Automated Reasoning 2018

Lecture 8: Theory of equality and uninterpreted functions(QF_EUF)

Instructor: Ashutosh Gupta

IITB, India

Compile date: 2018-08-13

Topic 8.1

Theory of equality and function symbols (EUF)

Reminder: Theory of equality and function symbols (EUF)

EUF syntax: first-order formulas with signature $\mathbf{S} = (\mathbf{F}, \emptyset)$,
i.e., countably many function symbols and no predicates.

The theory axioms include

1. $\forall x.\ x \approx x$
2. $\forall x, y.\ x \approx y \Rightarrow y \approx x$
3. $\forall x, y, z.\ x \approx y \wedge y \approx z \Rightarrow x \approx z$
4. for each $f/n \in \mathbf{F}$,

$$\forall x_1, .., x_n, y_1, .., y_n.\ x_1 \approx y_1 \wedge .. \wedge x_n \approx y_n \Rightarrow f(x_1, .., x_n) \approx f(y_1, .., y_n)$$

Note: Predicates can be easily added if desired

Commentary: Since the axioms are valid in FOL with equality, the theory is sometimes referred as the base theory.

Proofs in quantifier-free fragment of $\mathcal{T}_{EUF}$(QF_EUF)

The axioms translates to the proof rules of $\mathcal{T}_{EUF}$ as follows

$$\frac{x \approx y}{y \approx x}\,Symmetry$$

$$\frac{x \approx y \quad y \approx z}{x \approx z}\,Transitivity$$

$$\frac{x_1 \approx y_1 \quad .. \quad x_n \approx y_n}{f(x_1, .., x_n) \approx f(y_1, .., y_n)}\,Congruence$$

Example 8.1

Consider: $y \approx x \wedge y \approx z \wedge f(x, u) \not\approx f(z, u)$

$$\cfrac{\cfrac{\cfrac{\cfrac{y \approx x}{x \approx y} \quad y \approx z}{x \approx z}}{f(x, u) \approx f(z, u)} \quad f(x, u) \not\approx f(z, u)}{\perp}$$

Exercise: equality with uninterpreted functions

Exercise 8.1

If unsat, give proof of unsatisfiability

- $f(f(c)) \not\approx c \wedge f(c) \approx c$
- $f(f(c)) \approx c \wedge f(c) \not\approx c$
- $f(f(c)) \approx c \wedge f(f(f(c))) \not\approx c$
- $f^3(a) = a \wedge f^5(a) = a \wedge f(a) \neq a$

Topic 8.2

QF_EUF solving via SAT solver

Eager solving

Explicate all the theory reasoning as Boolean clauses.

Use SAT solver alone to check satisfiability.

Only possible for the theories, where we can bound the relevant instantiations of the theory axioms.

The eager solving for QF_EUF is called Ackermann's Reduction.

Notation: term encoder

Let *en* be a function that maps terms to new constants.

We can apply *en* on a formula to obtain a formula over the fresh constants.

Example 8.2

Consider $en = \{f(x) \mapsto t_1, f(y) \mapsto t_2, x \mapsto t_3, y \mapsto t_4\}$.

$en(x \approx y \Rightarrow f(x) \approx f(y)) = (t_3 \approx t_4 \Rightarrow t_1 \approx t_2)$

Notation: Boolean encoder

Let e be a Boolean encoder (defined in the last lecture).

Example 8.3

Consider $en = \{t_3 \approx t_4 \mapsto p_1, t_1 \approx t_2 \mapsto p_2\}$

$e(t_3 \approx t_4 \Rightarrow t_1 \approx t_2) = (p_1 \Rightarrow p_2)$

Ackermann's Reduction

The insight: the rules needed to be applied only finitely many possible ways.

Algorithm 8.1: QF_EUF_Sat(F)

Input: F formula QF_EUF

Output: SAT/UNSAT

Let Ts be subterms of F, en be $Ts \rightarrow$ fresh constants, e be a Boolean encoder;

$G := en(F)$;

foreach $f(x_1, .., x_n), f(y_1, .., y_n) \in Ts$ **do**

  $G := G \wedge en(x_1 \approx y_1 \wedge .. \wedge x_n \approx y_n \Rightarrow f(x_1, .., x_n) \approx f(y_1, .., y_n))$

foreach $t_1, t_2, t_3 \in Ts$ **do**

  $G := G \wedge en(t_1 \approx t_2 \wedge t_2 \approx t_3 \Rightarrow t_1 \approx t_3)$

foreach $t_1, t_2 \in Ts$ **do**

  $G := G \wedge en(t_1 \approx t_2 \Leftrightarrow t_2 \approx t_1)$

$G' := e(G)$;

return *CDCL(G')*

Exercise 8.2

Can we avoid clauses for the symmetry rule?

