Decision Procedures and Verification

Martin Blicha

Charles University

9.4.2018

Equality Logic and Uninterpreted Functions

Equality logic

- Equality logic $\approx$ Theory of equality
- As propositional logic where the atoms are equalities between variables over some infinite type.
  - Or between variables and constants

Definition

An equality logic formula is defined by the following grammar:

$$\begin{aligned} \mathit{fla} &: \mathit{fla} \wedge \mathit{fla} \mid \mathit{fla} \vee \mathit{fla} \mid \neg \mathit{fla} \mid \mathit{atom} \\ \mathit{atom} &: \mathit{term} = \mathit{term} \\ \mathit{term} &: \mathit{identifier} \mid \mathit{constant} \end{aligned}$$

where identifiers are variables defined over single ininite domain and constants are elements of the same domain as identiiers.

Complexity and expressiveness

Complexity of satisfiability

A satisfiability problem in equality logic is NP-complete.

- More natural modelling (high level structure preserved)
- More efficient (special decision procedures using the high level structure)

Removal of constants

For an equality logic formula φ^E, an equisatisfiable equality logic formula ψ^E without constants can be constructed in linear time.

- Replace constants with fresh variables.
- Add inequalities between these variables.

Adding functions

- Motivation: ability to model more than just the equality without adding too much complexity.

Definition (Equality logic with uninterpreted functions)

Let *Var* be a set of variables and *Fun* be a set of function symbols with arities. Equality logic formula with uninterpreted functions is given by the following grammar:

$$\begin{aligned} fla &: fla \wedge fla \mid fla \vee fla \mid \neg fla \mid atom \\ atom &: term = term \\ term &: var \mid f(term, \ldots, term) \end{aligned}$$

where $var \in Var$ and $f \in Fun$.

- Note: uninterpreted predicates are similar, but for simplicity we do not consider them here

Functional consistency

- Theory of uninterpreted functions includes axioms for functional consistency.
- Intuitively: "Instances of the same function return the same value if given equal arguments."
- For every function symbol $f \in \mathit{Fun}$ of arity $n > 0$ the following axiom is included:

$$\forall x_1, \dots, x_n, y_1, \dots, y_n$$
$$(x_1 = y_1 \wedge \dots \wedge x_n = y_n) \rightarrow (f(x_1, \dots, x_n) = f(y_1, \dots, y_n))$$

Benefits of uninterpreted function

- Formula φ with *interpreted* functions can be simplified to a formula φ^{UF} where each interpreted function is replaced by an uninterpreted one.
- Deciding validity of φ^{UF} can be much simpler than deciding validity of φ.

Observation

Let T be a theory with equality. For every formula φ it holds that if φ^{UF} is T-valid, then φ is T-valid.

- Validity with uninterpreted functions implies validity under *any* interpretation.
- The reverse implication does not hold.

Applications

- Verifying compiler optimization
  - Proving equivalence of programs
- Optimizing circuits
  - Proving equivalence of circuits

Application in program equivalence (1)

Original version

```
int power3(int in)
{
  int i, out_a;
  out_a = in;
  for (i = 0; i < 2; i++)
  {
    out_a = out_a * in;
  }
  return out_a;
}
```

Optimized version

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

Are these implementations equivalent?

Application in program equivalence (2)

- Encoding as formulas:
  - $\varphi_a := out0_a = in \land out1_a = out0_a * in \land out2_a = out1_a * in$
  - $\varphi_b := out0_b = (in * in) * in$
- Check validity of $\varphi_a \land \varphi_b \to out2_a = out0_b$

Application in program equivalence (2)

- Encoding as formulas:
  - $\varphi_a := out0_a = in \wedge out1_a = out0_a * in \wedge out2_a = out1_a * in$
  - $\varphi_b := out0_b = (in * in) * in$
- Check validity of $\varphi_a \wedge \varphi_b \rightarrow out2_a = out0_b$
- Replace multiplication by uninterpreted function G:
  - $\varphi_a^{UF} := out0_a = in \wedge out1_a = G(out0_a, in) \wedge out2_a = G(out1_a, in)$
  - $\varphi_b^{UF} := out0_b = G(G(in, in), in)$
- Check validity of $\varphi_a^{UF} \wedge \varphi_b^{UF} \rightarrow out2_a = out0_b$

