Decision procedures for the theory of equality

Silvio Ranise & Christophe Ringeissen

LORIA-INRIA Nancy Grand Est

Topics

- 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)

Index

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

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:

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

into

```
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!

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. $+ \rightsquigarrow f$ and $* \rightsquigarrow g$).
Then, prove the validity of the following proof obligation:

$$\left(\begin{array}{ll} y_{s0} = y_{t0} \wedge z_{s0} = z_{t0} & \wedge \\ x_{s1} = g(g(f(y_{s0}, z_{s0}), f(y_{s0}, z_{s0})), g(f(z_{s0}, y_{s0}), f(z_{s0}, y_{s0}))) & \wedge \\ aux1_{t1} = f(y_{t0}, z_{t0}) & \wedge \\ aux2_{t2} = f(z_{t0}, y_{t0}) & \wedge \\ x_{t3} = g(g(aux1_{t1}, aux1_{t1}), g(aux2_{t2}, aux2_{t2})) & \end{array} \right) \Rightarrow x_{s1} = x_{t3}$$

The satisfiability problem for equational formulae

Definition

Let Σ be a set of function and constant symbols. An **equational atom** is of form $s = t$ where s, t are Σ-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?

T_{UF}: An example

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

$$\left(\begin{array}{ll} y_{s0} = y_{t0} \wedge z_{s0} = z_{t0} & \wedge \\ x_{s1} = g(g(f(y_{s0}, z_{s0}), f(y_{s0}, z_{s0})), g(f(z_{s0}, y_{s0}), f(z_{s0}, y_{s0}))) & \wedge \\ aux1_{t1} = f(y_{t0}, z_{t0}) & \wedge \\ aux2_{t2} = f(z_{t0}, y_{t0}) & \wedge \\ x_{t3} = g(g(aux1_{t1}, aux1_{t1}), g(aux2_{t2}, aux2_{t2})) & \end{array}\right) \wedge x_{s1} \neq x_{t3}$$

which is indeed an equational formula whose atoms are built out of the symbols in $\Sigma := \{f/2, g/2, x_{s0}/0, y_{s0}/0, x_{t0}/0, y_{t0}/0, ...\}$

Arbitrary structures versus Herbrand structures

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

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

Given any structure $\mathcal{M}$ such that $\mathcal{M} \models \phi$, it is always possible to find a Herbrand structure $\mathcal{H}$ such that $\mathcal{H} \models \phi$

Herbrand universe: UH

Assume the following form $\boxed{\forall x_1, ..., x_k . \psi}$
where ψ is a Boolean combination of atoms without quantifiers

- $\mathrm{UH}_0 :=$ constants occurring in ψ
  - if there are no constants in ψ, then $\mathrm{UH}_0 := \{a\}$ (for a an arbitrary constant symbol)
- $\mathrm{UH}_{i+1} := \mathrm{UH}_i \cup \{f(t_1, ..., t_n) | f \text{ is in } \psi \text{ of arity } n \text{ and } t_1, ..., t_n \in \mathrm{UH}_i\}$
- The **Herbrand universe** is defined as follows:

$$\mathrm{UH} := \bigcup_{i=0}^{\infty} \mathrm{UH}_i$$

Herbrand structures

Definition

The Herbrand structure $\mathcal{H} = \langle \mathcal{D}_{\mathcal{H}}, \mathcal{I}_{\mathcal{H}} \rangle$ of $\forall x_1, ..., x_k . \psi$ (where ψ is a Boolean combination of atoms without quantifiers) is such that

- $\mathcal{D}_{\mathcal{H}}$ is the Herbrand universe of ψ
- $\mathcal{I}_{\mathcal{H}}$ is defined on (ground) terms as follows:

$$\begin{array}{rcl} \mathcal{I}_{\mathcal{H}}(c) & := & c \text{ if } c \text{ is a constant in } \psi \\ \mathcal{I}_{\mathcal{H}}(f(t_1, ..., t_n)) & := & \text{mapping the } n\text{-tuple of terms } (t_1, ..., t_n) \\ & & \text{to the term } f(t_1, ..., t_n) \end{array}$$

Herbrand theorem

Theorem

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

Proof.

($\Leftarrow$): obvious.
($\Rightarrow$): Let $\mathcal{M}$ be a model of $\phi = (\forall x_1, ..., x_k.\psi)$. We can define an interpretation over atoms $p(t_1 \ldots, t_n)$ where $t_1, \ldots, t_n \in \mathcal{D}_{\mathcal{H}}$:
$p(t_1 \ldots, t_n)$ 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

$$\mathcal{H} \models \phi \text{ if and only if } \mathcal{M} \models \phi$$

$\square$

Herbrand method (to refute formulae)

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

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

1. $G := \emptyset$
