Satisfiability Checking

Equalities and Uninterpreted Functions

Prof. Dr. Erika Ábrahám

Theory of Hybrid Systems
Informatik 2

WS 10/11

Equality logic with uninterpreted functions

We extend the propositional logic with equalities and uninterpreted functions.

Syntax: variables x over an arbitrary domain D, constants c (from the same domain D), function symbols F for functions of the type $D^n \to D$.

Terms:	t	$:=$	c	$\mid$	x	$\mid$	$F(t, \ldots, t)$
Formulas:	φ	$:=$	$t = t$	$\mid$	$(\varphi \wedge \varphi)$	$\mid$	$(\neg\varphi)$

Semantics: straightforward

Notation and assumptions:

- Formula with equalities: φ^E
- Formula with equalities and uninterpreted functions: φ^{UF}
- Same simplifications for parentheses as for propositional logic.
- Input formulas are in NNF.
- Input formulas are checked for satisfiability.

Motivation

- Equality logic and propositional logic are both NP-complete.
- Thus they model the same decision problems.
- Why to study both?
  - Convenience of modeling
  - Efficiency
- Extensions: Different domains, Boolean variables

Motivation

- Replacing functions by uninterpreted functions in a given formula is a common technique to make reasoning easier.
- It makes the formula weaker: $\models \varphi^{UF} \rightarrow \varphi$
- Ignore the semantics of the function, but:
- Functional congruence: Instances of the same function return the same value for equal arguments.

Removing constants

Theorem

There is an algorithm that generates for an input equality logic formula φ^E an equisatisfiable output formula $\varphi^{E'}$ without constants, in polynomial time.

Algorithm: Exercise

In the following we assume that the formulas do not contain constants.

Outline

Outline

1 Conjunction of equalities

2 Conjunction of equalities with uninterpreted functions

3 Arbitrary Boolean combination of equalities
- Equality graphs
- The Sparse Method

4 Arbitrary Boolean combination of equalities with UFs

First: Conjunction of equalities without UF

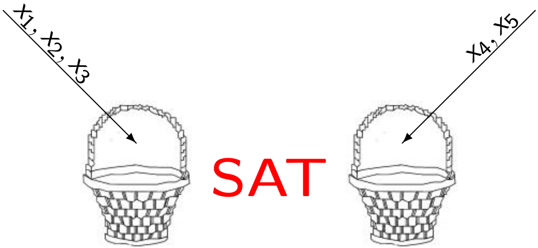
Input: A conjunction φ of equalities and disequalities without UF

Algorithm

1. Define an equivalence class for each variable in φ.
2. For each equality $x = y$ in φ: merge the equivalence classes of x and y.
3. For each disequality $x \neq y$ in φ:
   if x is in the same class as y, return 'UNSAT'.
4. Return 'SAT'.

Example

$$\varphi^E : \quad x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge x_5 \neq x_1$$



Equivalence class 1 Equivalence class 2

Outline

Next: Add uninterpreted functions

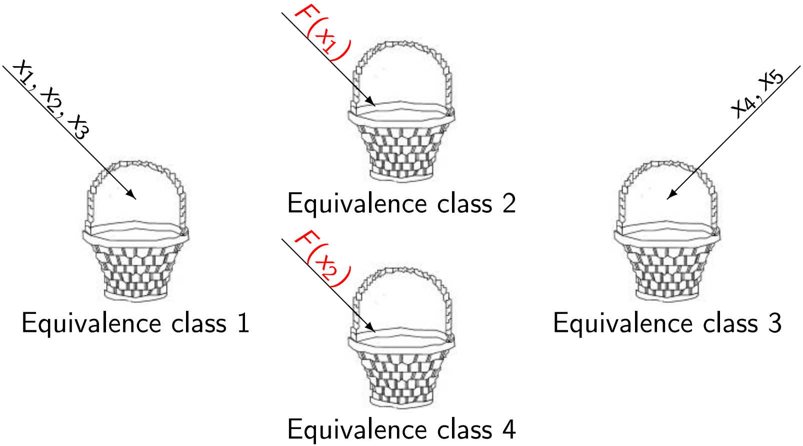
How do they relate?