Application in program equivalence (2)

- Encoding as formulas:
  - $\varphi_a := out0_a = in \wedge out1_a = out0_a * in \wedge out2_a = out1_a * in$
  - $\varphi_b := out0_b = (in * in) * in$
- Check validity of $\varphi_a \wedge \varphi_b \rightarrow out2_a = out0_b$
- Replace multiplication by uninterpreted function G:
  - $\varphi_a^{UF} := out0_a = in \wedge out1_a = G(out0_a, in) \wedge out2_a = G(out1_a, in)$
  - $\varphi_b^{UF} := out0_b = G(G(in, in), in)$
- Check validity of $\varphi_a^{UF} \wedge \varphi_b^{UF} \rightarrow out2_a = out0_b$
- If the formula is valid then the programs are equivalent.

Solving UF - Example

- $\varphi^{UF} := x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge x_5 \neq x_1 \wedge F(x_1) \neq F(x_3)$

Solving UF - Example

- $\varphi^{UF} := x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge x_5 \neq x_1 \wedge F(x_1) \neq F(x_3)$
- We can derive that $x_1 = x_3$

Solving UF - Example

- $\varphi^{UF} := x_1 = x_2 \land x_2 = x_3 \land x_4 = x_5 \land x_5 \neq x_1 \land F(x_1) \neq F(x_3)$
- We can derive that $x_1 = x_3$
- We can derive that $F(x_1) = F(x_3)$

Solving UF - Example

- $\varphi^{UF} := x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge x_5 \neq x_1 \wedge F(x_1) \neq F(x_3)$
- We can derive that $x_1 = x_3$
- We can derive that $F(x_1) = F(x_3)$
- We note the contradiction with $F(x_1) \neq F(x_3)$

Solving UF - Example

- $\varphi^{UF} := x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge x_5 \neq x_1 \wedge F(x_1) \neq F(x_3)$
- We can derive that $x_1 = x_3$
- We can derive that $F(x_1) = F(x_3)$
- We note the contradiction with $F(x_1) \neq F(x_3)$

The input formula is unsatisfiable!

Algorithm Cogruence closure

- Deciding conjunctive fragment of equality logic with uninterpreted functions
  - Here for single-argument functions

Algorithm Congruence Closure

---

1. Build congruence-closed equivalence classes.
   a) If $(t_1 = t_2) \in \varphi^{UF}$ then put all t_1 and t_2 to the same equivalence class. All other terms 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 $t_i \neq t_j \in \varphi^{UF}$ such that t_i and t_j are in the same equivalence class, return UNSAT. Otherwise return SAT.

Algorithm Cogruence closure - Notes

- Can be implemented efficiently with *union-find* data structure
- Resulting in $O(n \; log \; n)$ time complexity
- Can be used in $DPLL(T)$ procedure yielding a full decision procedure for UF
- Other approaches exists: reducing UF to equality logic and eventually to propositional logic.

Reducing UF to equality logic

- Ackermann's reduction
- Bryant's reduction

Ackermann's reduction

- A given φ^{UF} with uninterpreted functions is reduced to equality logic formula φ^{E} such that it is valid iff φ^{UF} is valid.
- Axioms of functional consistency need to be modeled within φ^{E} by auxiliary variables.
- φ^{E} is of the form $FC^{E} \Rightarrow \mathit{flat}^{E}$ where FC^{E} represents constraints for functional consistency and flat^{E} is φ^{UF} after replacement of functions with variables.

Ackermann's reduction

- Single uninterpreted function wiht single argument
- Input: An EUF formula φ^{UF}
- Output: An equality logic formula φ^{E} which is valid iff φ^{UF} is.

