SAT/SMT Solvers and Axiomatic Semantics
Announcements

- Presentations start on Thursday
  - A few of you still haven’t picked slots

- Attend and participate!
  - Will count towards your participation credit

- HW7 is up. It is optional

- Quiz 6
Outline

- SAT/SMT solvers
- Axiomatic semantics
  - IMP
  - Semantics
  - Verification condition generation
    - Essentially, what we called “backwards reasoning” in Principles of Software
- SMT-LIB
Logical Reasoning

- A lot of recent PL/SE research uses some form of automated logical reasoning
  - Software verification (e.g., Dafny uses SMT)
  - Symbolic execution (SMT)
  - Program synthesis
If you are interested

SAT Solvers

- Decide whether a propositional logic formula is satisfiable (sat) or unsatisfiable (unsat)
  - E.g., \((p \lor q) \rightarrow \lnot p\) is sat or unsat?
  - E.g., \((p \rightarrow q) \rightarrow \lnot(p \land \lnot q)\) is sat or unsat?

- A lot of work on SAT solvers
  - Boolean satisfiability is a fundamental NP-complete problem
  - A good SAT solver can “solve” many problems!!!
Variations of SAT

- **MaxSAT**: Given a formula in **Conjunctive Normal Form (CNF)**, find an assignment that maximizes number of satisfied clauses
  - E.g., \((p \lor q) \land !p \land !q\)

- **Partial MaxSAT**
  - **Hard clauses**: clauses that must be satisfied
  - **Soft clauses**: clauses that may remain unsatisfied
  - Partial MaxSAT: find an assignment that satisfies all hard clauses and maximizes number of satisfied soft clauses
Variations of SAT

- Weighted Partial MaxSAT
  - **Hard clauses**: clauses that must be satisfied
  - **Soft clauses**: clauses that may remain unsatisfied
  - **Weights**: soft clauses have weights
  - Weighted Partial MaxSAT: find an assignment that satisfies all hard clauses and maximizes the weight of satisfied soft clauses
    - E.g., suppose \((p \lor q)\) is a hard clause, \(!p\) is a soft clause with weight 2, and \(!q\) is soft with weight 1
    - What assignment maximizes \((p \lor q) \land !p \land !q\)
SMT Solvers

- **Satisfiability Modulo Theories** extends assertions/satisfiability beyond propositional logic

- Extends with **background theories**
  - Theory of equality: $x \neq y \land f(x) = f(y)$
  - Theory of arithmetic: $x < y \land \neg(x < y + 0)$
  - Theory of select/store (arrays): Hoare triple
    \[
    \{ \text{b.f = 5} \} \ a.f = 5 \{ \ a.f + b.f = 10 \} \]
    leads to formula
    $$\text{select}(f,b) = 5 \land \text{store}(f,a,5) \Rightarrow \text{select}(f,b) + \text{select}(f,a) = 10$$
SMT Solvers

- Examples
  - \((z>5 \land x>0) \lor (z<-5 \land x\leq0)\)
  - \((x>0 \land x+5>5) \lor (x\leq0 \land (x=0 \Rightarrow x+5+x=5))\)

- Lots of SMT solvers, e.g., Z3
  - My goal: become somewhat competent users of SMT solvers; be able to encode problems
Outline

- SAT/SMT solvers
- Axiomatic semantics
  - IMP
  - Semantics
  - Verification condition generation
    - Essentially, what we called “backwards reasoning” in Principles of Software
- SMT-LIB
Axiomatic Semantics

Consider program fragment

\[ t = x - y; \]
\[ \text{while} \ (t > 0) \ { \}
\[ \quad x = x - 1; \]
\[ \quad y = y + 1; \]
\[ \quad t = t - 1; \]
\[ \} \]

We are interested in proving these claims:

- When \( x > y \), program terminates
- When \( x > y \), values of \( x \) and \( y \) are swapped
Axiomatic Semantics

Not easy to prove using theories we studied so far
- Dataflow
- Abstract interpretation
- Types

E.g., neither gives a convenient way of encoding the assumption $x > y$ into reasoning and semantics
Axiomatic Semantics

Key idea:

- \{ P \} code \{ Q \}

- Semantics of a program construct is defined in terms of logical assertions and the effect of the construct on these assertions
History

  - Great optimism by Tony Hoare and Edsger Dijkstra
  - Bugs will be a thing of the past!
  - If you can prove programs correct, no need to even test!