Example: Ackermann's Reduction

Example 8.4

Consider formula $F = f(f(x)) \not\approx x \wedge f(x) \approx x$

Assumed that symmetric atoms mapped to same variable.

$Ts := \{f(f(x)), f(x), x\}$.

$en := \{f(f(x)) \mapsto f_1, f(x) \mapsto f_2, x \mapsto f_3\}$

Boolean encoding:

$$\{f_1 \approx f_3 \mapsto p_1, f_2 \approx f_3 \mapsto p_2, f_1 \approx f_3 \mapsto p_3\}$$

$G := en(F) := f_1 \not\approx f_3 \wedge f_2 \approx f_3$

$G' := \neg p_1 \wedge p_2$

Adding congruence consequences:

$G := G \wedge (f_2 \approx f_3 \Rightarrow f_1 \approx f_2)$.

$G' := G' \wedge (p_2 \Rightarrow p_3)$.

Adding transitivity consequences:

$$\begin{aligned} G := G &\wedge (f_1 \approx f_2 \wedge f_2 \approx f_3 \Rightarrow f_1 \approx f_3) \\ &\wedge (f_1 \approx f_3 \wedge f_2 \approx f_3 \Rightarrow f_1 \approx f_2) \\ &\wedge (f_1 \approx f_2 \wedge f_1 \approx f_3 \Rightarrow f_2 \approx f_3). \end{aligned}$$

$$\begin{aligned} G' := G' &\wedge (p_3 \wedge p_2 \Rightarrow p_1) \\ &\wedge (p_3 \wedge p_2 \Rightarrow p_1) \\ &\wedge (p_1 \wedge p_3 \Rightarrow p_2). \end{aligned}$$

Since G' is UNSAT, F is UNSAT.

Other eager encoding

- Byrant's Encoding

Topic 8.3

QF_EUF solver for SMT

Lazy theory solver

Axioms are applied on demand

CDCL determines the required literals to be analyzed.

Theory solver applies axioms only related to the literals.

Exercise 8.3

We have seen the lazy approach in the last lecture.
How can we have a mixed lasy/eager approach?

$DP_{EUF}.push$

Algorithm 8.2: $DP_{EUF}.push(t_1 \bowtie t_2)$

globals: set of terms $Ts := \emptyset$, set of pairs of classes $DisEq := \emptyset$, bool $conflictFound := 0$

$Ts := Ts \cup subTerms(t_1) \cup subTerms(t_2)$;

$C_1 := getClass(t_1)$; $C_2 := getClass(t_2)$; // if t_i is seen first time, create new class

if $\bowtie$ = "$\approx$" **then**

  if $C_1 = C_2$ **then** **return** ;

  if $(C_1, C_2) \in DisEq$ **then** $\{$ $conflictFound := 1$; **return**; $\}$;

  $C := mergeClasses(C_1, C_2)$; $parent(C) := (C_1, C_2, t_1 \approx t_2)$;

  $DisEq := DisEq[C_1 \mapsto C, C_2 \mapsto C]$

else

  // $\bowtie$ = "$\not\approx$"

  $DisEq := DisEq \cup (C_1, C_2)$;

  if $C_1 = C_2$ **then** $conflictFound := 1$; **return** ;

foreach $f(r_1, \ldots, r_n), f(s_1, \ldots, s_n) \in Ts \wedge \forall i \in 1..n.\ \exists C.\ r_i, s_i \in C$ **do**

  $DP_{EUF}.push(f(r_1, \ldots, r_n) \approx f(s_1, \ldots, s_n))$;

Completeness is not obvious

Example 8.5

Consider: $x \approx y \wedge y \approx z \wedge f(x,u) \not\approx f(z,u)$

$$\dfrac{\dfrac{\dfrac{x \approx y}{f(x,u) \approx f(y,u)} \quad \dfrac{y \approx z}{f(y,u) \approx f(z,u)}}{f(x,u) \approx f(z,u)} \qquad f(x,u) \not\approx f(z,u)}{\bot}$$

In the proof $f(y,u)$ occurs, which does not occur in the input formula.

