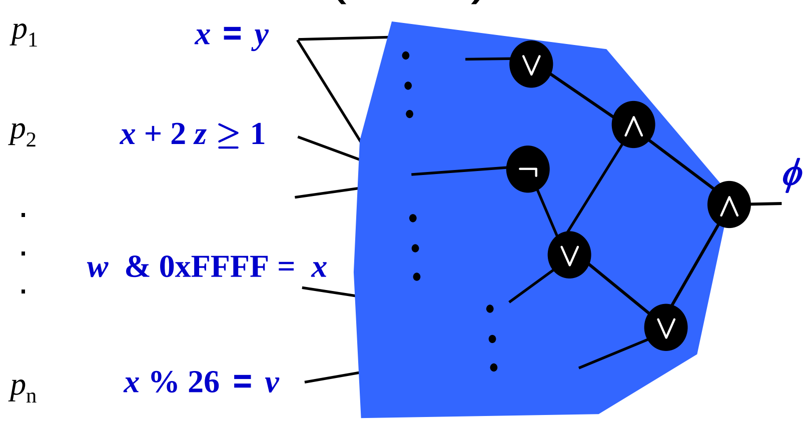
EECS 219C: Computer-Aided Verification

Satisfiability Modulo Theories

Introduction, Survey of Theories

Sanjit A. Seshia
EECS, UC Berkeley

Satisfiability Modulo Theories (SMT)



Is there an assignment to the *x,y,z,w* variables s.t. ϕ evaluates to 1?

Satisfiability Modulo Theories

- Given a formula in first-order logic, with associated background theories, is the formula satisfiable?
  - If SAT, return a satisfying assignment (a 'model')
  - If UNSAT, optionally generate a proof of unsatisfiability
- A hot topic of research in the last ~10 yrs
  - SMTLIB, SMTCOMP

Reference

Satisfiability Modulo Theories

Clark Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli.

Chapter 8 in the Handbook of Satisfiability, Armin Biere, Hans van Maaren, and Toby Walsh, editors, IOS Press, 2009.

(available from my webpage)

First-Order Logic

- A formal notation for mathematics, with expressions involving
  - Propositional symbols
  - Predicates
  - Functions and constant symbols
  - Quantifiers
- In contrast, propositional (Boolean) logic only involves propositional symbols and operators

First-Order Logic: Syntax

- As with propositional logic, expressions in first-order logic are made up of sequences of symbols.
- Symbols are divided into *logical symbols* and *non-logical symbols* or *parameters*.
- Example:

$$(x = y) \wedge (y = z) \wedge (f(z) \geq f(x)+1)$$

First-Order Logic: Syntax

- Logical Symbols
  - Parentheses: (,)
  - Propositional connectives: $\vee$, $\wedge$, $\neg$, $\rightarrow$, $\leftrightarrow$
  - Variables: v1, v2, . . .
  - Quantifiers: $\forall$, $\exists$
- Non-logical symbols/Parameters
  - Equality: =
  - Functions: +, -, %, bit-wise &, f(), concat, …
  - Predicates: $\leq$, is_substring, …
  - Constant symbols: 0, 1.0, null, …

Quantifier-free Subset

- We will largely restrict ourselves to formulas without quantifiers ($\forall$, $\exists$)
- This is called the quantifier-free subset/fragment of FOL with the relevant theory

Logical Theory

- Defines a set of parameters (non-logical symbols) and their meanings
- This definition is called a *signature*.
- Example of a signature:

  Theory of linear arithmetic over integers

  Signature is $(0,1,+,-,\leq)$ interpreted over $\mathbb{Z}$

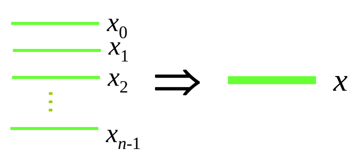
Common Theories

- Equality (with uninterpreted functions)
- Finite-precision bit-vectors – integers or floating-point
- Difference logic (over $\mathbb{Q}$ or $\mathbb{Z}$)
- Linear arithmetic (over $\mathbb{Q}$ or $\mathbb{Z}$)
- Arrays / memories
- Misc.: Non-linear arithmetic, strings, inductive datatypes (e.g. lists), sets, …

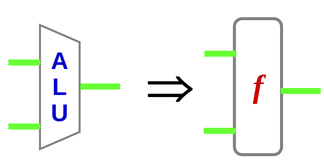
Theory of Equality and Uninterpreted Functions (EUF)