2. while there exists some ψ' 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*

Herbrand method: remarks

The formula $\forall x_1, ..., x_k.\psi$ 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 Γ of 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 ≥ 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 occurrences 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 $(a_1 \wedge \cdots \wedge a_n) \Rightarrow a_{n+1}$, 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

Unit Resolution	$\dfrac{S \cup \{L, C \vee \overline{L}\}}{S \cup \{L, C\}}$	if $\begin{array}{lcl} \overline{\neg A} & := & A \\ \overline{A} & := & \neg A \end{array}$
Unit Subsumption	$\dfrac{S \cup \{L, C \vee L\}}{S \cup \{L\}}$	
Splitting	$\dfrac{S}{S \cup \{A\} \mid S \cup \{\neg A\}}$	if A is an atom occurring in S

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

Herbrand method and T_{UF}

- 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 φ be an equational formula built out of the symbols in Σ
- Consider the following set EQ_Σ of axioms saying that $=$ is a **congruence relation**:

$$\begin{aligned}
\forall x.(x = x) & \\
\forall x, y.(x = y \Rightarrow y = x) & \\
\forall x, y, z.(x = y \wedge y = z \Rightarrow x = z) & \\
\forall ...x, y...(x = y \Rightarrow f(...x...) = f(...y...)) & \quad \text{for each } f \in \Sigma
\end{aligned}$$

Remark: φ is satisfiable in first-order logic **with** equality iff $\varphi \wedge EQ_\Sigma$ 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}-satisfiabillity problems
- Given an equational formula φ:
  1. compute the set Σ of function and constant symbols occurring in φ
  2. compute the set EQ_Σ
  3. return the result of applying the Herbrand method on $\varphi \wedge EQ_\Sigma$ (where $=$ is considered as an arbitrary predicate symbol)
- About termination: it is sufficient that Σ contains one non-constant symbols that the Herbrand universe of $\varphi \wedge EQ_\Sigma$ 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

$$\varphi \quad \equiv \quad f(f(f(a))) = a \wedge f(f(f(f(f(a))))) = a \wedge f(a) \neq a$$

unsatisfiable?

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

$$\begin{aligned}
\underline{f(f(f(a)))} = a \wedge f(f(\underline{f(f(f(a)))})) = a \wedge f(a) \neq a \\
f(f(f(a))) = a \wedge f(f(a)) = a \wedge f(a) \neq a \\
f(\underline{f(f(a))}) = a \wedge \underline{f(f(a))} = a \wedge f(a) \neq a \\
\boxed{f(a) = a} \wedge f(f(a)) = a \wedge \boxed{f(a) \neq a}
\end{aligned}$$

Contradiction!

- **Key observation**: in deriving the contradiction, we have only used terms and sub-terms which occur in the input formula φ!

A T_{UF}-satisfiability procedure

Theorem

$\varphi \wedge EQ_\Sigma$ is unsatisfiable iff $\varphi \wedge GEQ_\Sigma^\varphi$ is unsatisfiable, where GEQ_Σ^φ is the (finite) set of ground instances of EQ_Σ obtained by instantiating variables with all terms and sub-terms occurring in φ.

Corollary

Given an equational formula φ. The following algorithm

1. *compute the set Σ of function and constant symbols occurring in φ*
2. *compute the set GEQ_Σ^φ*
3. *return the result of checking the (Boolean) satisfiability of $\varphi \wedge GEQ_\Sigma^\varphi$*

terminates and returns whether φ is satisfiable or not.
Hence, T_{UF} is decidable.

Idea of the proof of theorem

$\varphi \wedge EQ_\Sigma$ is unsat. $\Rightarrow \varphi \wedge GEQ_\Sigma^\varphi$ is unsat.

consider the counter-positive...

$\varphi \wedge GEQ_\Sigma^\varphi$ is sat. $\Rightarrow \varphi \wedge EQ_\Sigma$ is sat.

Proof of theorem

1. $\varphi \wedge GEQ_{\Sigma}^{\varphi}$ is sat. $\Rightarrow \varphi \wedge EQ_{\Sigma}$ is sat.
   Assume $\varphi \wedge GEQ_{\Sigma}^{\varphi}$. So, there must exist a Herbrand structure $M = (D_M, I_M)$ satisfying both φ and GEQ_{Σ}^{φ}.
   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 φ, we have that $I_{M'}(t) = I_M(t)$ by construction, we derive that each equational atom a in $\varphi \wedge GEQ_{\Sigma}^{\varphi}$, we have that $M' \models a$ iff $M \models a$. Hence, $M' \models \varphi \wedge GEQ_{\Sigma}^{\varphi}$