- $x = y$, $F(x) = F(y)$: $\models (x = y) \rightarrow (F(x) = F(y))$
- $x = y$, $F(x) \neq F(y)$: conjunction unsatisfiable
- $x \neq y$, $F(x) = F(y)$: unrelated (conjunction satisfiable)
- $x \neq y$, $F(x) \neq F(y)$: $\models (F(x) \neq F(y)) \rightarrow (x \neq y)$

- $x = y$, $F(G(x)) = F(G(y))$: $\models (x = y) \rightarrow (F(G(x)) = F(G(y)))$

Next: Add uninterpreted functions

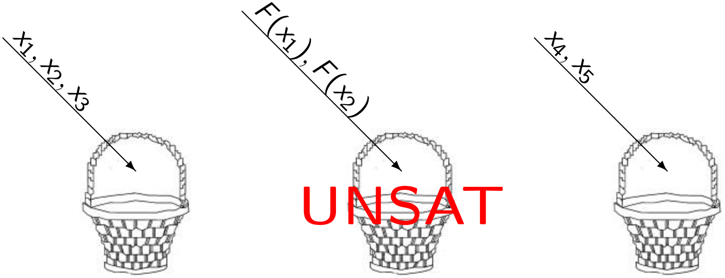
$$\varphi^E : \quad x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge x_5 \neq x_1 \wedge F(x_1) \neq F(x_2)$$



$$\varphi^E: \quad x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge x_5 \neq x_1 \wedge F(x_1) \neq F(x_2)$$

Congruence closure:

If all the arguments of two function applications are in the same class, merge the classes of the applications!



Equivalence class 1 Equivalence class 2 Equivalence class 3

Input: A conjunction φ of equalities and disequalities with UFs of type $D \to D$

Algorithm

1. $\mathcal{C} := \{\{t\} \mid t$ occurs as subexpression in an (in)equation in $\varphi\}$;
2. for each equality $t = t'$ in φ

   $\mathcal{C} := (\mathcal{C} \setminus \{[t], [t']\}) \cup \{[t] \cup [t']\}$;

   while exists $F(t), F(t')$ in φ with $[t] = [t']$ and $[F(t)] \neq [F(t')]$

   $\mathcal{C} := (\mathcal{C} \setminus \{[F(t)], [F(t')]\}) \cup \{[F(t)] \cup [F(t')]\}$;
3. for each inequality $t \neq t'$ in φ

   if $[t] = [t']$ return "UNSAT";
4. return "SAT";

Outline

1. Conjunction of equalities
2. Conjunction of equalities with uninterpreted functions
3. Arbitrary Boolean combination of equalities
   - Equality graphs
   - The Sparse Method
4. Arbitrary Boolean combination of equalities with UFs

Adding disjunctions

- One option: syntactic case-splitting,
  corresponds to transforming the formula to DNF.
- May result in exponential number of cases.
- Now we start looking at methods that split the search space instead.
  This is called semantic splitting.
- SAT is a very good engine for performing semantic splitting, due to its ability to guide the search, prune the search-space, and so on.

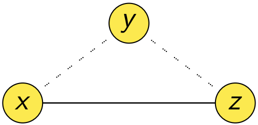
E-graphs

$$\varphi^E : x = y \land y = z \land z \neq x$$

- The equality predicates: $\{x = y, y = z, z \neq x\}$
- Break into two sets:

$$E_= = \{x = y, y = z\}, \quad E_{\neq} = \{z \neq x\}$$

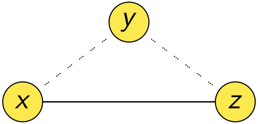
- The equality graph (E-graph) $G^E(\varphi^E) = \langle V, E_=, E_{\neq} \rangle$



The E-graph and Boolean structure in φ^E

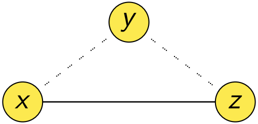
$$
\begin{array}{lll}
\varphi_1^E: & x = y \wedge y = z \wedge z \neq x & \text{unsatisfiable} \\
\varphi_2^E: & (x = y \wedge y = z) \vee z \neq x & \text{satisfiable!}
\end{array}
$$

Their E-graph is the same:



$\Longrightarrow$ The graph $G^E(\varphi^E)$ represents an abstraction of φ^E.
It ignores the Boolean structure of φ^E.

