Satisfiability Checking
Eager SMT Solving (Equality Logic, Bit-blasting)

Prof. Dr. Erika Ábrahám

RWTH Aachen University
Informatik 2
LuFG Theory of Hybrid Systems

WS 14/15
Outline

1. Eager SMT Solving

2. Eager SMT Solving for Equality Logic with Uninterpreted Functions
   - Equality Logic with Uninterpreted Functions
   - Eager SMT Solving for Uninterpreted Functions
     - Ackermann’s reduction
     - Bryant’s reduction
   - Eager SMT Solving for Equality Logic
     - Equality Graphs
     - The Sparse Method

3. Eager SMT Solving for Finite-precision Bit-vector Arithmetic
Outline

1 Eager SMT Solving

2 Eager SMT Solving for Equality Logic with Uninterpreted Functions
   - Equality Logic with Uninterpreted Functions
   - Eager SMT Solving for Uninterpreted Functions
     - Ackermann’s reduction
     - Bryant’s reduction
   - Eager SMT Solving for Equality Logic
     - Equality Graphs
     - The Sparse Method

3 Eager SMT Solving for Finite-precision Bit-vector Arithmetic
We want to extend propositional logic with theories.

For satisfiability checking, SAT-solving will be extended to SAT-modulo-theories (SMT) solving.

SMT-LIB: language, benchmarks, tutorials, ...

SMT-COMP: performance and capabilities of tools

SMT Workshop: held annually
How can such an extension to SMT solving look like?

We will see two basically different approaches:

- **Eager SMT solving** transforms logical formulas over some theories into satisfiability-equivalent propositional logic formulas and applies SAT solving. (“Eager” means theory first)
- **Lazy SMT solving** uses a SAT solver to find solutions for the Boolean skeleton of the formula, and a theory solver to check satisfiability in the underlying theory. (“Lazy” means theory later)

Today we will have a closer look at the **eager approach**.
All NP-complete problems can be transformed to equivalent propositional SAT problems (with polynomial effort).

However, this is not always effective in praxis (the transformation would sometimes solve the hardest part of the problem).

Some well-suited theories for eager SMT solving:
- Equalities and uninterpreted functions
- Finite-precision bit-vector arithmetic
- Quantifier-free linear integer arithmetic (QF_LIA)
- Restricted $\lambda$-calculus (e.g., arrays)
- ...
- Combinations of the above theories
Some Eager SMT Solver Implementations

- **UCLID**: Proof-based abstraction-refinement [Bryant et al., TACAS’07]
- **STP**: Solver for linear modular arithmetic to simplify the formula [Ganesh&Dill, CAV’07]
- **Spear**: Automatic parameter tuning for SAT [Hutter et al., FMCAD’07]
- **Boolector**: Rewrites, underapproximation, efficient SAT engine [Brummayer&Biere, TACAS’09]
- **Beaver**: Equality/constant propagation, logic optimization, special rules for non-linear operations [Jha et al., CAV’09]
- **SONOLAR**: Non-linear arithmetic [Brummayer et al., SMT’08]
- **SWORD**: Fixed-size bit-vectors [Jung et al, SMTCOMP’09]
- Layered eager approaches embedded in the lazy DPLL(T) framework: 
  - **CVC3** [Barrett et al.], 
  - **MathSAT** [Bruttomesso et al.], 
  - **Z3** [de Moura et al.]
Outline

1. Eager SMT Solving

2. Eager SMT Solving for Equality Logic with Uninterpreted Functions
   - Equality Logic with Uninterpreted Functions
   - Eager SMT Solving for Uninterpreted Functions
     - Ackermann’s reduction
     - Bryant’s reduction
   - Eager SMT Solving for Equality Logic
     - Equality Graphs
     - The Sparse Method

3. Eager SMT Solving for Finite-precision Bit-vector Arithmetic
Equality logic with uninterpreted functions

We extend propositional logic with
- **equalities** and
- **uninterpreted functions (UFs).**

**Syntax:**
- **variables** \( x \) over an arbitrary domain \( D \),
- **constants** \( c \) from the same domain \( D \),
- **function symbols** \( F \) for functions of the type \( D^n \to D \), and
- **equality** as predicate symbol.

\[
\text{Terms: } t ::= c \mid x \mid F(t, \ldots, t)
\]

\[
\text{Formulas: } \varphi ::= t = t \mid (\varphi \land \varphi) \mid (\neg \varphi)
\]

**Semantics:** straightforward
Motivation

- Equality logic and propositional logic are both $\text{NP-complete}$.
- Thus they model the same decision problems.
- Why to study both?
  - Convenience of modeling
  - Efficiency
Equality logic with uninterpreted functions

Notation and assumptions:

