Program verification

Nelma Moreira

Lecture 22
Equality Logic and Theory of Uninterpreted Functions

Equality Logic and Uninterpreted Functions, $\mathcal{EUF}$

- functional terms are added to the equality theory

$$\varphi^{uf} := \varphi^{uf} \wedge \varphi^{uf} \mid \neg\varphi^{uf} \mid t = t$$
$$t := x|c|F(t_1, \ldots, t_n)$$

- only functional congruence

$$x_1 = y_1 \wedge \cdots \wedge x_n = y_n \rightarrow F(x_1, \ldots, x_n) = F(y_1, \ldots, y_n)$$

- Functions of a given theory can be substituted by uninterpreted functions simplifying the validity proofs although equivalence is not preserved. We have
$$\models \varphi^{uf} \implies \models \varphi$$

- but if $\not\models \varphi^{uf}$ nothing can be concluded.

Example: Program equivalence

```
int power3(int in) {
    out = in;

    for(i=0; i<2; i++)
       out = out * in;

    return out;
}
```

```
int power3_new(int in) {
    out = (in*in)*in;
    return out;
}
```

Static single assignment (SSA) form:

$$out_1 = in \land$$
$$out_2 = out_1 * in \land$$
$$out_3 = out_2 * in$$

$$out_1' = (in * in) * in$$

Prove that both functions return the same value:

$$out_3 = out_1'$$

Static single assignment

1. Remove the variable declarations and `return` statements.
2. Unroll the `for` loop.
3. Replace the left-hand side variable in each assignment with a new auxiliary variables
4. Wherever a variable is read (referred to in an expression), replace it with the auxiliary variable that replaced it in the last place where it was assigned.
5. Conjoin all program statements.

In the example, given two programs we obtain two formulae φ_1 and φ_1' and we want to prove that

$$\varphi_1 \land \varphi_1' \implies out_3 = out_1'$$

Usage of uninterpreted functions

Static single assignment (SSA) form: $out_1 = in \wedge$ $out_2 = out_1 * in \wedge$ $out_3 = out_2 * in$	$out'_1 = (in * in) * in$
With uninterpreted functions: $out_1 = in \wedge$ $out_2 = F(out_1, in) \wedge$ $out_3 = F(out_2, in)$	$out'_1 = F(F(in, in), in)$

The advantage is that it is easy to prove the validity of uninterpreted functions. In the example multiplication $*$ is substituted by an uninterpreted function F and we obtain φ_1^{uf} and $\varphi_1'^{uf}$.

Decision procedures for conjunctions of equalities and with uninterpreted functions with congruence closure

Input: conjunction of literals φ^{uf}
Output: Satisfiable or Unsatisfiable

1. Build congruence-closed equivalence classes.
   a) Initially, put two terms t_1, t_2 (either variables or uninterpreted function instances) in their own equivalence class if $(t_1 = t_2)$ is a predicate in φ^{uf}. All other variables form singleton equivalence classes.
   b) Given two equivalence classes with a shared term, merge them. Repeat until there are no more classes to be merged.
   c) Compute the congruence closure: given two terms t_i, t_j that are in the same class and that $F(t_i)$ and $F(t_j)$ are terms in φ^{uf} for some uninterpreted function F, merge the classes of $F(t_i)$ and $F(t_j)$. Repeat until there are no more such instances.
2. If there exists a disequality $t_i \neq t_j$ in φ^{uf} such that t_i and t_j are in the same equivalence class, return "Unsatisfiable". Otherwise return "Satisfiable"

Ex. 22.1. *Let φ^{uf} be a conjunction*

$$x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge x_5 \neq x_1 \wedge F(x_1) \neq F(x_3).$$

Initially the equivalence classes are:

$$\{x_1, x_2\}, \{x_2, x_3\}, \{x_4, x_5\}, \{F(x_3)\}, \{F(x_1)\}$$

We merge equal terms in the same classe

$$\{x_1, x_2, x_3\}, \{x_4, x_5\}, \{F(x_3)\}, \{F(x_1)\}$$

By the congruence closure we have:

$$\{x_1, x_2, x_3\}, \{x_4, x_5\}, \{F(x_1), F(x_3)\}$$

Finally as $F(x_1) \neq F(x_3) \in \varphi^{uf}$, *the output is* Unsatisfiable.

