Satisfiability Checking

10 Summary I

Prof. Dr. Erika Ábrahám

RWTH Aachen University
Informatik 2
LuFG Theory of Hybrid Systems

WS 22/23

Outline

1. Propositional logic, theories, normal forms
2. DPLL+CDCL SAT solving
3. Eager SMT-solving: Equality logic with uninterpreted functions
   - From UF to EQ: Ackermann's reduction
   - From EQ to SAT: The Sparse method
4. Eager SMT solving: Finite-precision bit-vector arithmetic

10 Summary I

1 Propositional logic, theories, normal forms

2 DPLL+CDCL SAT solving

3 Eager SMT-solving: Equality logic with uninterpreted functions
- From UF to EQ: Ackermann's reduction
- From EQ to SAT: The Sparse method

4 Eager SMT solving: Finite-precision bit-vector arithmetic

Propositional logic: Syntax

- Abstract grammar:

$$\varphi \ := \ AP \ | \ (\neg\varphi) \ | \ (\varphi \land \varphi)$$

  with $AP \in AP$.

- Syntactic sugar:

$$\begin{array}{cccccl} & & \bot & & & := (a \land \neg a) \\ & & \top & & & := (a \lor \neg a) \\ (& \varphi_1 & \lor & \varphi_2 &) & := \neg((\neg\varphi_1) \land (\neg\varphi_2)) \\ (& \varphi_1 & \rightarrow & \varphi_2 &) & := ((\neg\varphi_1) \lor \varphi_2) \\ (& \varphi_1 & \leftrightarrow & \varphi_2 &) & := ((\varphi_1 \rightarrow \varphi_2) \land (\varphi_2 \rightarrow \varphi_1)) \\ (& \varphi_1 & \bigoplus & \varphi_2 &) & := (\varphi_1 \leftrightarrow (\neg\varphi_2)) \end{array}$$

Propositional logic: Semantics

- Structures for predicate logic:
  - Domain: $\mathbb{B} = \{0, 1\}$
  - Interpretation: assignment $\alpha : AP \to \{0, 1\}$
    Assign: set of all assignments
    Equivalently: $\alpha \in 2^{AP}$ or $\alpha \in \{0, 1\}^{AP}$
- Semantics: $\models \subseteq$ $(Assign \times \text{Formula})$ is defined recursively:

$\alpha \models p$ iff $\alpha(p) = \text{true}$
$\alpha \models \neg\varphi$ iff $\alpha \not\models \varphi$
$\alpha \models \varphi_1 \wedge \varphi_2$ iff $\alpha \models \varphi_1$ and $\alpha \models \varphi_2$

$\alpha \models \varphi_1 \vee \varphi_2$ iff $\alpha \models \varphi_1$ or $\alpha \models \varphi_2$
$\alpha \models \varphi_1 \rightarrow \varphi_2$ iff $\alpha \models \varphi_1$ implies $\alpha \models \varphi_2$
$\alpha \models \varphi_1 \leftrightarrow \varphi_2$ iff $\alpha \models \varphi_2$ iff $\alpha \models \varphi_2$
$\alpha \models \varphi_1 \bigoplus \varphi_2$ iff $\alpha \models \varphi_2$ iff $\alpha \not\models \varphi_2$

Logic extensions: Theories

Propositional logic	$(x \vee y) \wedge (\neg x \vee y)$
Equality	$(x = y \wedge y \neq z) \rightarrow (x \neq z)$
Uninterpreted functions	$(F(x) = F(y) \wedge y = z) \rightarrow F(x) = F(z)$
Linear real/integer arithmetic	$2x + y > 0 \wedge x + y \leq 0$ $2x = 1$
Real algebra	$x^2 + 2xy + y^2 < 0$

Normal forms

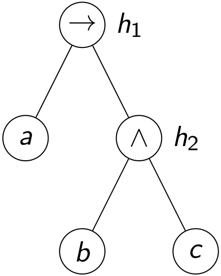
Negation Normal Form (NNF)	Arbitrarily nested disjunctions and conjunctions over atomic constraints and their negation.
Disjunctive Normal Form (DNF)	Disjunction of conjunctions of literals. $\bigvee_i \bigwedge_j \ell_{i,j}$
Conjunctive Normal Form (CNF)	Conjunction of disjunctions of literals. $\bigwedge_i \bigvee_j \ell_{i,j}$

Converting to CNF: Tseitin's encoding

- Formula:

  $\phi = (a \rightarrow (b \wedge c))$

The parse tree:



- Gate encodings:

  $(h_1 \leftrightarrow (a \ \rightarrow \ h_2))\wedge$

  $(h_2 \leftrightarrow (b \wedge c))\wedge$

  (h_1)

- Each gate encoding has a CNF representation with 3 or 4 clauses.

