Decision procedures for the theory of equality

Silvio Ranise & Christophe Ringeissen

LORIA-INRIA Nancy Grand Est
**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 rewriting (i.e. using oriented equalities)
1 A motivating example

2 The $T_{UF}$-satisfiability problem
   - A fundamental tool in automated reasoning: Herbrand theorem
   - Decidability of $T_{UF}$ 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 rewriting
   - Rewriting: formal preliminaries
   - Convergent rewrite relations as dec. proc’s for equality
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!
A motivating example

An (optimizing) compiler: an example

Example (cont’d)

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

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

into

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

which avoids the re-computation of sub-expressions!
A motivating example

An (optimizing) compiler: an example

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:

$$
\begin{align*}
    y_{s0} &= y_{t0} \land z_{s0} = z_{t0} \\
    x_{s1} &= g(g(f(y_{s0}, z_{s0}), f(y_{s0}, z_{s0})), g(f(z_{s0}, y_{s0}), f(z_{s0}, y_{s0}))) \\
    aux_{1t1} &= f(y_{t0}, z_{t0}) \\
    aux_{2t2} &= f(z_{t0}, y_{t0}) \\
    x_{t3} &= g(g(aux_{1t1}, aux_{1t1}), g(aux_{2t2}, aux_{2t2}))
\end{align*}
\Rightarrow x_{s1} = x_{t3}
$$
The satisfiability problem for equational formulae

Definition
Let $\Sigma$ be a set of function and constant symbols. An **equational atom** is of form $s = t$ where $s, t$ are $\Sigma$-terms. An **equational formula** is a Boolean combination of equational atoms.

**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?
The $T_{UF}$-satisfiability problem

$T_{UF}$: An example

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

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

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

A fundamental tool in automated reasoning: Herbrand theorem

Arbitrary structures versus Herbrand structures

Validity versus Satisfiability:
Given a sentence \( \varphi \), \( T \models \varphi \) iff \( T \cup \{ \neg \varphi \} \) is inconsistent

Problem: search for a model of the sentence \( \phi = (T \cup \{ \neg \varphi \}) \)
For some particular sentences \( \phi \), one can restrict without loss of generality to the subclass of models of \( \phi \) that are Herbrand structures

Given any structure \( M \) such that \( M \models \phi \), it is always possible to find a Herbrand structure \( H \) such that \( H \models \phi \)
The $T_{UF}$-satisfiability problem

A fundamental tool in automated reasoning: Herbrand theorem

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:

$$I_{\mathcal{H}}(c) := \begin{cases} c \text{ if } c \text{ is a constant in } \psi \\ I_{\mathcal{H}}(f(t_1, \ldots, t_n)) := \text{mapping the } n\text{-tuple of terms } (t_1, \ldots, t_n) \text{ to the term } f(t_1, \ldots, t_n) \end{cases}$$
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 $\mathcal{H} \models \phi$ if and only if $\mathcal{M} \models \phi$.
The $T_{UF}$-satisfiability problem

A fundamental tool in automated reasoning: Herbrand theorem

Herbrand method (to refute formulae)

- **Input**: $\forall x_1, \ldots, x_k.\psi$ where $\psi$ is a quantifier-free Boolean combination of atoms
- **Output**: satisfiable/unsatisfiable
- **Method**: Consider the Herbrand universe $UH$ of $\psi$ and enumerate the ground instances of $\psi$ obtained by replacing the variables of $\psi$ by terms in $UH$:

$$Gnd(\psi) = \{ \sigma(\psi) \mid \text{Dom}(\sigma) = \{x_1, \ldots, x_k\}, \text{Ran}(\sigma) \subseteq UH\}$$

1. $G := \emptyset$
2. while there exists some $\psi'$ in $Gnd(\psi) \setminus G$ do
   (i) $G := G \cup \{\psi'\}$
   (ii) If the Boolean abstraction of $G$ is an unsatisfiable Boolean formula, then return *unsatisfiable* (and the method terminates)
3. return *satisfiable*
The formula $\forall x_1, \ldots, x_k. \psi$ is consistent iff $Gnd(\psi)$ is consistent.

Remark: $Gnd(\psi)$ is usually an infinite theory.