Commentary: Our algorithm only derives facts consists of terms that occur in the input. If the above proofs exists, does it endanger the completeness of $DP_{EUF}.push$?

Completeness of $DP_{EUF}.push$

Theorem 8.1

Let $\Sigma = \{\ell_1, .., \ell_n\}$ be a set of literals in $\mathcal{T}_{EUF}$.
$DP_{EUF}.push(\ell_1); ...; DP_{EUF}.push(\ell_n);$ *finds conflict iff Σ is unsat.*

Proof.

Since $DP_{EUF}.push$ uses only sound proof steps of the theory, it cannot find conflict if Σ is sat.

Assume Σ is unsat and there is a proof for it.

Since $DP_{EUF}.push$ applies congruence only if the resulting terms appear in Σ, we show that there is a proof that contains only such terms.

Completeness of $DP_{EUF}.push$(contd.)

Proof(contd.)

Since Σ is unsat, there is $\Sigma' \cup \{s \not\approx t\} \subseteq \Sigma$ s.t. $\Sigma' \cup \{s \not\approx t\}$ is unsat and Σ' contains only positive literals.(why?)

Consider a proof that derives $s \approx t$ from Σ'.

Therefore, we must have a proof step such that

$$\frac{u_1 \approx u_2 \quad .. \quad u_{n-1} \approx u_n}{s \approx t},$$

Flattened transitivity rule!!

where $n \geq 2$, the premises have proofs from Σ', $u_1 = s$, and $u_n = t$.

Exercise 8.4

Show the last claim holds.

Commentary: We can generalize transitivity with more than two premises. $\frac{u_1 \approx u_2 \quad u_2 \approx u_3 \quad \dots \quad u_{n-1} \approx u_n}{u_1 \approx u_n}$

Completeness of $DP_{EUF}.push$(contd.)

Proof(contd.)

Wlog, we assume $u_i \approx u_{i+1}$ either occurs in Σ' or derived from congruence.

Observation: if $u_i \approx u_{i+1}$ is derived from congruence then the top symbols are same in u_i and u_{i+1}.

Now we show that we can transform the proof via induction over height of congruence proof steps.

Exercise 8.5

Justify the "wlog" claim.

Completeness of $DP_{EUF}.push$(contd.)

Proof(contd.)

claim: If s and t occurs in Σ', any proof of $s \approx t$ can be turned into a proof that contains only the terms from Σ'

base case:
If no congruence is used to derive $s \approx t$ then no fresh term was invented.(why?)

induction step:
We need not worry about $u_i \approx u_{i+1}$ that are coming from Σ'.

Only in the subchains of the equalities due to congruences may have new terms.

Example 8.6

$$\dfrac{\dfrac{x \approx y}{f(x,u) \approx f(y,u)} \quad \dfrac{y \approx z}{f(y,u) \approx f(z,u)}}{f(x,u) \approx f(z,u)}$$

Completeness of $DP_{EUF}.push$(contd.)

Proof(contd.)

Let $f(u_{11},..,u_{1k}) \approx f(u_{21},..,u_{2k}) \quad .. \quad f(u_{(j-1)1},..,u_{(j-1)k}) \approx f(u_{j1},..,u_{jk})$ be such a maximal subchain in the last proof step for $s \approx t$.

$$\frac{s \approx \ldots \quad \dfrac{u_{11} \approx u_{21} \quad .. \quad u_{1k} \approx u_{2k}}{f(u_{11},..,u_{1k}) \approx f(u_{21},..,u_{2k})} \quad .. \quad \dfrac{u_{(j-1)1} \approx u_{j1} \quad .. \quad u_{(j-1)k} \approx u_{jk}}{f(u_{(j-1)1},..,u_{(j-1)k}) \approx f(u_{j1},..,u_{jk})} \quad \ldots \approx t}{s \approx t},$$

We know $f(u_{11},..,u_{1k})$ and $f(u_{j1},..,u_{jk})$ occur in Σ'.(why?)

