms $p(t_1, \ldots, t_n)$ where $t_1, \ldots, t_n \in D_H$: $p(t_1, \ldots, t_n)$ is true in $\mathcal{H}$ if and only if $p(t_1, \ldots, t_n)$ is true in $\mathcal{M}$.

Then, by structural induction on formulas, we can show that

$$\mathcal{H} \models \phi \text{ if and only if } \mathcal{M} \models \phi$$
Herbrand method (to refute formulae)

- **Input**: $\forall x_1, ..., x_k. \psi$ where $\psi$ is a quantifier-free Boolean combination of atoms
- **Output**: satisfiable/unsatisfiable
- **Method**:
  - Consider the Herbrand universe of $\psi$
  - Enumerate the ground instances of $\psi$ by substituting the terms of the Herbrand universe to the variables $x_1, ..., x_k$
  - Check the Boolean satisfiability of the ground instances of $\psi$
  - If a ground instance of $\psi$ is unsatisfiable, then $\psi$ is unsatisfiable (and the method terminates)
  - Otherwise, keep enumerating the ground instances of $\psi$
Herbrand method: remarks

- In general, Herbrand method is a semi-decision procedure for unsatisfiability in the sense that it terminates whenever the input formula is unsatisfiable...
  This is so because of

Theorem (*Compactness*)

A set $\Gamma$ of Boolean formulae is satisfiable iff every finite set $\Delta \subseteq \Gamma$ is satisfiable.
Herbrand method: remarks

- In particular, Herbrand method terminates, regardless of the satisfiability or unsatisfiability of the input formula, when the **Herbrand universe is finite**...
  - ... since only finitely many ground instances must be considered
  - ... the Herbrand universe is finite whenever there are no function symbols in the input formula (only constants)
- Herbrand method **does not terminate** if the input formula is satisfiable and the Herbrand universe is infinite...
  - ... for this, it is sufficient to have one function symbols of arity $\geq 1$
- We assume to be able to check the (un-)satisfiability of Boolean formulae...
Checking Boolean (un-)satisfiability: how?

- Truth tables... *not very efficient!*
- SAT is computationally very demanding: $\mathcal{NP}$-problem
- For Horn formulae: linear-time (in the number of Boolean variables) algorithm exists
- In practice: Davis-Putnam-Logemann-Loveland (DPLL) algorithm

A **Horn formula** is a formula of the form:

$$(a_1 \land \cdots \land a_n) \Rightarrow a_{n+1}$$

for $a_i$ an atom, $i = 1, \ldots, n, n+1$
DPLL: abstract description

- Formulae are translated in clausal normal form, i.e. conjunction of clauses (i.e. disjunction of literals)
- It is always possible to do this but exponential blow-up if done naively... linear in time and space but formulae are only equisatisfiable
- In the following, conjunctions of clauses are considered as a set of sets of clauses:
  - associativity and commutativity of $\lor$ and $\land$ are built-in the data structure
  - also some simplification is built-in, e.g. $p \lor p$ is represented by the singleton set $\{p\}$
DPLL: abstract description

- Formulae are translated in **clausal normal form**, i.e. conjunction of clauses (i.e. disjunction of literals)
- It is always possible to do this but exponential blow-up if done naively... linear in time and space but formulae are only equisatisfiable
- In the following, conjunctions of clauses are considered as a set of sets of clauses:
  - associativity and commutativity of $\lor$ and $\land$ are built-in the data structure
  - also some simplification is built-in, e.g. $p \lor p$ is represented by the singleton set $\{p\}$
DPLL: abstract description

Let $S$ be a set of clauses

**Unit Resolution**

\[
\frac{S \cup \{L, C \lor \bar{L}\}}{S \cup \{L, C\}} \quad \text{if} \quad \overline{A} := A
\]

\[
\frac{A := \neg A}{A := \neg A}
\]

**Unit Subsumption**

\[
\frac{S \cup \{L, C \lor L\}}{S \cup \{L\}}
\]

**Splitting**

\[
\frac{S}{S \cup \{A\} \mid S \cup \{\neg A\}} \quad \text{if} \quad A \text{ is an atom occurring in } S
\]

There exists very efficient implementation of this calculus: zChaff, MiniSAT, Berkmin, ...
Recall that

- in first-order logic: the symbol of equality $=\,$, is **uninterpreted** (it is an arbitrary binary predicate symbol, written infix)
- in first-order logic with equality: the symbol of equality $=\,$, is **interpreted** to be the identity relation on the domain of the structure

Herbrand theorem is stated and proved in first-order logic (without equality)

**QUESTION**: can we use Herbrand method to check the satisfiability of equational formulae? So to have at least a semi-decision procedure...