- This algorithm can be implemented efficiently with a union-find data structure, which results in a time complexity of $O(n \log(n))$.
- To extend to general quantifier-free formulae one can use DPLL(T) or (*lazy*) variants; or eager algorithms that reduces the whole formula φ^{uf} to a equisatisfiable propositional formula (*eager* approach).

Ackerman reduction of uninterpreted functions to equality logic

Suppose we want to check

$$x_1 \neq x_2 \vee F(x_1) = F(x_2) \vee F(x_1) \neq F(x_3)$$

for validity.

1. First number the function instances:
$$x_1 \neq x_2 \vee F_1(x_1) = F_2(x_2) \vee F_1(x_1) \neq F_3(x_3)$$
2. Replace each function with a new variable:
$$x_1 \neq x_2 \vee f_1 = f_2 \vee f_1 \neq f_3$$
3. Add functional consistency constraints:
$$\left(\begin{array}{ll} (x_1 = x_2 \rightarrow f_1 = f_2) & \wedge \\ (x_1 = x_3 \rightarrow f_1 = f_3) & \wedge \\ (x_2 = x_3 \rightarrow f_2 = f_3) & \end{array} \right) \rightarrow$$
$$((x_1 \neq x_2) \vee (f_1 = f_2) \vee (f_1 \neq f_3))$$

Ackerman reduction

Input:	An $\mathcal{EUF}$ formula φ^{uf} with m instances of an uninterpreted function F
Output:	An equality logic formula φ^E such that φ^E is valid if and only if φ^{uf} is valid

1. Assign indices to the uninterpreted-function instances from subexpressions F, F_i outwards. Denote by F_i the instance of F that is given the index i, and by $arg(F_i)$ its single argument.
2. Let $flat^E = \mathcal{T}(\varphi^{uf})$, where $\mathcal{T}$ is a function that takes an $\mathcal{EUF}$ formula (or term) as input and transforms it to an equality formula (or term, respectively) by replacing each uninterpreted-function instance F_i with a new term-variable f_i (in the case of nested functions, only the variable corresponding to the most external instance remains).

3. Let FC^E denote the following conjunction of functional-consistency constraints:
$$FC^E := \bigwedge_{i=1}^{m-1} \bigwedge_{j=i+1}^{m} \mathcal{T}(arg(F_i)) = \mathcal{T}(arg(F_j)) \implies f_i = f_j$$
4. Let $\varphi^E := FC^E \implies flat^E$
5. Return φ^E

Example

Let φ be
$$x_1 = x_2 \implies F(F(G(x_1)))) = F(F(G(x_2)))$$

Consider the propositional variables g_1, g_2, f_1, f_2, f_3, and f_4

$$x_1 = x_2 \implies \underbrace{F(\underbrace{F(\overbrace{G(x_1)}^{g_1})}_{f_1})}_{f_2} = \underbrace{F(\underbrace{F(\overbrace{G(x_2)}^{g_2})}_{f_3})}_{f_4}$$

then $flat^E : x_1 = x_2 \implies f_2 = f_4$ and FC^E is

$$\begin{aligned}
x_1 = x_2 &\implies g_1 = g_2 \\
g_1 = f_1 &\implies f_1 = f_2 \\
g_2 = f_3 &\implies f_3 = f_4 \\
g_1 = g_2 &\implies f_1 = f_3 \\
g_1 = f_3 &\implies f_1 = f_4 \\
f_1 = g_2 &\implies f_2 = f_3 \\
f_1 = f_3 &\implies f_2 = f_4 \\
g_2 = f_3 &\implies f_1 = f_4
\end{aligned}$$

Thus, we have $\varphi^E = FC^E \implies flat^E$.

Eager procedures for equational logic

We will see how to construct a formula of propositional logic that is equisatisfiable to a formula of equational logic without quantifiers.

The SAT solver is called only once.

The presented algorithm will not be very efficient but can be optimized in order to execute in polynomial time and obtain a propositional formula with a cubic size in the number of variables of the equational formula.

Sets of literals of equalities and inequalities

Assume a equational formula φ^E (without constants) with Boolean operations in NNF.