- Also called the “free theory”
  - Because function symbols can take any meaning. The only property required is that these symbols map identical arguments to identical values i.e., $x = y \Rightarrow f(x) = f(y)$
  - Also need the properties the equality operator satisfies
- SMTLIB name: QF_UF

Data and Function Abstraction with EUF



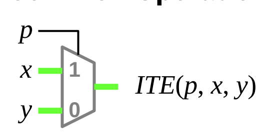
Bit-vectors to (unbounded) Integers



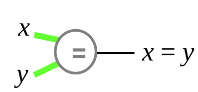
Functional units to Uninterpreted Functions

$a = x \wedge b = y \Rightarrow f(a,b) = f(x,y)$

Common Operations

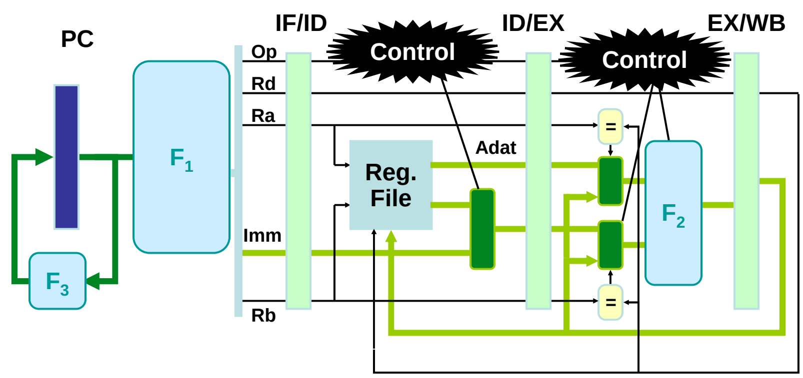


If-then-else



Test for equality

Hardware Abstraction with EUF



- For any Block that Transforms or Evaluates Data:
  - Replace with generic, unspecified function
  - Also view instruction memory as function

Example QF_UF (EUF) Formula

$$(x = y) \wedge (y = z) \wedge (f(x) \neq f(z))$$

Transitivity:

$$(x = y) \wedge (y = z) \Rightarrow (x = z)$$

Congruence:

$$(x = z) \Rightarrow (f(x) = f(z))$$

Equivalence Checking of Program Fragments

```
int fun1(int y) {
    int x, z;
    z = y;
    y = x;
    x = z;

   return x*x;
}

int fun2(int y) {
     return y*y;
}
```

SMT formula ϕ
Satisfiable iff programs non-equivalent

$(z = y \wedge y1 = x \wedge x1 = z \wedge ret1 = x1*x1)$
$\wedge$
$(ret2 = y*y)$
$\wedge$
$(ret1 \neq ret2)$

What if we use SAT to check equivalence?

S. A. Seshia 15

Equivalence Checking of Program Fragments

```
int fun1(int y) {
    int x, z;
    z = y;
    y = x;
    x = z;

   return x*x;
}

int fun2(int y) {
     return y*y;
}
```

SMT formula ϕ
Satisfiable iff programs non-equivalent

$(z = y \wedge y1 = x \wedge x1 = z \wedge ret1 = x1*x1)$
$\wedge$
$(ret2 = y*y)$
$\wedge$
$(ret1 \neq ret2)$

Using SAT to check equivalence (w/ Minisat)
32 bits for y: Did not finish in over 5 hours
16 bits for y: 37 sec.
8 bits for y: 0.5 sec.

S. A. Seshia 16

Equivalence Checking of Program Fragments

```
int fun1(int y) {
    int x, z;
    z = y;
    y = x;
    x = z;

   return x*x;
}

int fun2(int y) {
     return y*y;
}
```

SMT formula ϕ'

$$(z = y \wedge y1 = x \wedge x1 = z \wedge ret1 = sq(x1))$$
$$\wedge$$
$$(ret2 = sq(y))$$
$$\wedge$$
$$(ret1 \neq ret2)$$

Using EUF solver: 0.01 sec

Equivalence Checking of Program Fragments

```
int fun1(int y) {
    int x;
    x = x ^ y;
    y = x ^ y;
    x = x ^ y;

   return x*x;
}

int fun2(int y) {
     return y*y;
}
```

Does EUF still work?

No!
Must reason about bit-wise XOR.

Need a solver for bit-vector arithmetic.

Solvable in less than a sec. with a current bit-vector solver.