**ANSWER**: yes with a little bit of effort...
Satisfiability with and without equality

- Let $\varphi$ be an equational formula built out of the symbols in $\Sigma$
- Consider the following set $EQ_\Sigma$ of axioms saying that $=$ is a congruence relation:

$$
\forall x. (x = x) \\
\forall x, y. (x = y \Rightarrow y = x) \\
\forall x, y, z. (x = y \land y = z \Rightarrow x = z) \\
\forall \ldots x, y \ldots (x = y \Rightarrow f(\ldots x \ldots) = f(\ldots y \ldots)) \quad \text{for each } f \in \Sigma
$$

**Remark:** $\varphi$ is satisfiable in first-order logic with equality iff $\varphi \land EQ_\Sigma$ is satisfiable in first-order logic without equality
The satisfiability problem for equational formulae (SATEQ) is decidable by bounding the Herbrand universe. The theorem allows us to use the Herbrand method to solve arbitrary instances of SATEQ.

Given an equational formula $\varphi$:

1. Compute the set $\Sigma$ of function and constant symbols in $\varphi$.
2. Compute the set $EQ_\Sigma$.
3. Return the result of applying the Herbrand method on $\varphi \land EQ_\Sigma$ (where $=$ is considered as an arbitrary predicate symbol).

About termination: it is sufficient that $\Sigma$ contains one non-constant symbol that the Herbrand universe of $\varphi \land EQ_\Sigma$ is infinite and the procedure is not guaranteed to terminate.
Remarks on the semi-decision procedure for SATEQ

**BIG QUESTION**: can we turn the semi-decision procedure based on Herbrand method into a decision procedure (i.e. an algorithm which solves SATEQ)?

**ANSWER**: yes, by showing that it is always possible to find a finite subset of the Herbrand universe which is sufficient to detect unsatisfiability!
Example

- Consider the following instance of SATEQ: is

\[ \varphi \equiv f(f(f(a))) = a \land f(f(f(f(f(a))))) = a \land f(a) \neq a \]

unsatisfiable?

- By substituting equal by equal, we can derive a contradiction:

\[
\begin{align*}
    f(f(f(a))) &= a \land f(f(f(f(a)))) = a \land f(a) \neq a \\
    f(f(f(a))) &= a \land f(f(a)) = a \land f(a) \neq a \\
    f(f(f(a))) &= a \land f(f(a)) = a \land f(a) \neq a \\
    f(a) &= a \land f(f(a)) = a \land f(a) \neq a
\end{align*}
\]

Contradiction!

- **Key observation:** in deriving the contradiction, we have only used terms and sub-terms which occur in the input formula \( \varphi \)!
Decidability of SATEQ: theorem

Theorem

\[ \varphi \land EQ_\Sigma \text{ is unsatisfiable iff } \varphi \land GEQ_\Sigma^{\varphi} \text{ is unsatisfiable,} \]

where \( GEQ_\Sigma^{\varphi} \) is the (finite) set of ground instances of \( EQ_\Sigma \) obtained by instantiating variables with all terms and sub-terms occurring in \( \varphi \).

Corollary

Given an equational formula \( \varphi \). The following algorithm

1. compute the set \( \Sigma \) of function and constant symbols in \( \varphi \)
2. compute the set \( GEQ_\Sigma^{\varphi} \)
3. return the result of checking the (Boolean) satisfiability of \( \varphi \land GEQ_\Sigma^{\varphi} \)

terminates and returns whether \( \varphi \) is satisfiable or not.

Hence, SATEQ is decidable.
Idea of the proof of theorem

\[ \varphi \land EQ_\Sigma \text{ is unsat.} \Rightarrow \varphi \land GEQ_{\varphi}^\Sigma \text{ is unsat.} \]

consider the counter-positive...
Idea of the proof of theorem

\[ \varphi \land GEQ_\sum^\varphi \text{ is sat.} \Rightarrow \varphi \land EQ_\sum \text{ is sat.} \]
Proof of theorem

1. \( \varphi \land EQ_\Sigma \) is unsat. \( \Rightarrow \) \( \varphi \land GEQ_\varphi^\Sigma \) is unsat.