For $1 < i < j$, $f(u_{i1},..,u_{ik})$ may not occur in Σ'.

Exercise 8.6

Justify the (why?).

Completeness of $DP_{EUF}.push$(contd.)

Proof(contd.)

We can rewrite the proof in the following form.

$$\dfrac{s \approx \ldots \qquad \dfrac{\dfrac{u_{11} \approx u_{21} \quad .. \quad u_{(j-1)1} \approx u_{j1}}{u_{11} \approx u_{j1}} \quad .. \quad \dfrac{u_{1k} \approx u_{2k} \quad .. \quad u_{(j-1)k} \approx u_{jk}}{u_{1k} \approx u_{jk}}}{f(u_{11},..,u_{1k}) \approx f(u_{j1},..,u_{jk})} \qquad \ldots \approx t}{s \approx t}$$

Due to induction hypothesis, for each $i \in 1..k$,

since u_{1i} and u_{ji} occur in Σ', $u_{1i} \approx u_{ji}$ has a proof with the restriction. $\square$

Example 8.7

$$\dfrac{\dfrac{x \approx y}{f(x,u) \approx f(y,u)} \qquad \dfrac{y \approx z}{f(y,u) \approx f(z,u)}}{f(x,u) \approx f(z,u)} \rightsquigarrow \dfrac{\dfrac{x \approx y \quad y \approx z}{x \approx z}}{f(x,u) \approx f(z,u)}$$

Topic 8.4

Algorithms for EUF

DP_{EUF} implementation - union-find

Equivalence classes are usually implemented using union-find data structure

- each class is represented using a tree over its member terms
- root of the tree represents the class
- getClass() returns root of the tree, which involves traversing to the root
- mergeClasses() simply adds the root of smaller tree as a child of the root of larger class

Efficient data-structure: for n pushes, run time is $O(n \log n)$

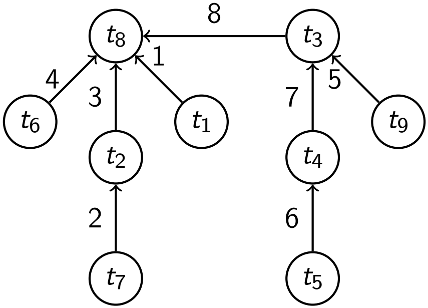
Exercise 8.7

Prove the above complexity

Example: union-find

Consider:

$$\underbrace{t_1 \approx t_8}_{1} \wedge \underbrace{t_7 \approx t_2}_{2} \wedge \underbrace{t_7 \approx t_1}_{3} \wedge \underbrace{t_6 \approx t_7}_{4} \wedge \underbrace{t_9 \approx t_3}_{5} \wedge \underbrace{t_5 \approx t_4}_{6} \wedge \underbrace{t_4 \approx t_3}_{7} \wedge \underbrace{t_7 \approx t_5}_{8} \wedge \underbrace{t_1 \not\approx t_4}_{9}$$



unsatCore using union find

- generate proof of unsatisfiablity using union find
- collect leaves of the proof, which can serve as an unsat core

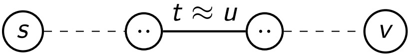
Proof generation in union-find

Proof generation from union find data structure for an unsat input.

The proof is constructed bottom up.

1. There must be a dis-equality $s \not\approx v$ that was violated.
   We need to find the proof for $s \approx v$.

2. Find the latest edge in the path between s and v. Let us say it is due to input literal $t \approx u$.



   Recursively, find the proof of $s \approx t$ and $u \approx v$.

   We stitch the proofs as follows

$$\dfrac{\dfrac{\cdots}{s \approx t} \quad t \approx u \quad \dfrac{\cdots}{u \approx v}}{s \approx v}$$

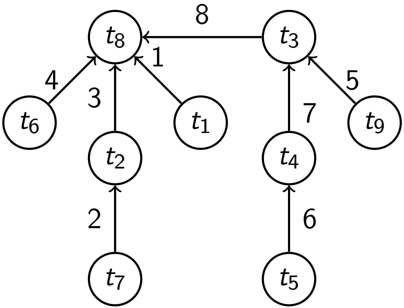
For improved algorithm: R. Nieuwenhuis and A. Oliveras. Proof-producing congruence closure. RTA'05, LNCS 3467

Commentary: We may need to apply symmetry rule to get the equality in right order.