Algorithm ACKERMANN'S REDUCTION

1. Assign indices to the uninterpreted-function instances from subexpressions outwards. Let F_i denote the i-th instance of F and $arg(F_i)$ denote its single argument.
2. Let $flat^E := \tau(\varphi^{UF})$, where τ is a function that replaces each occurence of uninterpreted function F_i with new variable f_i.
3. Let FC^E denote the following conjunction of functional-consistency con- straints:
   $FC^E := \bigwedge_i \bigwedge_j (\tau(arg(F_i)) = \tau(arg(F_j))) \Rightarrow f_i = f_j$
4. Return $\varphi^E := FC^E \Rightarrow flat^E$.

Ackermann's reduction - example

- Consider
  $\varphi^{UF} := (x_1 \neq x_2) \vee (F(x_1) = F(x_2)) \vee (F(x_1) \neq F(x_3))$
- Number instances of F:
  - $F(x_1) \ldots f_1$
  - $F(x_2) \ldots f_2$
  - $F(x_3) \ldots f_3$
- Replace function instances and establish function consistency
  - $flat^E := (x_1 \neq x_2) \vee (f_1 = f_2) \vee (f_1 \neq f_3)$
  - $$\begin{aligned} FC^E := {} & (x_1 = x_2) \Rightarrow (f_1 = f_2) \wedge \\ & (x_1 = x_3) \Rightarrow (f_1 = f_3) \wedge \\ & (x_2 = x_3) \Rightarrow (f_2 = f_3) \end{aligned}$$

Ackermann's reduction - multiple functions and nesting

- Consider $\psi^{UF} := (x_1 = x_2) \Rightarrow F(F(G(x_1))) = F(F(G(x_2)))$
- Number instances:
  - $G(x_1) \ldots g_1$
  - $F(G(x_1)) \ldots f_1$
  - $F(F(G(x_1))) \ldots f_2$
  - $G(x_2) \ldots g_2$
  - $F(G(x_2)) \ldots f_3$
  - $F(F(G(x_2))) \ldots f_4$
- Replace function instances and establish function consistency
  - $flat^E := (x_1 = x_2) \Rightarrow (f_2 = f_4)$
  - $$\begin{aligned} FC^E := (x_1 = x_2) \Rightarrow (g_1 = g_2) \quad &\wedge \quad (g_1 = f_1) \Rightarrow (f_1 = f_2) \wedge \\ (g_1 = g_2) \Rightarrow (f_1 = f_3) \quad &\wedge \quad (g_1 = f_3) \Rightarrow (f_1 = f_4) \wedge \\ (f_1 = g_2) \Rightarrow (f_2 = f_3) \quad &\wedge \quad (f_1 = f_3) \Rightarrow (f_2 = f_4) \wedge \\ (g_2 = f_3) \Rightarrow (f_3 = f_4) \quad & \end{aligned}$$

Ackermann's reduction - validity and satiafiability

- Validity
  - Checking validity of φ^{UF} is reduced to checking validity of $\varphi^{E} := FC^{E} \Rightarrow \mathit{flat}^{E}$
  - Equivalently, unsatisfiability of $\neg\varphi^{E} := FC^{E} \wedge \neg \mathit{flat}^{E}$ can be checked.
- Satisfiability
  - Checking satisfiability of φ^{UF} is reduced to satisfiability of $\varphi^{E} := FC^{E} \wedge \mathit{flat}^{E}$
    - Equivalently, non-validity of $\neg\varphi^{UF}$
    - Ackermann's reduction of $\neg\varphi^{UF}$ yields $FC^{E} \Rightarrow \neg \mathit{flat}^{E}$ (same constraints for functional consistency).
    - Checking non-validity of $FC^{E} \Rightarrow \neg \mathit{flat}^{E}$ is the same as checking satisfiability of $FC^{E} \wedge \mathit{flat}^{E}$.

Reduction from equality logic to propositional logic

- Graph-based reduction to propositional logic:
  - Propositional skeleton + transitivity constraints.
  - Transitivity constraints ensure the transitivy of equality is captured at the propositional level.
- Domain allocation
  - Based on the small-model property that the equality logic has.

1. Determine a domain allocation
2. Encode each variable as an enumerated type over its finite domain. Construct a propositional formula representing the equality logic formula under this finite domain and use SAT to check if this formula is satisfiable.