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

**Theorem**

*There is an algorithm that generates for an input formula $\varphi_{UF}$ an equisatisfiable output formula $\varphi_{UF}'$ without constants, in polynomial time.*

**Algorithm:** Exercise

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

1 Eager SMT Solving

2 Eager SMT Solving for Equality Logic with Uninterpreted Functions
   - Equality Logic with Uninterpreted Functions
   - Eager SMT Solving for Uninterpreted Functions
     - Ackermann’s reduction
     - Bryant’s reduction
   - Eager SMT Solving for Equality Logic
     - Equality Graphs
     - The Sparse Method

3 Eager SMT Solving for Finite-precision Bit-vector Arithmetic
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.
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
Outline

1 Eager SMT Solving

2 Eager SMT Solving for Equality Logic with Uninterpreted Functions
   - Equality Logic with Uninterpreted Functions
   - Eager SMT Solving for Uninterpreted Functions
     - Ackermann’s reduction
     - Bryant’s reduction
   - Eager SMT Solving for Equality Logic
     - Equality Graphs
     - The Sparse Method

3 Eager SMT Solving for Finite-precision Bit-vector Arithmetic
Given an input formula $\varphi^{UF}$ of equality logic with uninterpreted functions, transform the formula to a satisfiability-equivalent equality logic formula $\varphi^{E}$ of the form

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

where $\varphi_{\text{flat}}$ is a flattening of $\varphi^{UF}$, and $\varphi_{\text{cong}}$ is a conjunction of constraints for functional congruence.

For validity-equivalence check

$$\varphi^{E} := \varphi_{\text{cong}} \rightarrow \varphi_{\text{flat}}.$$
Ackermann’s reduction

- **Input:** \( \varphi^{UF} \) with \( m \) instances of an uninterpreted function \( F \).
- **Output:** satisfiability-equivalent \( \varphi^E \) without any occurrences of \( F \).

**Algorithm**

1. Assign indices to the \( F \)-instances.
2. \( \varphi_{flat} := T(\varphi^{UF}) \) where \( T \) replaces each occurrence \( F_i \) of \( F \) by a fresh Boolean variable \( f_i \).
3. \( \varphi_{cong} := \bigwedge_{i=1}^{m-1} \bigwedge_{j=i+1}^{m} (T(\text{arg}(F_i)) = T(\text{arg}(F_j))) \rightarrow f_i = f_j \)
4. Return \( \varphi_{flat} \land \varphi_{cong} \).
Ackermann’s reduction: Example

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

\[ \varphi_{flat} := (x_1 \neq x_2) \lor (f_1 = f_2) \lor (f_1 \neq f_3) \]

\[ \varphi_{cong} := ((x_1 = x_2) \rightarrow (f_1 = f_2)) \land \\
                  ((x_1 = x_3) \rightarrow (f_1 = f_3)) \land \\
                  ((x_2 = x_3) \rightarrow (f_2 = f_3)) \]

\[ \varphi^E := \varphi_{flat} \land \varphi_{cong} \]
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);`  
  `}`

- $\varphi_1 := out_0 = in \land out_1 = out_0 * in \land out_2 = out_1 * in$
- $\varphi_2 := out_b = (in * in) * in$
- $\varphi_3 := (\varphi_1 \land \varphi_2) \rightarrow (out_2 = out_b)$
Ackermann’s reduction: Example

\[ \varphi_3 := (out_0 = in \land out_1 = out_0 \ast in \land out_2 = out_1 \ast in \land out_b = (in \ast in) \ast in) \rightarrow (out_2 = out_b) \]

\[ \varphi^{UF} := (out_0 = in \land out_1 = G(out_0, in) \land out_2 = G(out_1, in) \land out_b = G(G(in, in), in)) \rightarrow (out_2 = out_b) \]
Ackermann’s reduction: Example

\[ \varphi_{UF} := (\text{out}_0 = \text{in} \land \text{out}_1 = G(\text{out}_0, \text{in}) \land \text{out}_2 = G(\text{out}_1, \text{in}) \land \text{out}_b = G(G(\text{in}, \text{in}), \text{in})) \rightarrow (\text{out}_2 = \text{out}_b) \]

\[ \varphi_{flat} := (\text{out}_0 = \text{in} \land \text{out}_1 = G_1 \land \text{out}_2 = G_2 \land \text{out}_b = G_4) \rightarrow (\text{out}_2 = \text{out}_b) \]
with

\[ \varphi_{cong} := ((\text{out}_0 = \text{out}_1 \land \text{in} = \text{in}) \rightarrow G_1 = G_2) \land ((\text{out}_0 = \text{in} \land \text{in} = \text{in}) \rightarrow G_1 = G_3) \land ((\text{out}_0 = G_3 \land \text{in} = \text{in}) \rightarrow G_1 = G_4) \land ((\text{out}_1 = \text{in} \land \text{in} = \text{in}) \rightarrow G_2 = G_3) \land ((\text{out}_1 = G_3 \land \text{in} = \text{in}) \rightarrow G_2 = G_4) \land ((\text{in} = G_3 \land \text{in} = \text{in}) \rightarrow G_3 = G_4) \]
Outline

