Decision Procedures and Verification

Martin Blichá

Charles University

9.4.2018
Equality Logic and Uninterpreted Functions
Equality logic

- Equality logic ≈ 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:

\[
fla : fla \land fla \mid fla \lor fla \mid \neg fla \mid atom
\]

\[
atom : term = term
\]

\[
term : identifier \mid constant
\]

where identifiers are variables defined over single infinite domain and constants are elements of the same domain as identifiers.
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 $\varphi^E$, an equisatisfiable equality logic formula $\psi^E$ 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:

$$fla : fla \land fla \mid fla \lor fla \mid \neg fla \mid atom$$

$$atom : term = term$$

$$term : var \mid f(term, \ldots, term)$$

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 \( f \in \text{Fun} \) of arity \( n > 0 \) the following axiom is included:

\[
\forall x_1, \ldots, x_n, y_1, \ldots, y_n \quad (x_1 = y_1 \land \cdots \land x_n = y_n) \rightarrow (f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n))
\]
Benefits of uninterpreted function

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

**Observation**

Let $T$ be a theory with equality. For every formula $\varphi$ it holds that if $\varphi^{UF}$ is $T$-valid, then $\varphi$ 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
Are these implementations equivalent?
Application in program equivalence (2)

- Encoding as formulas:
  - $\varphi_a := \text{out}0_a = \text{in} \land \text{out}1_a = \text{out}0_a \ast \text{in} \land \text{out}2_a = \text{out}1_a \ast \text{in}$
  - $\varphi_b := \text{out}0_b = (\text{in} \ast \text{in}) \ast \text{in}$

- Check validity of $\varphi_a \land \varphi_b \rightarrow \text{out}2_a = \text{out}0_b$
Application in program equivalence (2)

- Encoding as formulas:
  - $\varphi_a := \text{out}0_a = \text{in} \land \text{out}1_a = \text{out}0_a \ast \text{in} \land \text{out}2_a = \text{out}1_a \ast \text{in}$
  - $\varphi_b := \text{out}0_b = (\text{in} \ast \text{in}) \ast \text{in}$

- Check validity of $\varphi_a \land \varphi_b \rightarrow \text{out}2_a = \text{out}0_b$

- Replace multiplication by uninterpreted function $G$:
  - $\varphi_a^{UF} := \text{out}0_a = \text{in} \land \text{out}1_a = G(\text{out}0_a, \text{in}) \land \text{out}2_a = G(\text{out}1_a, \text{in})$
  - $\varphi_b^{UF} := \text{out}0_b = G(G(\text{in}, \text{in}), \text{in})$

- Check validity of $\varphi_a^{UF} \land \varphi_b^{UF} \rightarrow \text{out}2_a = \text{out}0_b$

If the formula is valid then the programs are equivalent.
Application in program equivalence (2)

- Encoding as formulas:
  - \( \varphi_a := out_{0a} = in \land out_{1a} = out_{0a} \ast in \land out_{2a} = out_{1a} \ast in \)
  - \( \varphi_b := out_{0b} = (in \ast in) \ast in \)

- Check validity of \( \varphi_a \land \varphi_b \rightarrow out_{2a} = out_{0b} \)

- Replace multiplication by uninterpreted function \( G \):
  - \( \varphi^{UF}_a := out_{0a} = in \land out_{1a} = G(out_{0a}, in) \land out_{2a} = G(out_{1a}, in) \)
  - \( \varphi^{UF}_b := out_{0b} = G(G(in, in), in) \)

- Check validity of \( \varphi^{UF}_a \land \varphi^{UF}_b \rightarrow out_{2a} = out_{0b} \)

- If the formula is valid then the programs are equivalent.
Solving UF - Example

\[ \varphi^{UF} := x_1 = x_2 \land x_2 = x_3 \land x_4 = x_5 \land x_5 \neq x_1 \land F(x_1) \neq F(x_3) \]
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 \( x_1 = x_3 \)
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