Equality and disequality paths



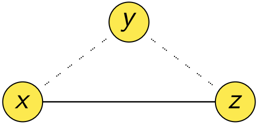
Definition (Equality Path)

A path that uses $E_=$ edges is an *equality path*. We write $x =^* z$.

Definition (Disequality Path)

A path that uses edges from $E_=$ and exactly one edge from $E_{\neq}$ is a *disequality path*. We write $x \neq^* z$.

Contradictory cycles



Definition (Contradictory Cycle)

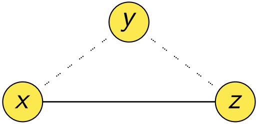
A cycle with one disequality edge is a *contradictory cycle*.

Theorem

For every two nodes x, y on a contradictory cycle the following holds:

- $x =^* y$
- $x \neq^* y$

Contradictory cycles



Definition

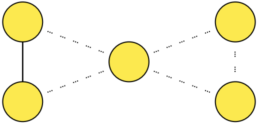
A subgraph of E is called *satisfiable* iff the conjunction of the predicates represented by its edges is satisfiable.

Theorem

A subgraph is unsatisfiable iff it contains a contradictory cycle.

Simple cycles

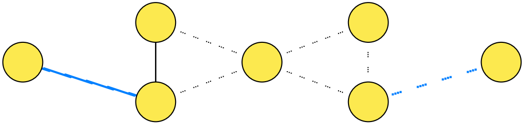
Question: What is a simple cycle?



Theorem

Every contradictory cycle is either simple, or contains a simple contradictory cycle.

Simplifying the E-graph

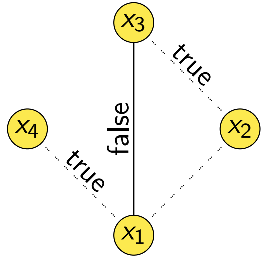


Let S be the set of edges that are not part of any contradictory cycle.

Theorem

Replacing all equations that correspond to solid edges in S with false, and all equations that correspond to dashed edges in S with true preserves satisfiability.

Simplifying the E-graph: Example

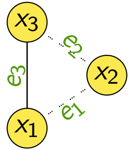


- $(x_1 = x_2 \quad \vee \quad x_1 = x_4) \quad \wedge$
  $(x_1 \neq x_3 \quad \vee \quad x_2 = x_3)$
- ~~$(x_1 = x_2 \quad \vee \quad \text{true} \quad) \quad \wedge$~~
  $(x_1 \neq x_3 \quad \vee \quad x_2 = x_3)$
- $\neg$false $\vee$ true
- $\longrightarrow$ Satisfiable!

Goal: Transform equality logic to propositional logic

Step 1: Encode all edges with Boolean variables

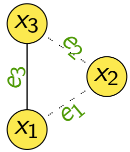
$$\varphi^E \iff x_1 = x_2 \wedge x_2 = x_3 \wedge x_1 \neq x_3$$
$$\varphi_{sk} \iff e_1 \wedge e_2 \wedge \neg e_3$$



- This is called the propositional skeleton
- This is an over-approximation
- Transitivity of equality is lost!
- $\rightarrow$ must add transitivity constraints!

Adding transitivity constraints

$$\varphi^E \iff x_1 = x_2 \wedge x_2 = x_3 \wedge x_1 \neq x_3$$
$$\varphi_{sk} \iff e_1 \wedge e_2 \wedge \neg e_3$$



Step 2: For each cycle: add a transitivity constraint

$$\varphi_{trans} = \begin{array}{l} (e_1 \wedge e_2 \longrightarrow e_3) \wedge \\ (e_1 \wedge e_3 \longrightarrow e_2) \wedge \\ (e_3 \wedge e_2 \longrightarrow e_1) \end{array}$$

Step 3: Check $\varphi_{sk} \wedge \varphi_{trans}$

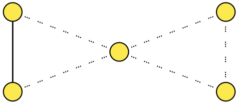
Question: Complexity?

Optimizations

There can be an *exponential number of cycles*, so let's try to improve this idea.