1 Eager SMT Solving

2 Eager SMT Solving for Equality Logic with Uninterpreted Functions
   - Equality Logic with Uninterpreted Functions
   - Eager SMT Solving for Uninterpreted Functions
     - Ackermann’s reduction
     - Bryant’s reduction
   - Eager SMT Solving for Equality Logic
     - Equality Graphs
     - The Sparse Method

3 Eager SMT Solving for Finite-precision Bit-vector Arithmetic
Bryant’s reduction

- Case expression:

\[ F_i^* = \text{case} \quad \begin{align*}
  x_1 = x_i & : f_1 \\
  x_2 = x_i & : f_2 \\
  \vdots \\
  x_{i-1} = x_i & : f_{i-1} \\
  \text{true} & : f_i
\end{align*} \]

where \( x_i \) is the argument \( \text{arg}(F_i) \) of \( F_i \) for all \( i \).

- Semantics:

\[
\bigvee_{j=1}^{i} \left( \left( \bigwedge_{k=1}^{j-1} (x_k \neq x_i) \right) \land (x_j = x_i) \land (F_i^* = f_j) \right)
\]
Bryant’s reduction

- **Input:** \( \varphi^{UF} \) with \( m \) instances of an uninterpreted function \( F \).
- **Output:** satisfiability-equivalent \( \varphi^E \) without any occurrences of \( F \).

**Algorithm**

1. Assign indices to the \( F \)-instances.
2. Return \( \mathcal{T}^*(\varphi^{UF}) \) where \( \mathcal{T}^* \) replaces each \( F_i(\text{arg}(F_i)) \) by

\[
\text{case } \mathcal{T}^*(\text{arg}(F_1)) = \mathcal{T}^*(\text{arg}(F_i)) : f_1 \\
\cdots \\
\mathcal{T}^*(\text{arg}(F_{i-1})) = \mathcal{T}^*(\text{arg}(F_i)) : f_{i-1} \\
\text{true} : f_i
\]
Bryant’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);
}

- $\phi_1 := out_0 = in \land out_1 = out_0 \ast in \land out_2 = out_1 \ast in$
- $\phi_2 := out_b = (in \ast in) \ast in$
- $\phi_3 := (\phi_1 \land \phi_2) \rightarrow (out_2 = out_b)$
Bryant’s reduction: Example

\[ \varphi_3 := (out_0 = in \land out_1 = out_0 \ast in \land out_2 = out_1 \ast in \land out_b = (in \ast in) \ast in) \rightarrow (out_2 = out_b) \]

\[ \varphi_{UF} := (out_0 = in \land out_1 = G(out_0, in) \land out_2 = G(out_1, in) \land out_b = G(G(in, in), in)) \rightarrow (out_2 = out_b) \]
Bryant’s reduction: Example

\[ \varphi_{UF} := (out_0 = in \land out_1 = G(out_0, in) \land out_2 = G(out_1, in) \land \]
\[ out_b = G(G(in, in), in)) \rightarrow (out_2 = out_b) \]

\[ \varphi^E := (out_0 = in \land out_1 = G_1^* \land out_2 = G_2^* \land \]
\[ out_b = G_4^*) \rightarrow (out_2 = out_b) \]

with

\[ G_1^* = \]
\[ G_2^* = case \]
\[ G_3^* = case \]
\[ G_4^* = case \]

\[ out_0 = out_1 \land in = in : g_1 \]
\[ true : g_2 \]
\[ out_0 = in \land in = in : g_1 \]
\[ out_1 = in \land in = in : g_2 \]
\[ true : g_3 \]
\[ out_0 = G_3^* \land in = in : g_1 \]
\[ out_1 = G_3^* \land in = in : g_2 \]
\[ in = G_3^* \land in = in : g_3 \]
\[ true : g_4 \]
1. Eager SMT Solving

2. Eager SMT Solving for Equality Logic with Uninterpreted Functions
   - Equality Logic with Uninterpreted Functions
   - Eager SMT Solving for Uninterpreted Functions
     - Ackermann’s reduction
     - Bryant’s reduction
   - Eager SMT Solving for Equality Logic
     - Equality Graphs
     - The Sparse Method

3. Eager SMT Solving for Finite-precision Bit-vector Arithmetic
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:
  
  \[ E_\equiv = \{x = y, y = z\}, \quad E_\not\equiv = \{z \neq x\} \]

- The equality graph (E-graph) $G^E(\varphi^E) = \langle V, E_\equiv, E_\not\equiv \rangle$
The E-graph and Boolean structure in $\varphi^E$

$\varphi_1^E : \ x = y \land y = z \land z \neq x \quad$ unsatisfiable

$\varphi_2^E : \ (x = y \land y = z) \lor z \neq x \quad$ satisfiable!