Prove the counter-positive: \( \varphi \land GEQ_\varphi^\Sigma \) is sat. \( \Rightarrow \varphi \land EQ_\Sigma \) is sat.

Assume \( \varphi \land GEQ_\varphi^\Sigma \). So, there must exist a structure \( M = (D_M, I_M) \) satisfying both \( \varphi \) and \( GEQ_\varphi^\Sigma \).

Consider a structure \( M' = (D_{M'}, I_{M'}) \) where:

- \( D_{M'} = D_M \cup \{\#\} \), where \( \# \notin D_M \)
- \( I_{M'} \) is defined as follows:

\[
I_{M'}(t) := \begin{cases} 
I_M(t) & \text{if } t \text{ occurs in } \varphi \\
\# & \text{otherwise}
\end{cases}
\]

Since for each term \( t \) occurring in \( \varphi \), we have that \( I_{M'}(t) = I_M(t) \) by construction, we derive that each equational atom \( a \) in \( \varphi \land GEQ_\varphi^\Sigma \), we have that \( M' \models a \) iff \( M \models a \). Hence, by definition of truth, we have that \( \varphi \land GEQ_\varphi^\Sigma \) is true in \( M' \)...
Proof of theorem

1. (cont’d from previous slide) \( \varphi \land GEQ^\varphi_\Sigma \) is sat. \( \Rightarrow \) \( \varphi \land EQ_\Sigma \) is sat.

Consider the set \( T \) obtained by deleting the terms and sub-terms in \( \varphi \) from its Herbrand universe and let \( GEQ^T_\Sigma \) be the set of instances of \( EQ_\Sigma \) obtained by substituting variables with terms in \( T \).

Since, \( l_{M'}(t) = \# \) for all \( t \in T \), we have that any formula in \( GEQ^T_\Sigma \) is satisfied by \( M' \) (Why?). Hence, all ground instances of \( EQ_\Sigma \) are true in \( M' \) and, by definition truth, \( EQ_\Sigma \) is valid in \( M' \).

As a consequence, \( M' \) is a model of \( EQ_\Sigma \) satisfying \( \varphi \), and we can conclude that \( \varphi \land EQ_\Sigma \) is satisfiable.

2. \( \varphi \land GEQ^\varphi_\Sigma \) is unsat. \( \Rightarrow \) \( \varphi \land EQ_\Sigma \) is unsat.

Exercise! (Hint: consider the counter-positive.)
Complexity of SATEQ and the designed decision procedure

- SATEQ is in $\mathcal{NP}$ since it subsumes SAT
- To evaluate the designed decision procedure, consider the sub-set of equational formulae built out of conjunctions of possibly negated equational atoms of the form $c = d$ (for $c, d$ being constant symbols): what about the complexity of the decision procedure for this class?
- Notice that for this class of formulae, the corresponding Boolean formulae are Horn clauses (i.e. clauses containing at most one positive literals)...
- The SAT problem for propositional Horn clauses can be solved in linear time in the number of Boolean variables...
- **QUESTION:** how many Boolean variables are in $\varphi \land GEQ_\Sigma^\varphi$?
Complexity of the designed decision procedure

- Let $n$ be the number of constants in $\varphi$
- $GEQ_{\Sigma}^{\varphi}$ will contain
  1. $n$ Boolean variables from instantiating: $\forall x.(x = x)$
  2. $n(n - 1)/2$ Boolean variables from instantiating:
     $\forall x, y.(x = y \Rightarrow y = x)$
  3. $n(n^2 - 1)/2$ Boolean variables from instantiating:
     $\forall x, y, z.(x = y \land y = z \Rightarrow x = z)$

- If $\varphi$ contains $m$ atoms, the formula $GEQ_{\Sigma}^{\varphi}$ will contain
  $m + n + n(n - 1)/2 + n(n^2 - 1)/2$ Boolean variables: when $m$ is a linear function of $n$, the decision procedure we have designed will have a **cubic complexity**

**QUESTION**: can we do better (for this particular subset of equational formulae)?
Towards a better decision procedure

Consider the sources of inefficiency in the previously designed decision procedure:

- a **quadratic** blow-up to handle symmetry of $=\$
- a **cubic** blow-up to handle transitivity of $=\$

Let us take a different perspective on equality: consider $=\$ as a binary relation which must be an equivalence (since it must be reflexive, symmetric, and transitive)