- Let $E_=$ be the set of positive literals in φ^E
- Let $E_{\neq}$ be the set of negative literals in φ^E

For example φ^E

$$\begin{array}{c}(x_1 \neq x_2 \vee y_1 \neq y_2 \vee f_1 = f_2) \wedge \\ (u_1 \neq f_1 \vee u_2 \neq f_2 \vee g_1 = g_2) \wedge \\ (u_1 = f_1 \vee u_2 = f_2 \vee z = g_1) \wedge z \neq g_2\end{array}$$

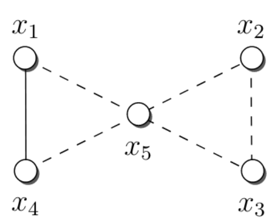
We have

$$\begin{array}{rcl}E_= & = & \{f_1 = f_2, g_1 = g_2, u_1 = f_1, u_2 = f_2, z = g_1\} \\ E_{\neq} & = & \{x_1 \neq x_2, y_1 \neq y_2, u_1 \neq f_1, u_2 \neq f_2, z \neq g_2\}\end{array}$$

Equality graph

Given a equality logic formula φ^E in NNF, the *equality graph* of φ^E, $G^E(\varphi^E)$ is the graph $(V, E_=, E_{\neq})$ where the nodes V are the variables in φ^E, the edges $E_=$ correspond to the set of positive literals and the edges $E_{\neq}$ to the set of negative literals.

For example, for $E_= = \{x_1 = x_5, x_2 = x_3, x_2 = x_5, x_4 = x_5\}$ e $E_{\neq} = \{x_1 \neq x_4\}$ we have



As in the case of conjunctions of literals, graphically we represent with a dashed line the edges that correspond to equalities and solid those of inequalities.

- The equational graph $G^E(\varphi^E)$ is an abstraction of φ^E

- It actually represents all formulas that have the same literals as φ^E
- Since it does not consider Boolean connectives, it can represent both satisfiable and unsatisfiable formulas
- For example $x_1 = x_2 \wedge x_1 \neq x_2$ and $x_1 = x_2 \vee x_1 \neq x_2$ are represented by the same graph.

Equality and Disequality Paths

- A *equality path* in G^E is a path with only edges of $E_=$. If there is such a path between x and y we say that $x =^* y$, for $x, y \in V$.
- A *disequality path* in G^E is a path with edges of $E_=$ and only one edge og $E_{\neq}$. If there is such a path between $x$and y we write $x \neq^* y$ $x, y \in V$.
- Any of these paths is simple if has no cycles.
- If $x =^* y$ it can happen that x and y have the same value but is not necessary (because we do not have the structure of the Boolean formula).
- For $x \neq^* y$ it can happen that x and y have different values
- in the example $x_1 =^* x_4$ and $x_1 \neq^* x_4$ but that may not be inconsistent
- A *contradictory cycle* is a cycle in G^E that has exactly one edge in $E_{\neq}$
- For $x, y \in V$ ia a contradictory cycle we have $x =^* y$ and $x \neq^* y$.
- The conjunction of literals of the cycle is unsatisfiable.
- In the example x_1, x_2, x_4 is a contradictory cycle

We can simplify formulas if there are literals that do not participate in contradictory cycles (simple).

Simplifications of the Formula

Algorithm 11.4.1: SIMPLIFY-EQUALITY-FORMULA

Input: An equality formula φ^{E}
Output: An equality formula $\varphi^{\mathrm{E}\prime}$ equisatisfiable with φ^{E}, with size less than or equal to the length of φ^{E}

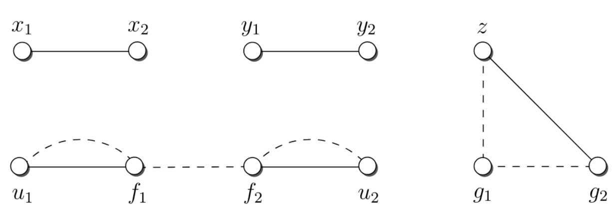
1. Let $\varphi^{\mathrm{E}\prime} := \varphi^{\mathrm{E}}$.
2. Construct the equality graph $G^{\mathrm{E}}(\varphi^{\mathrm{E}\prime})$.
3. Replace each pure literal in $\varphi^{\mathrm{E}\prime}$ whose corresponding edge is not part of a simple contradictory cycle with TRUE.
4. Simplify $\varphi^{\mathrm{E}\prime}$ with respect to the Boolean constants TRUE and FALSE (e.g., replace TRUE $\vee \phi$ with TRUE, and FALSE $\wedge \phi$ with FALSE).
5. If any rewriting has occurred in the previous two steps, go to step 2.
6. Return $\varphi^{\mathrm{E}\prime}$.