Their E-graph is the same:

$\Rightarrow$ The graph $G^E(\varphi^E)$ represents an abstraction of $\varphi^E$.
It ignores the Boolean structure of $\varphi^E$. 
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 $x \neq^* z$. 
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:*

- $x =^* y$
- $x \not=^* y$
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.
**Question:** What is a simple cycle?

![Diagram of a simple cycle](image)

**Theorem**

*Every contradictory cycle is either simple, or contains a simple contradictory cycle.*
Let $S$ be the set of edges that are not part of any contradictory cycle.

**Theorem**

Replacing

- all equations in $\varphi^E$ that correspond to solid edges in $S$ with false, and
- all equations in $\varphi^E$ that correspond to dashed edges in $S$ with true

preserves satisfiability.
Simplifying the E-graph: Example

\[\begin{align*}
(x_1 = x_2 & \lor x_1 = x_4) \land \\
(x_1 \neq x_3 & \lor x_2 = x_3) \\
(x_1 = x_2 & \lor \text{true}) \land \\
(x_1 \neq x_3 & \lor x_2 = x_3) \\
(x_1 \neq x_3 & \lor x_2 = x_3) \\
\neg \text{false} & \lor \text{true} \\
\rightarrow \text{Satisfiable!}
\end{align*}\]
Outline

1 Eager SMT Solving

2 Eager SMT Solving for Equality Logic with Uninterpreted Functions
   - Equality Logic with Uninterpreted Functions
   - Eager SMT Solving for Uninterpreted Functions
     - Ackermann’s reduction
     - Bryant’s reduction
   - Eager SMT Solving for Equality Logic
     - Equality Graphs
     - The Sparse Method

3 Eager SMT Solving for Finite-precision Bit-vector Arithmetic
Bryant & Velev 2000: The Sparse method

Goal: Transform equality logic to propositional logic

Step 1: Replace all equalities in the formula by Boolean variables

\[
\varphi^E \leftrightarrow x_1 = x_2 \land x_2 = x_3 \land x_1 \neq x_3
\]

\[
\varphi_{sk} \leftrightarrow e_1 \land e_2 \land \neg e_3
\]

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

\[ \varphi^E \iff x_1 = x_2 \land x_2 = x_3 \land x_1 \neq x_3 \]
\[ \varphi_{sk} \iff e_1 \land e_2 \land \neg e_3 \]

**Step 2:** For each cycle in the equality graph: add a transitivity constraint

\[ \varphi_{trans} = (e_1 \land e_2 \rightarrow e_3) \land (e_1 \land e_3 \rightarrow e_2) \land (e_3 \land e_2 \rightarrow e_1) \]

**Step 3:** Check \( \varphi_{sk} \land \varphi_{trans} \)

**Question:** Complexity?
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?
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**: Iteratively connect the neighbors of the vertices.
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.
So far we did not consider the polarity of the edges. Claim: in the following graph, $\varphi_{\text{trans}} = e_2 \land e_3 \rightarrow e_1$ is sufficient.

This works because of the monotonicity of NNF.
Equality logic to propositional logic

- **Input:** Equality logic formula $\varphi^E$
- **Output:** satisfiability-equivalent propositional logic formula $\varphi^E$

**Algorithm**

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

6. Return $\varphi_{sk} \land \varphi_{trans}$. 
Outline

1. Eager SMT Solving

2. Eager SMT Solving for Equality Logic with Uninterpreted Functions
   - Equality Logic with Uninterpreted Functions
   - Eager SMT Solving for Uninterpreted Functions
     - Ackermann’s reduction
     - Bryant’s reduction
   - Eager SMT Solving for Equality Logic
     - Equality Graphs
     - The Sparse Method

3. Eager SMT Solving for Finite-precision Bit-vector Arithmetic
Finite-precision bit-vector arithmetic

“Bit blasting”:

- Model bit-level operations (functions and predicates) by Boolean circuits
- Use Tseitin’s encoding to generate propositional SAT encoding
- Use a SAT solver to check satisfiability
- Convert back the propositional solution to the theory

Effective solution for many applications.

- Example: Bounded model checking for C programs (CBMC) [Clarke, Kroening, Lerda, TACAS’04]
...from the Decision Procedures website.