This amounts to transform the equality $c = d$ into two ordered pairs $(c, d)$ and $(d, c)$

So, we still have a quadratic blow up due to symmetry but we can do something to handle transitivity...
**Equality as a binary relation**

- **QUESTION**: how can we represent a (finite) binary relation in the store of a computer?
- **ANSWER**: as a graph!
- **IDEA**: use an efficient graph traversal algorithm to handle transitivity!
Equality as a binary relation: summary

- If we consider equality as a binary relation and represent it by means of a graph, then
  1. the number of nodes in the graph is quadratic in the number of constants in the conjunctions of equational literals
  2. checking the unsatisfiability of a conjunction of equational literals amounts to checking whether there exists a disequality $c \neq d$ such that the vertex labelled with $c$ and that labelled with $d$ are connected

**QUESTION:** what is the complexity of the best algorithm to find whether two nodes in a graph are connected?

**ANSWER:** it is linear in sum of the number of nodes and the number of edges
A better decision procedure for conjunctions of equational literals

Let $\varphi$ be a conj. of equational literals of the form $c = d$ or $\neg c = d$

1. Let $\varphi^{eq}$ be the conjunction of all equalities and $\varphi^{diseq}$ be the conjunctions of all disequalities in $\varphi$
2. Build the graph $G$ associated with $\varphi^{eq}$
3. Let $c \neq d$ be a disequality in $\varphi^{diseq}$:
   - if $c$ and $d$ are connected in $G$, then return unsatisfiable
   - otherwise, consider another disequality in $\varphi^{diseq}$
4. When all diseq. in $\varphi^{diseq}$ have been considered, return satisfiable

If the number of atoms in $\varphi$ is linear in the number of constants in $\varphi$, then the running time of the algorithm will be quadratic in the number of constants in $\varphi$...

Better than the cubic behavior of the previous procedure!
Remarks

- Notice that we have separated equalities and disequalities in the procedure because of the following reasons:
  - **Conjunctions of equalities are always satisfiable**
    
    Exercise: show why! (Hints: you need to consider a particular structure which satisfies all equalities... how can you make equal any constant?)
  
  - **Convexity of equality**: if the conjunction $\varphi^{eq} \land \varphi^{diseq}$ of equational literals is unsatisfiable, then there must exist just one disequality $c \neq d$ in $\varphi^{diseq}$ such that $\varphi^{eq} \land c \neq d$ is unsatisfiable
Can we do even better than quadratic?

Source of inefficiency: symmetry or, equivalently, bidirectionality of equality

**QUESTION**: can we orient the equality in one direction without losing refutation completeness, i.e. without returning satisfiable when it is unsatisfiable?

**Example**: check the unsatisfiability of \( c = c_1 \land c = c_2 \land c_1 \neq c_2 \)

Now, orient the two equalities from left-to-right, i.e.

\[
\begin{align*}
c & \rightarrow c_1 \\
c & \rightarrow c_2
\end{align*}
\]

and consider the reflexive and transitive closure \( \rightarrow^* \) of \( \rightarrow \).

Unfortunately: \( c_1 \not\rightarrow^* c_2 \). So, \( \rightarrow^* \subseteq = \) and \( \rightarrow^* \) is different from =

However, if we consider the symmetric, reflexive, and transitive closure \( \leftrightarrow^* \) of \( \rightarrow \), then we have \( \leftrightarrow^* \) is equal to =
Orienting equalities

- **GOAL**: orient equalities so to disregard symmetry and to greatly reduce the number of transitivity chains (or, equivalently, paths in the associated oriented graph) to be considered in such a way that we can still show the satisfiability of sets of literals over constants without losing precision (i.e. without losing refutational completeness).

- Formally, we introduce a binary relation $\rightarrow$ (to emphasize that it is an oriented version of $=$) on the constants in $\varphi^{eq}$.