Finite-Precision Bit-Vector Arithmetic (QF_BV)

- Fixed width data words
  - Can model int, short, long, etc.
- Arithmetic operations
  - E.g., add/subtract/multiply/divide & comparisons
  - Two's complement and unsigned operations
- Bit-wise logical operations
  - E.g., and/or/xor, shift/extract and equality
- Boolean connectives

Linear Arithmetic (QF_LRA, QF_LIA)

- Boolean combination of linear constraints of the form

  $$(a_1\, x_1 + a_2\, x_2 + \ldots + a_n\, x_n \sim b)$$

- x_i's could be in $\mathbb{Q}$ or $\mathbb{Z}$, $\sim\ \in\ \{\geq,>,\leq,<,=\}$

- Many applications, including:
  - Verification of analog circuits
  - Software verification, e.g., of array bounds

Difference Logic (QF_IDL, QF_RDL)

- Boolean combination of linear constraints of the form

$$x_i \geq x_j + c_{ij}$$

or $x_i \geq c_j$

- Applications:
  - Software verification (most linear constraints are of this form)
  - Processor datapath verification
  - Job shop scheduling / real-time systems
  - Timing verification for circuits

Scheduling Jobs

- n jobs, with execution times $T_1, T_2, \ldots, T_n$
- Let $s_1, s_2, \ldots, s_n$ be the start times, $f_1, f_2, \ldots, f_n$ the finish times
- No pre-emption
- Some jobs need the same resource (cannot execute simultaneously)
- Need to finish all jobs before time t=T, starting at t=0

Scheduling Jobs

- n jobs, with execution times $T_1, T_2, \ldots, T_n$
- Let $s_1, s_2, \ldots, s_n$ be the start times, $f_1, f_2, \ldots, f_n$ the finish times
- No pre-emption
- Some jobs need the same resource (cannot execute simultaneously)

Formulation in QF_IDL: A conjunction of the following:

- $f_i = s_i + T_i$
- If jobs i and j share the same resource, then
  $(s_i \geq f_j)$ or $(s_j \geq f_i)$
- $\max_i f_i < T$

S. A. Seshia 23

Linear Arithmetic (QF_LRA, QF_LIA)

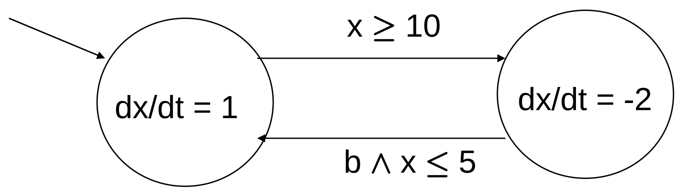
- Boolean combination of linear constraints of the form

  $$(a_1 x_1 + a_2 x_2 + \ldots + a_n x_n \geq b)$$

- x_i's could be in $\mathbb{Q}$ or $\mathbb{Z}$

S. A. Seshia 24

Modeling and Verifying Hybrid Systems with QF_LRA



Can we write an SMT formula to check whether x can ever be < 0?

Arrays/Memories

- SMT solvers can also be very effective in modeling data structures in software and hardware
  – Arrays in programs
  – Memories in hardware designs: e.g. instruction and data memories, CAMs, etc.

Theory of Arrays (QF_AX) Select and Store

- Two interpreted functions: select and store
  - select(A,i)
  - store(A,i,d)
- Two main axioms:
  - select(store(A,i,d), i) = d
  - select(store(A,i,d), j) = select(A,j) for $i \neq j$
- One other axiom: “extensionality”
  - $(\forall$ i. select(A,i) = select(B,i)$) \Rightarrow$ A = B

Equivalence Checking of Program Fragments

```
int fun1(int y) {
   int x[2];
   x[0] = y;
   y = x[1];
   x[1] = x[0];

  return x[1]*x[1];
}
```

```
int fun2(int y) {
     return y*y;
}
```

How can we express the equivalence checking problem as an SMT formula with arrays?

Equivalence Checking of Program Fragments

```
int fun1(int y) {
    int x[2];
    x[0] = y;
    y = x[1];
    x[1] = x[0];

   return x[1]*x[1];
}

int fun2(int y) {
      return y*y;
}
```

SMT formula ϕ''

$[\ x1 = store(x,0,y) \wedge y1 = select(x1,1)$
$\wedge\ x2 = store(x1,1,select(x1,0))$
$\wedge\ ret1 = sq(select(x2,1))\ ]$
$\wedge$
$(\ ret2 = sq(y)\ )$
$\wedge$
$(\ ret1 \neq ret2\ )$