Proof of theorem

1. (cont'd from previous slide)
   Since $I_{M'}(t) = \#$ for all $t \in D_{M'}$ not occurring in φ, we can check that any formula in $Gnd(EQ_\Sigma) \setminus GEQ_\Sigma^\varphi$ is true in M'.
   Hence, all ground instances of EQ_Σ are true in M', and so $M' \models EQ_\Sigma$.
   Consequently, $M' \models EQ_\Sigma$ and $M' \models \varphi$. Thus, $\varphi \wedge EQ_\Sigma$ is satisfiable.
2. $\varphi \wedge EQ_\Sigma$ is sat. $\Rightarrow \varphi \wedge GEQ_\Sigma^\varphi$ 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 occurrences of Boolean variables...
- **QUESTION**: how many occurrences of Boolean variables are in $\varphi \wedge GEQ_{\Sigma}^{\varphi}$?

Complexity of the designed decision procedure

- Assume φ contains a number of atoms linear in the number of constants n in φ.
- GEQ_{Σ}^{φ} will contain
  1. a linear number of occurrences of Boolean variables from instantiating: $\forall x.(x = x)$
  2. a quadratic number of occurrences of Boolean variables from instantiating: $\forall x, y.(x = y \Rightarrow y = x)$
  3. a cubic number of occurrences 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/constants

A better decision procedure for conjunctions of equational literals

- Let φ be a conj. of equational literals of the form $c = d$ or $\neg c = d$
  1. let φ^{eq} be the conjunction of all equalities and φ^{diseq} be the conjunctions of all disequalities in φ
  2. build the graph G associated with φ^{eq}
  3. let $c \neq d$ be a disequality in φ^{diseq}:
     - if c and d are connected in G, then return *unsatisfiable*
     - otherwise, consider another disequality in φ^{diseq}
  4. when all diseq. in φ^{diseq} have been considered, return *satisfiable*
- If the number of atoms in φ is linear in the number of constants in φ, then the running time of the algorithm will be quadratic in the number of constants in φ...
- **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 $\varphi^{eq} \wedge \varphi^{diseq}$ of equational literals is unsatisfiable, then there must exist just one disequality $c \neq d$ in φ^{diseq} such that $\varphi^{eq} \wedge c \neq d$ is unsatisfiable

Definition

A theory T is said to be *convex* if for any T-satisfiable set of equalities Γ, 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 $c = c_1 \wedge c = c_2 \wedge c_1 \neq c_2$
  Now, orient the two equalities from left-to-right, i.e.

$$\begin{array}{rcl} c & \rightarrow & c_1 \\ c & \rightarrow & c_2 \end{array}$$

  and consider the **reflexive and transitive** closure $\rightarrow^*$ of $\rightarrow$.
  Unfortunately: $c_1 \not\rightarrow^* c_2$. So, $\rightarrow^* \subset =$ 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 loosing refutation completeness
- Formally, we introduce a binary relation $\rightarrow$ (to emphasize that it is an oriented version of $=$) on the constants in φ^{eq}
- We call $\rightarrow$ the rewrite relation induced by φ^{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

$$s_1, s_2, ..., s_n, s_{n+1}, ...$$

such that $s_i \rightarrow s_{i+1}$ for $i = 1, 2, ..., n, ...$

- To emphasize that $s_i \rightarrow s_{i+1}$ for $i = 1, 2, ..., n, ...$, we will also write derivations as follows:

$$s_1 \rightarrow s_2 \rightarrow ... \rightarrow s_n \rightarrow s_{n+1} \rightarrow ...$$

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

$$c_1 \rightarrow \underline{c_2 \rightarrow} c_3 \rightarrow c_1 \rightarrow \cdots \quad \text{infinite derivation}$$

$$c_1 \rightarrow \underline{c_2 \rightarrow} c_4 \rightarrow c_6 \quad \text{finite derivation}$$

Rewrite relations: definitions

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 $\varphi^{eq} \wedge c \neq d$
  1. let $\rightarrow$ be the rewrite relation associated with φ^{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 $c \to^* c_1$ and $c \to^* c_2$ with c_i in normal form for $i = 1, 2$. Recall the definition for an element being in normal form. Then, remember that $\to$ is confluent by assumption and so there must exist and element d such that $c_i \to^* d$ for $i = 1, 2$ and derive the contradiction.