Theorem

It is sufficient to constrain simple cycles only.



Only two simple cycles here.

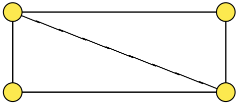
Question: Complexity?

Optimizations

Still, there may be an exponential number of simple cycles.

Theorem

It is sufficient to constrain chord-free simple cycles.



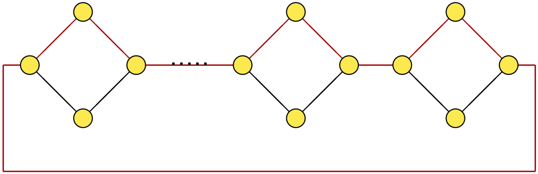
Question: How many simple cycles?

Question: How many chord-free simple cycles?

Question: Complexity?

Optimizations

Still, there may be an exponential number of chord-free simple cycles...



Solution: make graph 'chordal' by adding edges!

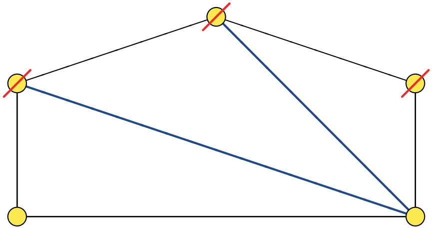
Making the E-graph chordal

Definition (Chordal graph)

A graph is *chordal* iff every cycle of length 4 or more has a chord.

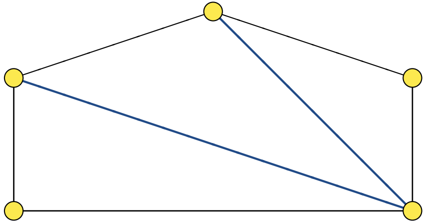
Question: How to make a graph chordal?

A: Eliminate vertices one at a time, and connect their neighbors.



Making the E-graph chordal

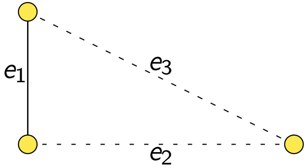
- Once the graph is chordal, we only need to constrain the triangles.



- Note that this procedure adds not more than a polynomial number of edges, and results in a polynomial number of constraints.

Exploiting the polarity

- So far we did not consider the polarity of the edges.
- Claim: in the following graph, $\varphi_{trans} = e_2 \land e_3 \longrightarrow e_1$ is sufficient.



- This works because of the monotonicity of NNF.

Equality logic to propositional logic

- Input: Equality logic formula φ^E
- Output: satisfiability-equivalent propositional logic formula φ^E

Algorithm

1. Construct φ_{sk} by replacing each equality $t_i = t_j$ in φ^E by a fresh Boolean variable $e_{i,j}$.
2. Construct the E-graph $G^E(\varphi^E)$ for φ^E.
3. Make $G^E(\varphi^E)$ chordal.
4. $\varphi_{trans} = true$.
5. For each triangle $(e_{i,j}, e_{j,k}, e_{k,i})$ in $G^E(\varphi^E)$:

$$\begin{aligned}\varphi_{trans} := \varphi_{trans} \quad & \wedge (e_{i,j} \wedge e_{j,k}) \rightarrow e_{k,i} \\ & \wedge (e_{i,j} \wedge e_{i,k}) \rightarrow e_{j,k} \\ & \wedge (e_{i,k} \wedge e_{j,k}) \rightarrow e_{i,j}\end{aligned}$$

6. Return $\varphi_{sk} \wedge \varphi_{trans}$.

Outline

From uninterpreted functions to equality logic

We lead back the problems of equality logic with uninterpreted functions to those of equality logic without uninterpreted functions.

Two possible reductions:

- Ackermann's reduction
- Bryant's reduction

We look only at Ackermann.

Ackermann's reduction

Given an input formula φ^{UF} of equality logic with uninterpreted functions, transform the formula to a satisfiability-equivalent equality logic formula φ^{E} of the form

$$\varphi^{E} := \varphi_{flat} \wedge \varphi_{cong},$$

where φ_{flat} is a flattening of φ^{UF}, and φ_{cong} is a conjunction of constraints for functional congruence.