- 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 formulae is satisfiable iff every finite set $\Delta \subseteq \Gamma$ is satisfiable.
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: NP-problem
- In practice: Davis-Putnam-Logemann-Loveland (DPLL) algorithm, whose input is a conjunction of clauses, where a clause is a disjunction of literals
- For Horn clauses: linear-time (in the number of Boolean variables) algorithm exists
  
  A Horn clause is a disjunction of literals containing at most one positive literal.

  Thus, a Horn clause is of the form \((a_1 \land \cdots \land a_n) \Rightarrow a_{n+1}\), where \(a_i\) is an atom for \(i = 1, \ldots, n + 1\)

A detailed presentation in Lecture 6
DPLL: abstract description

Let $S$ be a set of clauses

1. **Unit Resolution**
   \[
   \frac{S \cup \{L, C \vee \overline{L}\}}{S \cup \{L, C\}} \quad \text{if} \quad \overline{A} := A, \quad \overline{\overline{A}} := A
   \]

2. **Unit Subsumption**
   \[
   \frac{S \cup \{L, C \vee L\}}{S \cup \{L\}}
   \]

3. **Splitting**
   \[
   \frac{S}{S \cup \{A\} \mid S \cup \{\overline{A}\}} \quad \text{if} \ A \text{ is an atom occurring in } S
   \]

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

- in first-order logic: the symbol of equality $=\,$, is \textbf{uninterpreted} (it is an arbitrary binary predicate symbol, written infix)
- in first-order logic with equality: the symbol of equality $=\,$, is \textbf{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...
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
Application of the theorem: a semi-decision procedure for $T_{UF}$

- The theorem allows us to use Herbrand method to solve arbitrary $T_{UF}$-satisfiability problems.
- Given an equational formula $\varphi$:
  1. compute the set $\Sigma$ of function and constant symbols occurring 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 symbols 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

- **BIG QUESTION**: can we turn the semi-decision procedure based on Herbrand method into a decision procedure
- **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 $T_{UF}$-satisfiability problem
  \[
  \varphi \equiv f(f(f(a))) = a \land f(f(f(f(a)))) = a \land f(a) \neq a
  \]
  unsatisfiable?
- By substituting equal by equal, we can derive a contradiction:

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

  Contradiction!

- **Key observation**: in deriving the contradiction, we have only used terms and sub-terms which occur in the input formula $\varphi$!
A $T_{UF}$-satisfiability procedure

**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 occurring 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, $T_{UF}$ is decidable.
Idea of the proof of theorem

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

consider the counter-positive...

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

1. \( \varphi \land \text{GEQ}_\Sigma^\varphi \) is sat. \( \Rightarrow \) \( \varphi \land \text{EQ}_\Sigma \) is sat.

Assume \( \varphi \land \text{GEQ}_\Sigma^\varphi \). So, there must exist a Herbrand structure \( M = (D_M, I_M) \) satisfying both \( \varphi \) and \( \text{GEQ}_\Sigma^\varphi \).

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 \( \varphi \land \text{GEQ}_\Sigma^\varphi \), we have that \( M' \models a \) iff \( M \models a \). Hence, \( M' \models \varphi \land \text{GEQ}_\Sigma^\varphi \).
(cont’d from previous slide)
Since $l_{M'}(t) = \#$ for all $t \in D_{M'}$ not occurring in $\varphi$, we can check that any formula in $Gnd(EQ_\Sigma) \setminus GEQ_\Sigma^\varphi$ is true in $M'$. Hence, all ground instances of $EQ_\Sigma$ are true in $M'$, and so $M' \models EQ_\Sigma$. Consequently, $M' \models EQ_\Sigma$ and $M' \models \varphi$. Thus, $\varphi \land EQ_\Sigma$ is satisfiable.

$\varphi \land EQ_\Sigma$ is sat. $\Rightarrow \varphi \land GEQ_\Sigma^\varphi$ is sat.
Easy
Complexity of $T_{UF}$ and the designed decision procedure

- $T_{UF}$ is in 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 literal)...
- 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 $\varphi \land GEQ_\Sigma$?
Assume $\varphi$ contains a number of atoms linear in the number of constants $n$ in $\varphi$.