- We call $\rightarrow$ the reduction relation induced by $\varphi^{eq}$.
Reduction relations and reductions: basic concepts

- Let $S$ be a set of constants and $\rightarrow \subseteq S \times S$
- A reduction of $\rightarrow$ is a (possibly infinite) sequence
  \begin{align*}
  s_1, s_2, \ldots, s_n, s_{n+1}, \ldots
  \end{align*}
  such that $s_i \rightarrow s_{i+1}$ for $i = 1, 2, \ldots, n, \ldots$
- To emphasize that $s_i \rightarrow s_{i+1}$ for $i = 1, 2, \ldots, n, \ldots$, we will also write reductions as follows:
  \begin{align*}
  s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_n \rightarrow s_{n+1} \rightarrow \ldots
  \end{align*}

Example: if $\rightarrow := \{ c_1 \rightarrow c_2, c_2 \rightarrow c_3, c_3 \rightarrow c_1, c_2 \rightarrow c_4, c_4 \rightarrow c_6 \}$, then

\begin{align*}
  c_1 \rightarrow \overline{c_2} \rightarrow \overline{c_3} \rightarrow c_1 \rightarrow \cdots & \quad \text{infinite reduction} \\
  c_1 \rightarrow \overline{c_2} \rightarrow c_4 \rightarrow c_6 & \quad \text{finite reduction}
\end{align*}
A decision procedure for $\varphi^{eq} \land \varphi^{diseq}$

1. Consider the following set of inference rules

   \[ CP : \quad S \cup \{ l = r', l = r \} \vdash S \cup \{ l = r', l = r, r' = r \} \quad \text{if } l \succ r, r' \& r' \succ r \]

   \[ TR : \quad S \cup \{ t = t \} \vdash S \]

   \[ DH : \quad S \cup \{ l = r', l \neq r \} \vdash S \cup \{ l = r', r' \neq r \} \quad \text{if } l \succ r, r' \& r' \succ r \]

   \[ UN : \quad S \cup \{ t \neq t \} \vdash \Box \]

2. If $\varphi^{eq} \land \varphi^{diseq} \vdash_* \Box$, then return unsatisfiable

3. Otherwise, return satisfiable
Instead of considering all equalities first, the rules allow us to interleave the processing of equalities and disequalities: this allows us the early detection of inconsistencies (if any).

With a fixed (during the application of the rules) ordering $\succ$ on constants, the number of possible applications of rules is quadratic in the number of constants (worst case).

$CP$ (critical pair) is also called Superposition and $DH$ (disequality handler) is called Paramodulation when considering general clauses.
QUESTION: can we reuse the previously introduced techniques to check the satisfiability of conjunctions of equational literals built out of function symbols?

ANSWER: yes, by using a simple trick and extending the set of inference rules introduced above
**Trick: flattening**

- Flatten terms by introducing “fresh” constants, e.g.

  \[
  \{ f(f(f(a))) = b \} \rightsquigarrow \{ f(a) = c_1, f(f(c_1)) = b \} \\
  \rightsquigarrow \{ f(a) = c_1, f(c_1) = c_2, f(c_2) = b \} \\
  \{ g(h(d))) \neq a \} \rightsquigarrow \{ h(a) = c_1, g(c_1) \neq a \} \\
  \rightsquigarrow \{ h(a) = c_1, g(c_1) = c_2, c_2 \neq a \}
  \]

- **Exercise**: show that this transformation preserves satisfiability

- The number of constants introduced is equal to the number of sub-terms occurring in the input set of literals

- **Key observation**: after flattening, literals are “close” to literals built out of constants only... we need to take care of substitution in a very simple way...
The extended set of inference rules

\begin{align*}
\text{CP} & \quad \frac{c = c' \quad c = d}{c' = d} \quad \text{if } c \succ c' \text{ and } c \succ d \\
\text{Cong}_1 & \quad \frac{c_j = c_j' \quad f(c_1, \ldots, c_j, \ldots, c_n) = c_{n+1}}{f(c_1, \ldots, c_j', \ldots, c_n) = c_{n+1}} \quad \text{if } c_j \succ c_j' \\
\text{Cong}_2 & \quad \frac{f(c_1, \ldots, c_n) = c_{n+1}' \quad f(c_1, \ldots, c_n) = c_{n+1}}{c_{n+1} = c_{n+1}'} \\
\text{DH} & \quad \frac{c = c' \quad c \neq d}{c' \neq d} \quad \text{if } c \succ c' \text{ and } c \succ d \\
\text{UN} & \quad \frac{c \neq c}{\Box}
\end{align*}

Notice that we only need to compare constants!
A decision procedure for conjunctions of arbitrary equational literals

1. Flatten literals
2. Exhaustive application of the rules in the previous slide
3. If $\square$ is derived, then return unsatisfiable
4. Otherwise, return satisfiable

In the worst case, the complexity is quadratic in the number of sub-terms occurring in the input set of equational literals

Exercise: explain why.
You can do better (i.e. $O(n \log n)$) by using a dynamic ordering over constants...