10 Summary I

1. Propositional logic, theories, normal forms

2. DPLL+CDCL SAT solving

3. Eager SMT-solving: Equality logic with uninterpreted functions
   - From UF to EQ: Ackermann's reduction
   - From EQ to SAT: The Sparse method

4. Eager SMT solving: Finite-precision bit-vector arithmetic

Clause status under partial assignments

- Given a partial assignment, a clause can be

Satisfied:	at least one literal is true
Unsatisfied:	all literals are false
	$\rightarrow$ conflict
Unit:	one literal is unassigned, the remaining literals are false
	$\rightarrow$ propagation
Unresolved:	all other cases

- Example: $C = (x_1 \vee x_2 \vee x_3)$

x_1	x_2	x_3	C
1	0		satisfied
0	0	0	unsatisfied
0	0		unit
	0		unresolved

The basic DPLL+CDCL SAT algorithm

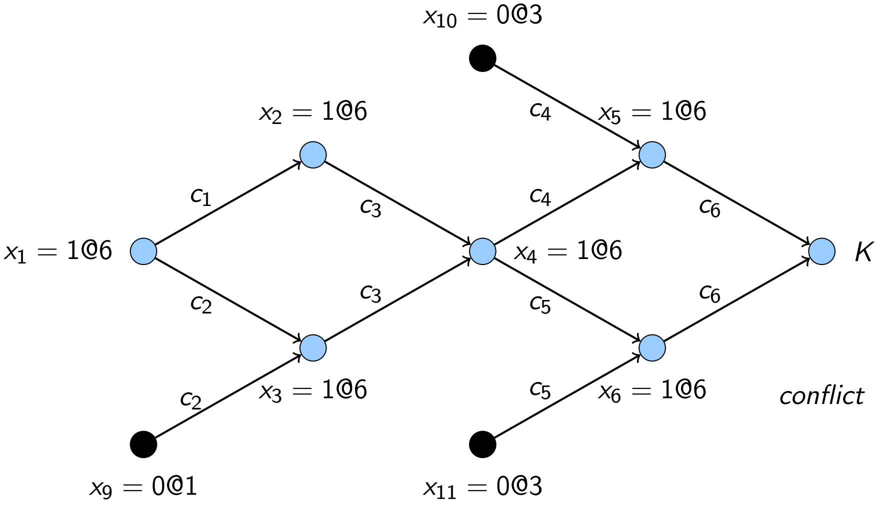
```
if (!BCP()) return UNSAT;
while (true)
{
        if (!decide()) return SAT;
        while (!BCP())
                if (!resolve_conflict()) return UNSAT;
}
```

Choose the next variable and value.
Return false if all variables are assigned.

Boolean constraint propagation with watched literals.
Return false if reached a conflict.

Conflict resolution and backtracking. Return false if impossible.

Implication graph



Conflict resolution

The resolution inference rule for CNF:

$$\frac{(l \vee l_1 \vee l_2 \vee \ldots \vee l_n) \quad (\neg l \vee l'_1 \vee \ldots \vee l'_m)}{(l_1 \vee \ldots \vee l_n \vee l'_1 \vee \ldots \vee l'_m)} \text{ Resolution}$$

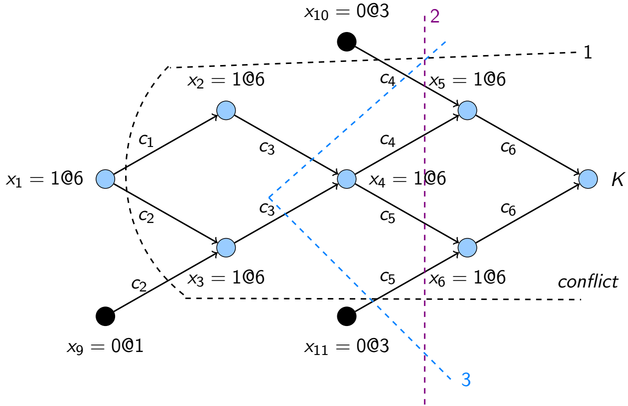
Example:

$$\frac{(a \vee b) \quad (\neg a \vee c)}{(b \vee c)}$$

- Resolution is a sound and complete inference system for CNF.
- The input formula is unsatisfiable iff there exists a proof of the empty clause.