2. Let $\to := \{(c_1, c_2), (c_2, c_3), (c_3, c_5), (c_2, c_4), (c_4, c_5)\}$.
   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 $\to$ 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 φ^{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

$$\begin{array}{ccc} c_4 \leftarrow c_2 \rightarrow & c_3 \\ c_4 \rightarrow c_5 \leftarrow & c_3 \end{array}$$

Towards terminating rewrite relations

- **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: no infinite decreasing chain $e_1 \succ e_2 \succ \cdots$
- **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

Consider $\rightarrow$ is a rewrite relation over a finite set of constants S and $\succ$ is an ordering over S such that $\rightarrow \subseteq \succ$ and $\succ$ is total on S, e.g.,

$$e \succ d \succ c \succ b \succ a \quad \text{for } S = \{a, b, c, d, e\}$$

Then $\succ$ is necessarily a reduction ordering and so $\rightarrow$ is terminating. By Newmann's Lemma, one can now check for local confluence.

Let us now analyze in which situation a rewrite relation is not locally confluent...

How to get local confluence?

- Assume a constant c can be rewritten in two different ways:
  $$c \to d \text{ and } c \to c', \text{ respectively}$$
- To restore local confluence, we can add the equality $c' = d$. Then $c' = d$ can be oriented as the rewrite rule $c' \to d$ id $c' \succ d$ and as $d \to c'$ if $d \succ c'$
- **Observation**: $\varphi^{eq} \models c' = d$

Computing locally confluent rewrite relations

- we say that $c \to d$ and $c \to c'$ overlap and the overlapped constant c generates the **critical pair** $c' = d$
- Key idea: successively discover overlapped terms until no more critical pairs are produced
- To do this, we have to detect all identical left-hand-sides of the rewrite relation $\to$
- **Termination of adding critical pairs**: the process terminates since the number of critical pairs is bounded by $|S \times S|$, where S is the set of constants in φ^{eq}

A decision procedure for $\varphi^{eq} \wedge \varphi^{diseq}$

1. Consider the following set of inference rules

$$\text{CP} \quad \frac{c = c' \qquad c = d}{c' = d} \quad \text{if } c \succ c' \text{ and } c \succ d$$

$$\text{DH} \quad \frac{c = c' \qquad c \neq d}{c' \neq d} \quad \text{if } c \succ c' \text{ and } c \succ d$$

$$\text{UN} \quad \frac{c \neq c}{\square}$$

2. if $\varphi^{eq} \wedge \varphi^{diseq} \vdash^* \square$, then return *unsatisfiable*
3. otherwise, return *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

What about a more general satisfiability problem?

- **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.

$$
\begin{array}{rcl}
\{f(f(f(a))) = b\} & \leadsto & \{f(a) = c_1, f(f(c_1)) = b\} \\
 & \leadsto & \{f(a) = c_1, f(c_1) = c_2, f(c_2) = b\} \\
\{g(h(a)) \neq a\} & \leadsto & \{h(a) = c_1, g(c_1) \neq a\} \\
 & \leadsto & \{h(a) = c_1, g(c_1) = c_2, c_2 \neq a\}
\end{array}
$$

- 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

CP	$$\frac{c = c' \qquad c = d}{c' = d}$$	if $c \succ c'$ and $c \succ d$
Cong_1	$$\frac{c_j = c'_j \qquad f(c_1, \ldots, c_j, \ldots, c_n) = c_{n+1}}{f(c_1, \ldots, c'_j, \ldots, c_n) = c_{n+1}}$$	if $c_j \succ c'_j$
Cong_2	$$\frac{f(c_1, \ldots, c_n) = c'_{n+1} \qquad f(c_1, \ldots, c_n) = c_{n+1}}{c_{n+1} = c'_{n+1}}$$	
DH	$$\frac{c = c' \qquad c \neq d}{c' \neq d}$$	if $c \succ c'$ and $c \succ d$
UN	$$\frac{c \neq c}{\Box}$$	

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 $\Box$ 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

Armando, A., Ranise, S., and Rusinowitch, M. (2003).
A rewriting approach to satisfiability procedures.
Inf. Comput., 183(2):140–164.

Nelson, G. and Oppen, D. C. (1980).
Fast decision procedures based on congruence closure.
J. ACM, 27(2):356–364.

Nieuwenhuis, R. and Oliveras, A. (2007).
Fast congruence closure and extensions.
Inf. Comput., 205(4):557–580.