For validity-equivalence check

$$\varphi^{E} := \varphi_{cong} \rightarrow \varphi_{flat}.$$

Note: This is quite similar to leading back equality logic to propositional logic by

$$\varphi_{sk} \wedge \varphi_{trans}.$$

Ackermann's reduction

- Input: φ^{UF} with m instances of an uninterpreted function F.
- Output: satisfiability-equivalent φ^{E} without any occurrences of F.

Algorithm

1. Assign indices to the F-instances.
2. $\varphi_{flat} := \mathcal{T}(\varphi^{UF})$ where $\mathcal{T}$ replaces each occurrence F_i of F by a fresh Boolean variable f_i.
3. $\varphi_{cong} := \bigwedge_{i=1}^{m-1} \bigwedge_{j=i+1}^{m} (\mathcal{T}(arg(F_i)) = \mathcal{T}(arg(F_j))) \rightarrow f_i = f_j$
4. Return $\varphi_{flat} \wedge \varphi_{cong}$.

Ackermann's reduction: Example

$$
\begin{array}{lcl}
\varphi^{UF} & := & (x_1 \neq x_2) \vee (F(x_1) = F(x_2)) \vee (F(x_1) \neq F(x_3)) \\
\varphi_{flat} & := & (x_1 \neq x_2) \vee (f_1 = f_2) \vee (f_1 \neq f_3) \\
FC^E & := & ((x_1 = x_2) \quad \rightarrow \quad (f_1 = f_2)) \wedge \\
 & & ((x_1 = x_3) \quad \rightarrow \quad (f_1 = f_3)) \wedge \\
 & & ((x_2 = x_3) \quad \rightarrow \quad (f_2 = f_3)) \\
\varphi^E & := & \varphi_{cong} \wedge \varphi_{flat}
\end{array}
$$

Ackermann's reduction: Example

- int power3 (int in){
  int out = in;
  for (int i=0; i<2; i++)
  out = out * in;
  return out;
  }
- int power3_b (int in){
  return ((in * in) * in);
  }

- $\varphi_1 \;:=\; out_0 = in \wedge out_1 = out_0 * in \wedge out_2 = out_1 * in$
- $\varphi_2 \;:=\; out_b = (in * in) * in$
- $\varphi_3 \;:=\; (\varphi_1 \wedge \varphi_2) \rightarrow (out_2 = out_b)$

Ackermann's reduction: Example

$$\begin{aligned}\varphi_3 \quad := \quad & (out_0 = in \wedge out_1 = out_0 * in \wedge \\ & out_2 = out_1 * in \wedge out_b = (in * in) * in) \rightarrow \\ & (out_2 = out_b)\end{aligned}$$

$$\begin{aligned}\varphi^{UF} \quad := \quad & (out_0 = in \wedge out_1 = G(out_0, in) \wedge \\ & out_2 = G(out_1, in) \wedge out_b = G(G(in, in), in)) \rightarrow \\ & (out_2 = out_b)\end{aligned}$$

Ackermann's reduction: Example

$$\varphi^{UF} := (out_0 = in \wedge out_1 = G(out_0, in) \wedge out_2 = G(out_1, in) \wedge out_b = G(G(in, in), in)) \rightarrow (out_2 = out_b)$$

$$\varphi_{flat} := (out_0 = in \wedge out_1 = G_1 \wedge out_2 = G_2 \wedge out_b = G_4) \rightarrow (out_2 = out_b) \text{ with}$$

$$\begin{array}{lcl} \varphi_{cong} := & ((out_0 = out_1 \wedge in = in) & \rightarrow & G_1 = G_2)\wedge \\ & ((out_0 = in \wedge in = in) & \rightarrow & G_1 = G_3)\wedge \\ & ((out_0 = G_3 \wedge in = in) & \rightarrow & G_1 = G_4)\wedge \\ & ((out_1 = in \wedge in = in & \rightarrow & G_2 = G_3)\wedge \\ & ((out_1 = G_3 \wedge in = in) & \rightarrow & G_2 = G_4)\wedge \\ & ((in = G_3 \wedge in = in) & \rightarrow & G_3 = G_4) \end{array}$$