Example: union-find proof generation

Consider:

$$\underbrace{t_1 \approx t_8}_{1} \wedge \underbrace{t_7 \approx t_2}_{2} \wedge \underbrace{t_7 \approx t_1}_{3} \wedge \underbrace{t_6 \approx t_7}_{4} \wedge \underbrace{t_9 \approx t_3}_{5} \wedge \underbrace{t_5 \approx t_4}_{6} \wedge \underbrace{t_4 \approx t_3}_{7} \wedge \underbrace{t_7 \approx t_5}_{8} \wedge \underbrace{t_1 \not\approx t_4}_{9}$$



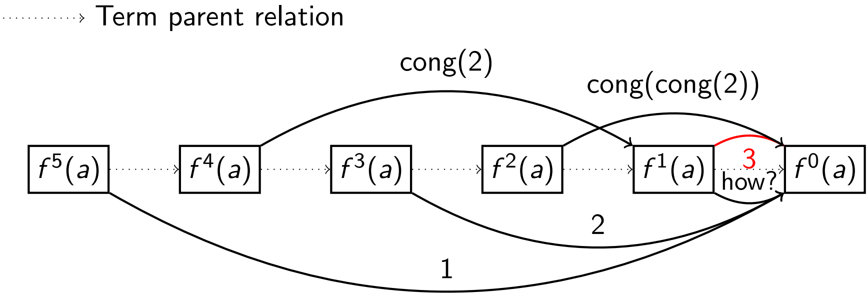
$$\dfrac{t_1 \not\approx t_4 \qquad \dfrac{\dfrac{t_7 \approx t_1}{t_1 \approx t_7} \qquad t_7 \approx t_5 \qquad t_5 \approx t_4}{t_1 \approx t_4}}{\perp}$$

1. $t_1 \not\approx t_4$ is violated.
2. 8 is the latest edge in the path between t_1 and t_4
3. 8 is due to $t_7 \approx t_5$
4. Look for proof of $t_1 \approx t_7$ and $t_5 \approx t_4$
5. 3 is the latest edge between t_1 and t_7, which is due to $t_7 \approx t_1$.
6. Similarly, $t_5 \approx t_4$ is edge 6

Example: extending to congruence

Example 8.8

Run union find on $\underbrace{f^5(a) \approx a}_{1} \land \underbrace{f^3(a) \approx a}_{2} \land \underbrace{f(a) \not\approx a}_{3}$



Extract proof from the above graph?

Union-find in the context of SMT solver

SMT solver design causes frequent calls to getClass(), which is not constant time.

To make it constant time, we may add another field in each node that points to the root.

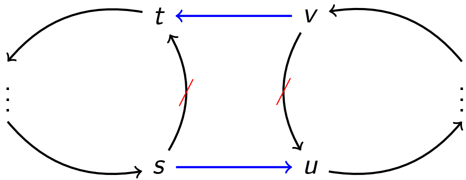
- Increases the cost of merge: needs to update the root field in each node
- Traversal in the tree needs a stack

Why not use a simpler data structure?

Union-find using circular linked lists

We may represent the equivalence class using circular linked lists and each node has a field to indicate the root, therefore getClass() is constant time

- merging two circular linked lists via field *next*



$$s.next, v.next := v.next, s.next$$

Exercise 8.8

How to split circular linked lists at two given nodes?

Detlefs, D., Nelson, G., Saxe, J.B.: Simplify: a theorem prover for program checking. J. ACM 52 (2005) (p386-389,p420-433)

Merge/unmerge classes

- On class merge,
  - the two circular linked lists are merged and
  - the root fields in the smaller of the two are set to the root of the other.
  - the "looser" root of the smaller list is recorded in order for possible unmerge
- On backtracking, we iterate over the loosers record in the reverse order and unmerge
  1. Let node x be the current top looser root.
  2. r := getClass(x); $r.next, x.next := x.next, r.next$.
  3. make x root of the part that contains x.