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

- We can derive that $x_1 = x_3$
- We can derive that $F(x_1) = F(x_3)$
- We note the contradiction with $F(x_1) \neq F(x_3)$
Solving UF - Example

\begin{itemize}
  \item $\varphi^{UF} := x_1 = x_2 \land x_2 = x_3 \land x_4 = x_5 \land x_5 \neq x_1 \land F(x_1) \neq F(x_3)$
  \item We can derive that $x_1 = x_3$
  \item We can derive that $F(x_1) = F(x_3)$
  \item We note the contradiction with $F(x_1) \neq F(x_3)$
\end{itemize}

The input formula is unsatisfiable!
Algorithm **COGRUENCE CLOSURE**

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

---

**Algorithm COGRUENCE 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 \(\varphi^{UF}\) 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 exist: 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 $\varphi^{UF}$ with uninterpreted functions is reduced to equality logic formula $\varphi^E$ such that it is valid iff $\varphi^{UF}$ is valid.
- Axioms of functional consistency need to be modeled within $\varphi^E$ by auxiliary variables.
- $\varphi^E$ is of the form $FC^E \Rightarrow flat^E$ where $FC^E$ represents constraints for functional consistency and $flat^E$ is $\varphi^{UF}$ after replacement of functions with variables.
Ackermann’s reduction

- Single uninterpreted function with single argument
- Input: An EUF formula $\varphi^{UF}$
- Output: An equality logic formula $\varphi^E$ which is valid iff $\varphi^{UF}$ 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 $\tau$ is a function that replaces each occurrence of uninterpreted function $F_i$ with new variable $f_i$.
3. Let $FC^E$ denote the following conjunction of functional-consistency constraints:
   
   $FC^E := \land_i \land_j (\tau(arg(F_i)) = \tau(arg(F_j))) \Rightarrow f_i = f_j$

4. Return $\varphi^E := FC^E \Rightarrow flat^E$. 
Consider

\[ \varphi^{UF} := (x_1 \neq x_2) \lor (F(x_1) = F(x_2)) \lor (F(x_1) \neq F(x_3)) \]

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

- \( flat^E := (x_1 \neq x_2) \lor (f_1 = f_2) \lor (f_1 \neq f_3) \)
- \( FC^E := (x_1 = x_2) \Rightarrow (f_1 = f_2) \land \\
    (x_1 = x_3) \Rightarrow (f_1 = f_3) \land \\
    (x_2 = x_3) \Rightarrow (f_2 = f_3) \)
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
  - $flat^E := (x_1 = x_2) \Rightarrow (f_2 = f_4)$
  - $FC^E := (x_1 = x_2) \Rightarrow (g_1 = g_2) \land (g_1 = f_1) \Rightarrow (f_1 = f_2) \land \ldots$
Ackermann’s reduction - validity and satisfiability

- **Validity**
  - Checking validity of $\varphi^{UF}$ is reduced to checking validity of $\varphi^E := FC^E \Rightarrow flat^E$
  - Equivalently, unsatisfiability of $\neg\varphi^E := FC^E \land \neg flat^E$ can be checked.

- **Satisfiability**
  - Checking satisfiability of $\varphi^{UF}$ is reduced to satisfiability of $\varphi^E := FC^E \land flat^E$
    - Equivalently, non-validity of $\neg\varphi^{UF}$
    - Ackermann’s reduction of $\neg\varphi^{UF}$ yields $FC^E \Rightarrow \neg flat^E$ (same constraints for functional consistency).
    - Checking non-validity of $FC^E \Rightarrow \neg flat^E$ is the same as checking satisfiability of $FC^E \land flat^E$.  

Reduction from equality logic to propositional logic

- Graph-based reduction to propositional logic:
  - Propositional skeleton + transitivity constraints.
  - Transitivity constraints ensure the transitivity 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.