Example

Let

$$\varphi^E := (x_1 \neq x_2 \vee y_1 \neq y_2 \vee f_1 = f_2) \wedge (u_1 \neq f_1 \vee u_2 \neq f_2 \vee g_1 = g_2) \wedge (u_1 = f_1 \vee u_2 = f_2 \vee z = g_1) \wedge z \neq g_2$$

the graph G^E is



The edges $f_1 = f_2$, $x_1 \neq x_2$ and $y_1 \neq y_2$ are not part of any simple contradictory cycle and can therefore be substituted by true.

$$\varphi'^E := (\mathsf{true} \vee \mathsf{true} \vee \mathsf{true}) \wedge (u_1 \neq f_1 \vee u_2 \neq f_2 \vee g_1 = g_2) \wedge (u_1 = f_1 \vee u_2 = f_2 \vee z = g_1) \wedge z \neq g_2$$

Simplifying

$$\varphi'^E := (u_1 \neq f_1 \vee u_2 \neq f_2 \vee g_1 = g_2) \wedge (u_1 = f_1 \vee u_2 = f_2 \vee z = g_1) \wedge z \neq g_2$$

And in this case, if we calculate the graph, we see that we can not simplify any further. However, if the contradictory cycles disappear, we can conclude that the formula is satisfiable (and only by simplifying).

Reduction to propositional Logic (*sparse method*, Bryant et al)

Nonpolar equality graph

Let φ^E be a equational formula, a *nonpolar equality graph* of φ^E, $G^E_{NP}(\varphi^E)$ is a graph (V, E) where V are the variables of φ^E and the edges E correspond to $At(\varphi^E)$, i.e., all atomic formulae (equalities) φ^E.

- Note that $x_1 \neq x_2$ is an abbreviation of $\neg x_1 = x_2$, then G^E_{NP} only $x_1 = x_2$ is present in E.
- Instead of literals we only consider equalities (omitting the polarity).

Transformation to propositional logic

Given φ^E the procedure generates two propositional formulas $e(\varphi^E)$ and $\mathcal{B}_{trans}$ such that

$$\varphi^E \text{ is satisfiable } \iff e(\varphi^E) \wedge \mathcal{B}_{trans} \text{ is satisfiable}$$

- The formula $e(\varphi^E)$ is the *propositional skeleton* of φ^E, where every predicate $x_i = x_j$ ($i \leq j$) is replaced with a new Boolean variable $e_{i,j}$
- The formula $\mathcal{B}_{trans}$ is a conjunction of implications, the *transitive constraints*. Each such implication is associated with a cycle in the nonpolar equality graph G^E_{NP}.
- For a cycle with n edges $\mathcal{B}_{trans}$ forbids an assignment false to one of the edges when all the other edges are assigned true.

Correctness

- If φ^E is satisfiable $e(\varphi^E)$ is also satisfiable
- The constraints $\mathcal{B}_{trans}$ are enough to ensure that φ^E is satisfiable if $e(\varphi^E)$ is.

Let $\varphi^E := x_1 = x_2 \wedge (((x_2 = x_3) \wedge (x_1 \neq x_3)) \vee (x_1 \neq x_2))$ then

$$e(\varphi^E) := e_{1,2} \wedge (((e_{2,3} \wedge (\neg e_{1,3})) \vee (\neg e_{1,2}))$$

The formulae $x_1 = x_2$, $x_2 = x_3$ and $x_1 = x_3$ form a cycle in G^E_{NP} then the transitive constraints are:

$$\begin{aligned}\mathcal{B}_{trans} :=(&(e_{1,2} \wedge e_{2,3}) \implies e_{1,3}) \wedge \\ &(e_{1,2} \wedge e_{1,3}) \implies e_{2,3}) \wedge \\ &(e_{2,3} \wedge e_{1,3}) \implies e_{1,2}).\end{aligned}$$

A nonch

Algorithm 11.5.1: EQUALITY-LOGIC-TO-PROPOSITIONAL-LOGIC

Input: An equality formula φ^{E}
Output: A propositional formula equisatisfiable with φ^{E}