Conflict resolution and non-chronological backtracking

Apply resolution up in the implication tree until a UIP (Unique Implication Point) has been reached:



1.$(x_{10} \vee \neg x_1 \vee x_9 \vee x_{11})$

2.$(x_{10} \vee \neg x_4 \vee x_{11})$

3.$(x_{10} \vee \neg x_2 \vee \neg x_3 \vee x_{11})$

⋮

⋮

- Backtrack to the second largest decision level in the conflict clause.
- This resolves the conflict and triggers an implication by the new conflict clause.

Decision heuristics - VSIDS

VSIDS(Variable State Independent Decaying Sum)

1. Each variable has an activity initialized to 0.
2. When resolution gets applied to a clause, the activities of its literals are increased.
3. Decision: The unassigned variable with the highest activity is chosen.
4. Periodically, all the activities are divided by a constant.

10 Summary I

1. Propositional logic, theories, normal forms
2. DPLL+CDCL SAT solving
3. Eager SMT-solving: Equality logic with uninterpreted functions
   - From UF to EQ: Ackermann's reduction
   - From EQ to SAT: The Sparse method
4. Eager SMT solving: 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.

$$\begin{array}{lllllll} \textit{Terms:} & t & := & c & \mid & x & \mid & F(t, \ldots, t) \\ \textit{Formulas:} & \varphi & := & t = t & \mid & (\varphi \land \varphi) & \mid & (\neg \varphi) \end{array}$$

Semantics: straightforward

From UF to EQ: Ackermann's reduction

- Input: φ^{UF}
- Output: Satisfiability-equivalent φ^{EQ} without UF

Algorithm

1. Let $\varphi_{flat} := \varphi^{UF}$ and $Inst = \emptyset$.
2. While φ contains some UF:
   Choose UF-instanz $F(t_1, \ldots, t_n)$ in φ_{flat} with UF-free arguments.
   Choose fresh (theory) variable F_i.
   Replace each occurrence of $F(t_1, \ldots, t_n)$ in φ_{flat} by F_i.
   Add $(F(t_1, \ldots, t_n), F_i)$ to $Inst$.
3. Let $\varphi_{cong} := true$.
4. While $Inst \neq \emptyset$:
   Choose and remove some $(F(t_1, \ldots, t_n), F_i)$ from $Inst$.
   $\varphi_{cong} := \varphi_{cong} \wedge \bigwedge_{(F(t'_1,\ldots,t'_n),F_j)\in Inst} ((\bigwedge_{k=1}^{n} t_k = t'_k) \rightarrow F_i = F_j)$.
5. Return $\varphi_{flat} \wedge \varphi_{cong}$.

From EQ to SAT: The Sparse method

- Input: Equality logic formula φ^E
- Output: Satisfiability-equivalent propositional logic formula φ^{EQ}

Algorithm

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

1. Propositional logic, theories, normal forms
2. DPLL+CDCL SAT solving
3. Eager SMT-solving: Equality logic with uninterpreted functions
   - From UF to EQ: Ackermann's reduction
   - From EQ to SAT: The Sparse method
4. Eager SMT solving: Finite-precision bit-vector arithmetic

Abstract grammar:

$$
\begin{array}{lcl}
\texttt{formula} & ::= & \texttt{formula} \vee \texttt{formula} \quad | \quad \neg\texttt{formula} \quad | \quad \texttt{atom} \\
\texttt{atom} & ::= & \texttt{boolId} \quad | \quad \texttt{term}[\texttt{constant}] \quad | \quad \texttt{term rel term} \\
\texttt{rel} & ::= & = \quad | \quad < \\
\texttt{term} & ::= & \texttt{constant} \quad | \quad \texttt{theoryId} \quad | \quad \sim \texttt{term} \quad | \\
 & & \texttt{term op term} \quad | \quad \texttt{atom?term:term} \quad | \\
 & & \texttt{term}[\texttt{constant:constant}] \quad | \quad \texttt{ext(term)} \\
\texttt{op} & ::= & + \quad | \quad - \quad | \quad \cdot \quad | \quad / \quad | \\
 & & << \quad | \quad >> \quad | \quad \& \quad | \quad \vert \quad | \quad \oplus \quad | \quad \circ
\end{array}
$$

