Satisfiability Modulo Theories
Lecture 7 - A Theory Solver for $\mathcal{UF}$

(slides revision: Saturday 14th March, 2015, 11:47)

Roberto Bruttomesso

Seminario di Logica Matematica
(Corso Prof. Silvio Ghilardi)

1 Dicembre 2011

Outline

Introduction

Recall from the first lecture:

In SMT a theory $\mathcal{T}$ is defined over a **signature** Σ, which is a set of function and predicate symbols. The equality symbol $=$ is assumed to be included in every signature.

The signature of $\mathcal{UF}$ can include infinite functional and predicate symbols $\Sigma = \{f, g, h, \ldots, p, q, \ldots\}$, which can be used as usual to build $\mathcal{T}$-terms using variables

Examples:

$$x = f(g(y), z) \qquad p(x, g(x)) \qquad g(y) \neq g(z)$$

$\mathcal{UF}$ is the so-called **empty** theory as it does not contain any "explicit" axiom

Introduction

Still, there is equality $=$. So the following logic rules must be respected by any satisfiable $\mathcal{UF}$-formula[1]

$$\frac{}{s = s}\ \text{(refl.)} \qquad \frac{s = t}{t = s}\ \text{(symm.)} \qquad \frac{s = t \wedge t = r}{s = r}\ \text{(tran.)}$$

for all $\mathcal{UF}$-terms s, t, r

Moreover, there is a further condition that has to hold

$$\frac{s_1 = t_1 \wedge \ldots \wedge s_n = t_n}{f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)}\ \text{(cong.)}$$

[1] Notice that for $\mathcal{IDL}$ and $\mathcal{LRA}$ this was implicitly true.

Handling equalities: Union Find

Let's focus first on conjunctions of positive equalities between variables, (functions, congruence, and negated equalities are not considered for the moment)

Union-Find algorithms (Tarjan). It is based on the notion of **equivalence classes**. Equivalence classes form a partition of the set of variables V, i.e.,

- each partition is non empty
- each partition is disjoint
- the union of the partitions is V

Let φ^+ be a conjunction of positive equalities: Union-Find will find the partition of V such that:

$$x,\ y \text{ are in the same partition iff } \varphi^+ \Rightarrow x = y$$

Union-Find

Input: a conjunction of positive equalities φ^+
(e.g., $\varphi^+ \equiv\ x{=}y\ \wedge\ w{=}a\ \wedge\ w{=}b$, $V \equiv \{x, y, w, z, a, b, c\}$)
Initialization: one equivalence class per each variable in V

$$\{\ \mathbf{x}\ \} \quad \{\ \mathbf{y}\ \} \quad \{\ \mathbf{z}\ \} \quad \{\ \mathbf{w}\ \} \quad \{\ \mathbf{a}\ \} \quad \{\ \mathbf{b}\ \} \quad \{\ \mathbf{c}\ \}$$

For each equality $s = t$ in φ^+, merge the class containing s with that containing t (e.g., $x = y$)

$$\{\ \mathbf{x}, y\ \} \quad \{\ \mathbf{z}\ \} \quad \{\ \mathbf{w}\ \} \quad \{\ \mathbf{a}\ \} \quad \{\ \mathbf{b}\ \} \quad \{\ \mathbf{c}\ \}$$

The final situation is the desired partition

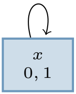
$$\{\ \mathbf{x}, y\ \} \quad \{\ \mathbf{z}\ \} \quad \{\ w, \mathbf{a}, b\ \} \quad \{\ \mathbf{c}\ \}$$

Notice that $\varphi^+ \Rightarrow s = t$ iff s, t in the same partition
A variable per class is nominated **representant**

Implementing Union-Find

We assume each variable x is a `Node * x`

```
struct Node {
  int    size;     // Keep track of a class size
  int    rank;     // Keep track of a class rank
  Node * root;     // Points to class' representant
};
```



initialized with size = 1, rank = 0, and root = x

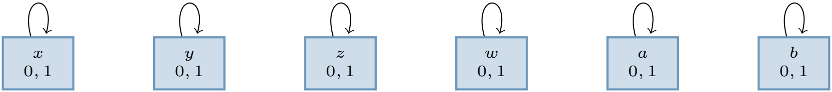
```
procedure Union( x, y )
assert( x == x.root && y == y.root )
if ( x == y ) return
if ( x.rank < y.rank )
    x.root = y
    y.size = y.size + x.size
else if ( x.rank > y.rank )
    y.root = x
    x.size = x.size + y.size
else
    x.root = y
    x.size = x.size + y.size
    x.rank = x.rank +1
```

```
procedure Find( x )
r = x
if ( r ≠ r.root )
    r = Find( r.root )
return r
```