$GEQ^\varphi_\Sigma$ will contain

1. a linear number of Boolean variables from instantiating: $\forall x. (x = x)$
2. a quadratic number of Boolean variables from instantiating: $\forall x, y. (x = y \Rightarrow y = x)$
3. a cubic number of Boolean variables from instantiating: $\forall x, y, z. (x = y \land y = z \Rightarrow x = z)$

this leads to a decision procedure with 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)

**IDEA**: represent the binary relation as a graph, to handle transitivity
Equality as a binary relation

- If we consider equality as a binary relation and represent it by means of a graph, then
  - checking the unsatisfiability of a conjunction of equational literals amounts to checking whether there exists a disequality $c \neq d$ such that the vertices $c$ and $d$ 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 (cf. Tarjan)

NB: linear complexity if the number of edges/equations is assumed to be linear in the number of nodes/variables.
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 the theory 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

Definition

A theory $T$ is said to be convex if for any $T$-satisfiable set of equalities $\Gamma$, we have $T \models (\Gamma \Rightarrow \bigvee_{i=1}^{n} s_i = t_i)$ implies there exists some $k \in [1, n]$ such that $T \models (\Gamma \Rightarrow s_k = t_k)$.
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 loosing 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.

$c \rightarrow c_1$

$c \rightarrow c_2$

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 into rewrite rules in such a way that we can still show the satisfiability of sets of literals over constants without losing refutation 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 rewrite relation induced by $\varphi^{eq}$.
Rewrite relations: derivation

- Let \( S \) be a set of constants and \( \rightarrow \subseteq S \times S \).
- A derivation w.r.t. \( \rightarrow \) is a (possibly infinite) sequence
  \[ S_1, S_2, \ldots, S_n, S_{n+1}, \ldots \]
  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 derivations as follows:
  \[ S_1 \rightarrow S_2 \rightarrow \ldots \rightarrow S_n \rightarrow S_{n+1} \rightarrow \ldots \]

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

- \( c_1 \rightarrow c_2 \rightarrow c_3 \rightarrow c_1 \rightarrow \cdots \) infinite derivation
- \( c_1 \rightarrow c_2 \rightarrow c_4 \rightarrow c_6 \) finite derivation
Let $S$ be a set of constants and $\rightarrow \subseteq S \times S$

- $\rightarrow$ is **terminating** if there is no infinite sequence $s_1 \rightarrow s_2 \rightarrow \cdots$
- $\rightarrow$ is **confluent** (or Church-Rosser) if $\leftarrow^* \circ \rightarrow^* \subseteq \rightarrow^* \circ \leftarrow^*$
- $\rightarrow$ is **locally confluent** if $\leftarrow \circ \rightarrow^* \subseteq \rightarrow^* \circ \leftarrow^*$
- A rewrite relation $\rightarrow$ is **convergent** if $\rightarrow$ is confluent and terminating
Rewrite relations: some important properties

- **Lemma.** If $\rightarrow$ is convergent, then for every $c$ there exists a unique normal form denoted with $nf(c)$.

- **Key observation:** consider the problem of checking the unsatisfiability of $\varphi^{eq} \land c \neq d$
  1. let $\rightarrow$ be the rewrite relation associated with $\varphi^{eq}$
  2. if $\rightarrow$ is convergent, then rewrite $c$ to $nf(c)$ and $d$ to $nf(d)$
  3. if $nf(c)$ is identical to $nf(d)$, then return *unsatisfiable*
  4. otherwise, return *satisfiable*

Two key features of convergent rewrite relations:
- termination guarantees that the computation terminates
- confluence allows “*don’t-care*” choice in the order of rewrite steps
Rewrite relations: exercises

1. Prove the lemma in the previous slide
   **Hint:** By contradiction, assume that for some $c$ there exist $c_1, c_2$ such that $c \rightarrow^* c_1$ and $c \rightarrow^* c_2$ with $c_i$ in normal form for $i = 1, 2$. Recall the definition for an element being in normal form. Then, remember that $\rightarrow$ is confluent by assumption and so there must exist and element $d$ such that $c_i \rightarrow^* d$ for $i = 1, 2$ and derive the contradiction.