Example: merge/demerge classes

Data structures for congruence

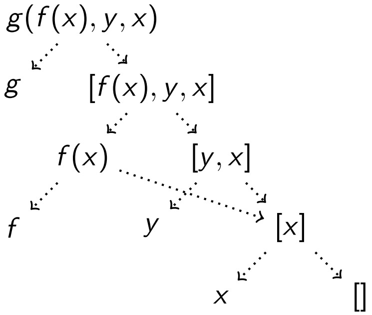
Terms as binary DAGs

- Term has two children: the top symbol and argument list
- Argument list has two children: the first term and tail list

We compute equivalence of terms as well as term lists.

The left child is called "car" and the right child is called "cdr".

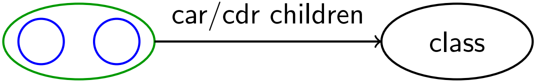
Example 8.9



Data structures for congruence

We add three fields in nodes to maintain equivalence class of nodes whose

1. car children are equivalent — car parent classes
2. cdr children are equivalent — cdr parent classes
3. both children are equivalent — congruent classes



Green: car/cdr children are in same class
Blue: both children in the same class

Exercise 8.9

Prove: each class consists of nodes that are either car children or cdr children.

Commentary: Car/Cdr sets are again maintained as circular linked lists. Similarly (un)merged but trigger by (un)merger of their car/cdr children. The looser root needs to keep sufficient information for unmerge. 3rd class is stored as the earlier union-find data structure.

Data Structures for congruence II

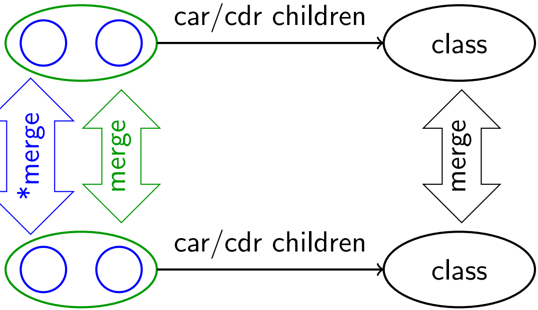
We also maintain a hash map containing $(x.car.root, x.cdr.root) \mapsto x$ entries for the roots of the congruent classes

The hash map allows to quickly identify which two congruent classes are ready to be merged.

Applying congruence upon merger

Consider two classes are being merged.

- In the smaller car/cdr parent class, iterate over the roots of the congruent classes
  - Check if they can be merged with the congruent classes in the other parent class using the hash map.
  - If two congruent classes merge, it triggers a new merge of classes



*conditional merge of congruent classes

Data structure for disequalities

For each equivalence class, we maintain a set of the other unmergable classes

- the set cannot be maintained as a circular linked lists over nodes by adding new field
- The set is maintained "exogenously", i.e., extra nodes allocated

Exercise 8.10

If we have input that says some $n > 2$ terms are distinct,

```
(distinct t1 ... tn)
```

How many entries we need in the unmergable classes lists?
Can we do it better?

Example: congruence data structure

Example 8.10

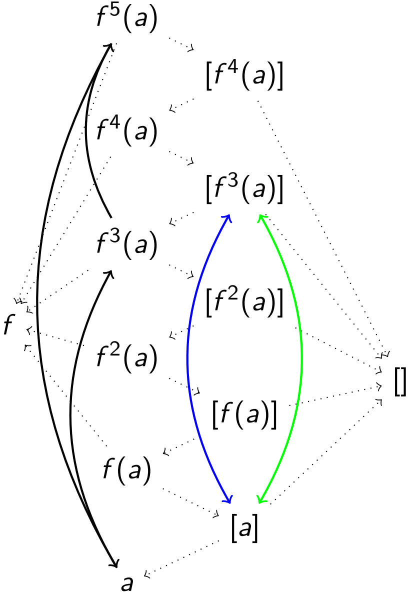
Consider

$\underbrace{f^5(a) \approx a}_{1} \wedge$

$\underbrace{f^3(a) \approx a}_{2} \wedge$

$\underbrace{f(a) \not\approx a}_{3}$

$class = \{f^5(a), f^3(a), a\}$



Topic 8.5

Problems

Problem

Exercise 8.11 (1.5 points)

Prove/Disprove that the following formula is unsat.

$$(f^4(a) \approx a \vee f^6(a) \approx a) \wedge f^3(a) \approx a \wedge f(a) \not\approx a$$

If unsat give a proof otherwise give a satisfying assignment.

Please show a run of DPLL($\mathcal{T}$) and union-find on the above example.

End of Lecture 8