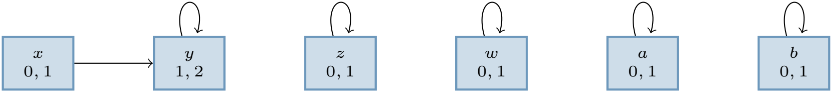
```
procedure Union-Find( φ+ )
foreach ( x = y ∈ φ+ )
    x = Find( x )
    y = Find( y )
    Union( x, y )
```

Example

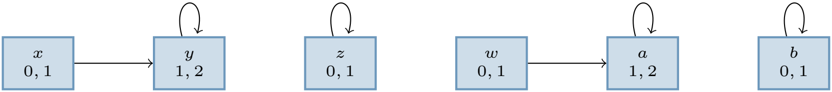
$\varphi^+ \equiv\ x=y\ \wedge\ w=a\ \wedge\ w=b$



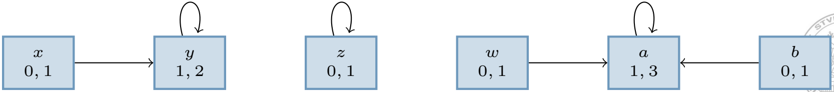
Processing $x = y$. State after Union(Find(x), Find(y))



Processing $w = a$. State after Union(Find(w), Find(a))



Processing $w = b$. State after Union(Find(w), Find(b))



Complexity

If we have n input variables

Union complexity: $O(1)$
Find complexity: $O(\log n)$

The complexity of Find is linear in the rank (height) of the trees. However because Union does not increase rank unless necessary, trees are always **balanced**. The following invariant holds

Invariant

For each representant x, $2^{x.rank} \leq x.size$

Worst case $\quad x.size = n \quad$ and so $\quad x.rank = \log_2 n$

If m equalities n variables, worst case of $O(m \log n)$

There is an improvement for Find, called **path compression**, that decrease the bound to $O(m\alpha(m,n))$, where α is the inverse of Ackermann's function ($\alpha(m,n) \leq 4$ in practice)

Quick-Find approach

```
struct Node {
  int    size;     // Keep track of a class size
  Node * next;     // Next element in eq. class (circular list)
  Node * root;     // Points to class' representant
};
```

initialized as size $= 0$, next $= x$, root $= x$

```
procedure Union( x, y )
assert( x == x.root && y == y.root )
if ( x == y ) return
if ( x.size > y.size )
    SWAP( x, y )
s = x.next
while ( s ≠ x )
    s.root = y
    s = s.next
SPLICE( x, y )
y.size = y.size + x.size
```

```
procedure Find( x )
return x.root
```

```
procedure Union-Find( φ+ )
foreach ( x = y ∈ φ+ )
    x = Find( x )
    y = Find( y )
    Union( x, y )
```

Union $O(n)$, Find $O(1)$, Union-Find $O(n \log n)$

Handling Functions

Let's consider also function symbols. We want to extend Union-Find to handle functions as well. For instance

$$x = y \ \wedge \ f(x) = z \ \wedge \ f(y) = w$$

should result into

$$\{ \ x, \ y \ \} \qquad\qquad \{ \ f(x), \ f(y), \ z, \ w \ \}$$

We don't see the details, but just a generic algorithm. Let φ^+ be a conjunction of positive equalities between $\mathcal{UF}$-terms

procedure Congruence-Closure(φ^+)
$\quad$ *pending* $= \varphi^+$
$\quad$ while (*pending* not empty)
$\qquad$ $(s = t) =$ *pending*.pop( )
$\qquad$ $s =$ Find(s)
$\qquad$ $t =$ Find(t)
$\qquad$ Union(s, t)
$\qquad$ for each pair p, q such that

1. Find(p) $\not\equiv$ Find(q)
2. $p \equiv f(s_1, \ldots, s_n)$, $\ q \equiv f(t_1, \ldots, t_n)$
3. Find(s_1)=Find(t_1),...,Find(s_n)=Find(t_n)

$\qquad\quad$ *pending*.push($p = q$)

After Congruence-Closure we have that

$$s, \ t \text{ are in the same partition iff } \varphi^+ \Rightarrow s = t$$

Example

$\varphi^+ \equiv f(a,b) = a \;\wedge\; f(f(a,b),b) = c$

$pending = \{ \quad f(a,b) = a, \quad f(f(a,b),b) = c \quad \}$

$$\{\ a\ \} \qquad \{\ b\ \} \qquad \{\ c\ \} \qquad \{\ f(a,b)\ \} \qquad \{\ f(f(a,b),b)\ \}$$

$pending = \{ \quad f(f(a,b),b) = c \quad f(f(a,b),b) = f(a,b) \quad \}$

$$\{\ a, f(a,b)\ \} \qquad \{\ b\ \} \qquad \{\ c\ \} \qquad \{\ f(f(a,b),b)\ \}$$