$\sim x$	: bit-wise negation of x	$ext(x)$:	sign- or zero-extension of x
$x << d$:	left-shift with distance d	$x \circ y$:	concatenation of x and y

Finite-precision bit-vector arithmetic: Semantics

First for variables and constants:

Definition (Bit-vector)

A bit-vector x of length ℓ is a function

$$x : \{0, \ldots, \ell - 1\} \to \{0, 1\} \ .$$

Notation: x_i for $x(i)$, and graphically:

$x_{\ell-1}$	$x_{\ell-2}$	$\cdots$	x_2	x_1	x_0

Binary encoding: $[\![x_{[\ell]U}]\!] := \sum_{i=0}^{\ell-1} x_i \cdot 2^i$

Two's complement: $[\![x_{[\ell]S}]\!] := -2^{\ell-1} \cdot x_{\ell-1} + \sum_{i=0}^{\ell-2} x_i \cdot 2^i$

Notation: e.g. $x_{[32]S}$

Finite-precision bit-vector arithmetic: Semantics

- Arithmetic expressions: e.g.

$$[\![a_{[\ell]U} \ +_{[\ell]U} \ b_{[\ell]U}]\!] \ = \ ([\![a_{[\ell]U}]\!] \ + \ [\![b_{[\ell]U}]\!]) \bmod 2^{\ell}$$

- Relational operators: e.g.

$$[\![a_{[\ell]U} \ < \ b_{[\ell]U}]\!] = \mathit{true} \iff [\![a_{[\ell]U}]\!] \ < \ [\![b_{[\ell]U}]\!]$$

- Logical bit-wise operators: using λ-terms, e.g. for bit-wise or:

$$bv\_or \ := \ \lambda x.\ \lambda y.\ \lambda i \in \{0, \ldots, \ell - 1\}.\ x_i \vee y_i$$

Finite-precision bit-vector arithmetic: Propositional encoding

Propositional skeleton + constraints for meaning of sub-expressions

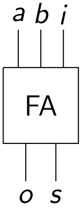
- Arithmetic operators: modulo 2^ℓ computations!
  $\rightsquigarrow$ Boolean circuit model + Tseitin (see next slide)

- Relational operators: exercise.

- Logical bit-wise operators:

$$a \,|_{[\ell]}\, b \text{ with } \mu(a \,|_{[\ell]}\, b)_i = c_i \qquad \rightsquigarrow \qquad \bigwedge_{i=0}^{\ell-1} \big(c_i \Leftrightarrow (a_i \vee b_i)\big)$$

Encoding $a + b$ for bits



Full adder:

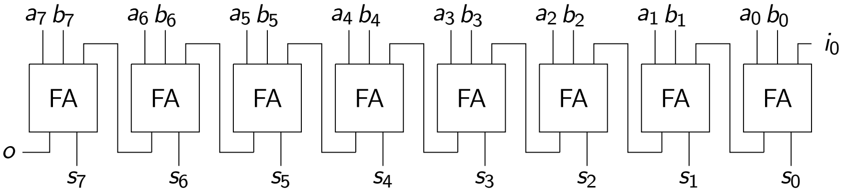
$$o \equiv (a+b+i)\ div\ 2 \equiv (a \wedge b) \vee (a \wedge i) \vee (b \wedge i)$$
$$s \equiv (a+b+i)\ mod\ 2 \equiv a \oplus b \oplus i$$

$$\begin{array}{ll} o: & (a \vee b \vee \neg o) \wedge (a \vee \neg b \vee i \vee \neg o) \wedge (a \vee \neg b \vee \neg i \vee o) \wedge \\ & (\neg a \vee b \vee i \vee \neg o) \wedge (\neg a \vee b \vee \neg i \vee o) \wedge (\neg a \vee \neg b \vee o) \\ s: & (a \vee b \vee i \vee \neg s) \wedge (a \vee b \vee \neg i \vee s) \wedge (a \vee \neg b \vee i \vee s) \wedge \\ & (a \vee b \vee \neg i \vee \neg s) \wedge (\neg a \vee b \vee i \vee s) \wedge (\neg a \vee b \vee \neg i \vee \neg s) \wedge \\ & (\neg a \vee \neg b \vee i \vee \neg s) \wedge (\neg a \vee \neg b \vee \neg i \vee s) \end{array}$$

Number of clauses: $6 + 8 = 14$

Encoding $a + b$ for bit-vectors

Carry chain adder:



Adds 2ℓ variables and 14ℓ clauses.