2. Let $\rightarrow := \{(c_1, c_2), (c_2, c_3), (c_3, c_5), (c_2, c_4), (c_4, c_5)\}$.
   1. Find all possible derivations from $c_1$ to $c_5$
   2. Show that $c_5$ is the normal form of $c_1$
   3. Show that $\rightarrow$ is convergent
Convergent rewrite relations and the satisfiability problem

**QUESTION:** how can we establish that $\rightarrow$ is convergent?

**ANSWER:** Newmann’s Lemma. *A terminating and locally confluent relation is confluent.*

Local confluence is much easier to check than confluence: it is possible to check confluence by considering all possible ways (which are finitely many!) of rewriting an element by using an oriented equation in $\varphi^{eq}$

Example: if $\rightarrow := \{(c_1, c_2), (c_2, c_3), (c_3, c_5), (c_2, c_4), (c_4, c_5)\}$, then

\[
\begin{align*}
c_4 & \leftarrow c_2 \rightarrow c_3 \\
c_4 & \rightarrow c_5 \leftarrow c_3
\end{align*}
\]
**QUESTION**: How can ensure the termination of $\rightarrow$?

**ANSWER**: using ordering relations, which precisely formalize the idea of orienting an equality

A strict ordering $\succ$ on a set of elements is an irreflexive, antisymmetric and transitive binary relation

$\succ$ is a reduction ordering if it is a strict ordering which is also terminating:

**Exercise**: check that the reduction ordering $\succ$ on a a finite set of elements can be represented by a finite tree

**Key property**: A rewrite relation $\rightarrow$ is terminating if there exists a reduction ordering $\succ$ such that $\rightarrow$ is included in $\succ$
Towards confluent rewrite relations

- **Observation**: once termination has been established, we apply Newmann’s Lemma and check for local confluence.
- Let $\succ$ be a reduction ordering such that $\rightarrow \subseteq \succ$.
- We now analyze in which situation a rewrite relation is not locally confluent...
How to get local confluence?

- Assume a constant $t$ can be rewritten in two different ways: $t \rightarrow r$ and $t \rightarrow r'$, respectively.
- Assume (without loss of generality) that $r \succ r'$.
- To restore local confluence, we can add the pair $(r, r')$ to $\rightarrow$.
- **Observation:** we can do this since $r \leftrightarrow^* r'$.
Computing locally confluent rewrite relations

- we say that $t \rightarrow r$ and $t \rightarrow r'$ is a **critical pair**
- **Key idea**: successively discover critical pairs until no more critical pairs are produced
- To do this, we have to detect all identical left-hand-sides of the rewrite relation $\rightarrow$
- **Termination of adding critical pairs**: when the rewrite relation $\rightarrow$ is induced by $\varphi^{eq}$ the process must terminate since the set of possible critical pairs is bounded by $S \times S$, where $S$ is the set of constants in $\varphi^{eq}$
1 Consider the following set of inference rules

\[
\begin{align*}
\text{CP} & \quad \frac{c = c'}{c' = d} \quad \text{if } c \succ c' \text{ and } c \succ d \\
\text{DH} & \quad \frac{c = c'}{c' \neq d} \quad \text{if } c \succ c' \text{ and } c \succ d \\
\text{UN} & \quad \frac{c \neq c'}{} \quad \text{if } c \neq c
\end{align*}
\]

2 if \( \varphi^{eq} \land \varphi^{diseq} \vdash^* \square \), then return \textit{unsatisfiable}

3 otherwise, return \textit{satisfiable}
A decision procedure: remarks

- 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.

\[
\{ f(f(f(a))) = b \} \leadsto \{ f(a) = c_1, f(f(c_1)) = b \} \\
\leadsto \{ f(a) = c_1, f(c_1) = c_2, f(c_2) = b \}
\]

\[
\{ g(h(a)) \neq a \} \leadsto \{ h(a) = c_1, g(c_1) \neq a \} \\
\leadsto \{ h(a) = c_1, g(c_1) = c_2, c_2 \neq a \}
\]

- **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

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

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 □ 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 [Armando et al., 2003]

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

See [Nelson and Oppen, 1980, Nieuwenhuis and Oliveras, 2007]
References

A rewriting approach to satisfiability procedures.

Fast decision procedures based on congruence closure.

Fast congruence closure and extensions.