1. Construct a Boolean formula $e(\varphi^{\mathrm{E}})$ by replacing each atom of the form $x_i = x_j$ in φ^{E} with a Boolean variable $e_{i,j}$.
2. Construct the nonpolar equality graph $G^{\mathrm{E}}_{\mathrm{NP}}(\varphi^{\mathrm{E}})$.
3. Make $G^{\mathrm{E}}_{\mathrm{NP}}(\varphi^{\mathrm{E}})$ chordal.
4. $\mathcal{B}_{trans} :=$ TRUE.
5. For each triangle $(e_{i,j}, e_{j,k}, e_{i,k})$ in $G^{\mathrm{E}}_{\mathrm{NP}}(\varphi^{\mathrm{E}})$,

$$\begin{aligned}\mathcal{B}_{trans} := \mathcal{B}_{trans} \wedge & \\ &(e_{i,j} \wedge e_{j,k} \implies e_{i,k}) \wedge \\ &(e_{i,j} \wedge e_{i,k} \implies e_{j,k}) \wedge \\ &(e_{i,k} \wedge e_{j,k} \implies e_{i,j}) \ .\end{aligned} \tag{11.42}$$

6. Return $e(\varphi^{\mathrm{E}}) \wedge \mathcal{B}_{trans}$.

Complexity and Optimization

- The algorithm can have exponential complexity because the number of cycles in a graph can be exponential

- A *chord* in a cycle is any edge that connects two nonadjacent vertices in a cycle

- Bryant et al shown that

  > It is sufficient to add transitive constraints for simple chord-free cycles

- *Chordal graphs* A chordal graph is an undirected graph in which no cycle of size 4 or more is chord-free.

- Every graph can be made chordal in a time polynomial in the number of vertices

- Since the only chord-free cycles in a chordal graph are triangles, this implies that applying the procedure to these graphs can be done in polynomial time and obtain a formula whose size is not more than cubic in the number of variables (3 constraints for each triangle). The newly added chords are represented by new variables that appear in $\mathcal{B}_{trans}$ but not in $e(\varphi^E)$.

Algorithm 11.5.1: EQUALITY-LOGIC-TO-PROPOSITIONAL-LOGIC

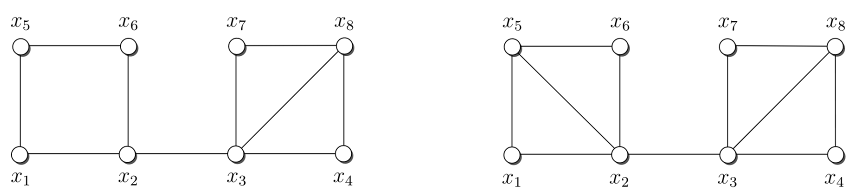
Input: An equality formula φ^{E}
Output: A propositional formula equisatisfiable with φ^{E}

1. Construct a Boolean formula $e(\varphi^{\mathrm{E}})$ by replacing each atom of the form $x_i = x_j$ in φ^{E} with a Boolean variable $e_{i,j}$.
2. Construct the nonpolar equality graph $G^{\mathrm{E}}_{\mathrm{NP}}(\varphi^{\mathrm{E}})$.
3. Make $G^{\mathrm{E}}_{\mathrm{NP}}(\varphi^{\mathrm{E}})$ chordal.
4. $\mathcal{B}_{trans} := \text{TRUE}$.
5. For each triangle $(e_{i,j}, e_{j,k}, e_{i,k})$ in $G^{\mathrm{E}}_{\mathrm{NP}}(\varphi^{\mathrm{E}})$,

$$\begin{aligned}\mathcal{B}_{trans} := \mathcal{B}_{trans} \wedge & \\ &(e_{i,j} \wedge e_{j,k} \implies e_{i,k}) \wedge \\ &(e_{i,j} \wedge e_{i,k} \implies e_{j,k}) \wedge \\ &(e_{i,k} \wedge e_{j,k} \implies e_{i,j})\ .\end{aligned} \tag{11.42}$$

6. Return $e(\varphi^{\mathrm{E}}) \wedge \mathcal{B}_{trans}$.

A nonchordal nonpolar equality graph corresponding to φ^E and a possible chordal version of it (right).



For the triangle (x_1, x_2, x_5),

$$\begin{aligned} e_{1,2} \wedge e_{2,5} &\implies e_{1,5}\ , \\ e_{1,5} \wedge e_{2,5} &\implies e_{1,2}\ , \\ e_{1,2} \wedge e_{1,5} &\implies e_{2,5}\ . \end{aligned}$$

References

[BdM15] Nikolai Bjorner and Leonardo de Moura. *Z3 Theorem Prover*. Rise, Microsft, 2015.

[BM07] Aaron R. Bradley and Zohar Manna. *The Calculus of Computation: Decision Procedures with Applications to Verification*. Springer Verlag, 2007.

[KS16] Daniel Kroening and Ofer Strichman. *Decision Procedures:An Algorithmic Point of View*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016.