$pending = \{ \quad f(f(a,b),b) = f(a,b) \quad \}$

$$\{\ a, f(a,b)\ \} \qquad \{\ b\ \} \qquad \{\ f(f(a,b),b), c\ \}$$

$pending = \{\}$

$$\{\ a, f(a,b), f(f(a,b),b), c\ \} \qquad \{\ b\ \}$$

Solving

Finally we are ready to check satisfiability of φ. As before $\varphi \equiv \varphi^+ \wedge \varphi^-$. Negated equalities can be checked after building equivalence classes

```
1   procedure Check( φ⁺, φ⁻ )
2   Congruence-Closure( φ⁺ )
3   for each s ≠ t in φ⁻
4       s' = Find( s )
5       t' = Find( t )
6       if ( s' ≡ t' )
7           return unsat
8   return sat
```

This is correct, because

- at line 3, s, t are in the same partition iff $\varphi^+ \Rightarrow s = t$
- at line 7, s, t are in the same partition, but $s \neq t$

$\mathcal{T}$-solver features

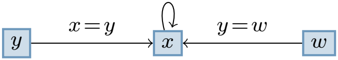
- Incrementality: for free, as Congruence-Closure is already incremental
- Backtrackability: backtracking one equality amounts to **undo** all the operations done during Congruence-Closure. (This means that backtracking is as expensive as solving)
- Minimal $\mathcal{T}$-conflicts: computing minimal conflicts in $\mathcal{UF}$ is theoretically very hard (NP-complete). We'll an acceptable way to compute them
- Theory-Propagation: it amounts to track all unassigned equalities such as $a = b$. If during some Union(x, y), a and b become equal (because for instance a is in the class of x and b is in the class of y), then propagate $a = b$

First of all notice that $\mathcal{T}$-conflicts are always of the form

$$\{\ s \neq t, \text{“other equalities that cause } s \text{ and } t \text{ to join the same class”}\}$$

We reconstruct the conflict by storing the reason that caused two classes to collapse. When a $s \neq t$ causes *unsat*, we call a routine Explain(s,t) that collects all the reasons that made s equal to t
Example

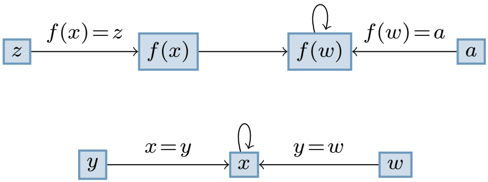
$$\{\ x = y,\ y = w,\ f(x) = z,\ f(w) = a,\ a \neq z\ \}$$



On processing $a \neq z$, we call Explain(a, z)

Computing $\mathcal{T}$-conflicts

$$\{\ x = y,\ y = w,\ f(x) = z,\ f(w) = a,\ a \neq z\ \}$$



If a and z are in the same class, it means that there are paths of the form

$$a \rightarrow^* u \qquad z \rightarrow^* u$$

(it could happen that $u \equiv a$ or $u \equiv z$)
Explain(s, t) intuitively works as follows:

1. Traverse $s \rightarrow^* u$ and collect all labels on edges
2. Traverse $t \rightarrow^* u$ and collect all labels on edges
3. If during collection an empty label was found
    1. It must be some $f(s_1, \ldots, s_n) \rightarrow f(t_1, \ldots, t_n)$
    2. call Explain(s_i, t_i) for $i \in [1..n]$
4. At the end of the recursions, the collected labels (without repetitions) are a $\mathcal{T}$-conflict

Layered Approach

Since $\mathcal{UF}$ is the empty theory, it is contained in any other theory $\mathcal{T}$. So the following implication holds

If a conjunction φ is $\mathcal{UF}$-unsatisfiable, then it is $\mathcal{T}$-unsatisfiable

Example:

$$x + y \neq z + w \quad \wedge \quad x = z \quad \wedge \quad y = w$$

is an $\mathcal{LRA}$-formula, but it is $\mathcal{UF}$-unsatisfiable

$$plus(x, y) \neq plus(z, w) \quad \wedge \quad x = z \quad \wedge \quad y = w$$

Therefore we can use $\mathcal{UF}$-solver as a layer for $\mathcal{LRA}$-solver on an input conjunction φ:

- Call $\mathcal{UF}$-solver on φ. If *unsat* return *unsat*
- Call $\mathcal{LRA}$-solver on φ

This can save many calls to $\mathcal{LRA}$-solver. Since $\mathcal{UF}$-solver is quick, this can save time !

Exercises

1. Prove by induction the invariant on slide 9
2. Write the precise pseudocode for Explain
3. Is the following conjunction satisfiable ?

$$f(g(x)) \neq f(y) \ \wedge \ x = w \ \wedge \ g(x) = h(z) \ \wedge \ z = x \ \wedge \ h(w) = y$$

4. Verify your anwser using an SMT-solver of your choice. In smtlib2 the above can be written as

```
(set-logic QF_UF)
(declare-sort U 0)
(declare-fun f ( U ) U)
...
(declare-fun x ( ) U)
(assert (not (= (f (g x)) (f y))))
...
(check-sat)
```