SMT (Satisfiability Modulo Theory)

- **Boolean SAT**
  - 60-second crash course
    - \[ A \land (\neg B \lor C) \land \neg D \rightarrow \vdash \]
    - \[ (\neg B \lor C) \land (B \lor \neg D) \land (D \lor \neg A) \land (A \lor B \lor D) \]

- **Theories**
  - Is the following formula satisfiable?
    - \[ x \geq 0 \land y \geq 0 \land 2 \cdot x \leq 10 \land |x - y| > 20 \land 4 \cdot x + 2 \cdot y = 47 \]
  - A theory is:
    - A vocabulary of (first-order) symbols
    - Designated interpretations for some or all of the symbols in the vocabulary

- **Theory of QF_UF**
  - Quantifier-Free Uninterpreted Functions
    - Consists of: arbitrary function symbols, equality relation \( = \)
    - \( = \) must be interpreted as... equality.
      - Satisfies equality axioms:
        \[ \forall x, y. x = y \rightarrow f(x) = f(y) \] (for all functions, of any arity)
  - QF_UF is also known as UF, EUF, and \( T \)
QF_UF Solver

• **Input**: a set \( \varphi \) of *equalities* and *disequalities* in QF_UF.

• **Output**: A model \( M \models \varphi \), or **UNSAT** if none exists.

\[
g(a) = c \\
g(a) = f(b) \\
h(g(a)) = h(c) \\
a \neq b \\
a \neq c \\
b \neq c \\
g(a) = c \\
g(a) = f(b) \\
h(g(a)) = h(c) \\
f(c) \neq f(f(b))
\]

6

QF_UF Solver

• **Algorithm**.
  ▪ Collect all the terms occurring in \( \varphi \).
  ▪ Construct the congruence relation \( t_1 \sim t_2 \):
    start with the input *equalities* and compute a closure w.r.t. equality axioms.
  ▪ Create a domain with one element per congruence class.
    Set the interpretations of terms according to which class contains them.
  ▪ Check the obtained model against the *disequalities*.
    - All hold: return model.
    - Otherwise: **UNSAT**.

7

DPLL(T) = SAT + Theory

• **Input**: a Boolean combination of propositional variables and literals from the theory.

• **Output**: (same as before) model or **UNSAT**.

• **General idea**.
  ▪ Treat literals as opaque propositions and try to find a SAT assignment.
  ▪ Query theory solver to check if this assignment is consistent with the theory.
    Theory solver should return a new (“learned”) literal in case of **UNSAT**.

8

Example: DPLL(QF_UF)

\[
g(a) = c \land (f(g(a)) \neq f(c) \lor g(a) = d) \land c \neq d \\
A \land (\neg B \lor C) \land \neg D \rightarrow \text{SAT} \\
g(a) = c \land f(g(a)) \neq f(c) \land g(a) = d \land c \neq d \\
\text{(QF_UF)} \rightarrow \text{UNSAT} \\
g(a) = c \rightarrow f(g(a)) = f(c) \\
A \lor B
\]

9

10
Theory of QF_LIA

• Quantifier-Free Linear Integer Arithmetic
  ▸ Consists of: +, −, multiplication by constant, equality (=), inequalities (≠, <, >, ≤, ≥).
  ▸ Designated interpretation: standard integer domain. Standard arithmetic and comparison operators.
• Turns out — QF_LIA is decidable.
  ▸ (we will skip the algorithm; ☺︎)

Quantiﬁers

• Our two main weapons:

  Skolemization
  basically gets rid of existentials

  ∀x. ∃y. Q(x, y) × P(y)
  ≡

  ∀x. Q(x, f(x)) ∧ P(f(x))
(+ compactness)

Herbrand’s Theorem
  instructs us to instantiate universals

  ∀x. φ(x)
  ≡

  φ(f_1), φ(f_2), φ(f_3), ...

Why should we even CARE?

Proof by Refutation

• Task: Prove that a formula φ is valid.

  ⊨ φ

• Alternatively:
  Prove that ¬φ is unsatisﬁable.

  ¬∃M. M ⊨ ¬φ

Proving Things About Programs

• ...but in ﬁrst-order logic.
Proving Things About Programs

• ...but in first-order logic.

\[
\begin{align*}
\{x + 1 = y + 1\} x := x + 1 &\quad \{x = y + 1\} \\
\{x = y + 1\} x := x + 1 &\quad \{x = y + 1\}
\end{align*}
\]

A general algorithm that will do this systematically for any program written in WHILE-language

• Known more succinctly as “weakest precondition”

• Define a new function this time:
  
  \( wp[c]Q \)

  Maps a command \( c \) and a postcondition \( Q \) (an assertion) to the weakest precondition \( P \) that will make \( \{P\} c \{Q\} \) hold.

Weakest Precondition

\[
\begin{align*}
wp[skip]Q & = Q \\
wp[x := e]Q & = Q[e/x] \\
wp[c_1 ; c_2]Q & = wp[c_1][wp[c_2]Q] \\
wp[if b then c_1 else c_2]Q & = \\
(b \land wp[c_1]Q) & \lor \\
(-b \land wp[c_2]Q)
\end{align*}
\]

Weakest Precondition

\[
\begin{align*}
wp[\text{while } b \text{ do } c]Q & = \ ? \\
wp[P \text{ while } b \text{ do } c]Q & = \\
P \land \forall v. (P \land b \rightarrow wp[c]P) & \land \\
(P \land \neg b \rightarrow Q)
\end{align*}
\]

Weakest Precondition

\[
\begin{align*}
wp[i := 0 ; (i \leq n) \text{ while } i < n \text{ do } i := i + 1][i=n] & = \\
(wp[i := 0][i \leq n] \land wp[(i \leq n) \text{ while } i < n \text{ do } i := i + 1][i=n]) & = \\
wp[i := 0][i \leq n] & \land \\
(i \leq n & \land (\neg(i \leq n) \rightarrow i=n))
\end{align*}
\]

\[
\begin{align*}
wp[P \text{ while } b \text{ do } c]Q & = \\
P \land \forall v. (P \land b \rightarrow wp[c]P) & \land \\
(P \land \neg b \rightarrow Q)
\end{align*}
\]
Now it’s almost too easy.

\[ \{(P) \land (Q)\} \models \Phi \implies P \rightarrow wp[s]Q \]

\( (wp[s]Q) \subseteq \{(P) \land (Q)\} \)

Next week

We will do some synthesis.

(yum.)

Lab #7

Row-wise color (analogous for columns)

Color of box \( i \)

\( \oplus \)

\( (s_k \leq i) \)

Number of prefix sums that are \( \leq \) the index

\( a_1 \)

\( a_2 \)

\( a_3 \)

\( a_4 \)

Row-wise color (analogous for columns)

Color of box \( i \)

\( \oplus \)

\( (s_k \leq i) \)

\( (a_i \neq \emptyset) \)

(a different depiction of the same principle)