- Middle ages: “Social Processes and Proofs of Theorems and Programs”, De Millo, Lipton and Perlis, 1979
  - Proofs in math work because there is a social process
  - Program proofs are too boring for social process to form
  - Programs change too fast and proofs are too brittle

- A renaissance: Z3, other automated logical reasoning tools
  - Some success stories from Microsoft
  - There is some optimism again…
Hoare triples \{ P \} stmt \{ Q \}

- \( P \) is the precondition, \( Q \) is the postcondition
- Triple is a logical formula: if \( P \) holds before \( \text{stmt} \) execution and \( \text{stmt} \) terminates, then \( Q \) holds afterwards
- E.g., \( \{ x > -1/2 \} \ x = x + 3 \ \{ x > 5/2 \} \)

- \( \{ P \} \ \text{stmt} \ \{ Q \} \): partial correctness assertion
- \( [ P ] \ \text{stmt} [ Q ] \): total correctness assertion

We will concern with partial correctness only
The IMP Language

Expressions
\[ e ::= n \mid x \mid e_1 + e_2 \mid e_1 = e_2 \]

Commands (i.e., statements, change state):
\[ c ::= x := e \mid c_1 ; c_2 \mid \text{if (e) then } c_1 \text{ else } c_2 \mid \text{while (e) do } c \mid \text{skip} \]

A big-step operational semantics
- Judgments for expressions: \((e, \sigma) \rightarrow n\)
- Judgments for commands: \((c, \sigma) \rightarrow \sigma'\)
Operational Semantics

\[(e, \sigma) \rightarrow n\]

\[(x := e, \sigma) \rightarrow \sigma[x \leftarrow n]\]

\[(e, \sigma) \rightarrow \text{True}\]

\[(c_1, \sigma) \rightarrow \sigma'\]

\[(\text{if } (e) \text{ then } c_1 \text{ else } c_2, \sigma) \rightarrow \sigma'\]

\[(e, \sigma) \rightarrow \text{False}\]

\[(c_2, \sigma') \rightarrow \sigma''\]

\[(c_1; c_2, \sigma) \rightarrow \sigma''\]

\[(c, \sigma) \rightarrow \sigma'\]

\[(\text{while } (e) \text{ do } c, \sigma') \rightarrow \sigma''\]

\[(\text{while } (e) \text{ do } c, \sigma) \rightarrow \sigma''\]

Program Analysis CSCI 4450/6450, A Milanova (based on MIT Program Analysis OCW)
Meaning of Assertions

\[ \{ P \} \ c \ \{ Q \} \]

- Let \( P \) be a logical assertion
  - E.g., \( x < y \) or \( x + y = 5 \)
  - \( P \) implicitly references state \( \sigma \)

- \( \sigma \vdash P \) (read: \( \sigma \) entails \( P \)) means that assertion \( P \) holds on state \( \sigma \)
  - E.g., \( \sigma = [x \mapsto 5, y \mapsto 10, z \mapsto 0] \vdash x < y \)
  - Does \( \sigma' = [x \mapsto 10, y \mapsto 10, z \mapsto 0] \vdash x < y \) ?

- Partial correctness \( \{ P \} \ c \ \{ Q \} \) therefore is
  - \( \forall \sigma, \forall \sigma'. \ (\sigma \vdash P \land (c, \sigma) \rightarrow \sigma') \Rightarrow \sigma' \vdash Q \)
Static Semantics

\[
\{ P[e/x] \} \ x := e \ { P } \\
\{ P \} \ c_1 \{ Q \} \quad \{ Q \} \ c_2 \{ R \} \\
\{ P \} \ c_1 ; c_2 \{ R \}
\]

\[
\{ P \land e \} \ c_1 \{ Q \} \quad \{ P \land !e \} \ c_2 \{ Q \}
\]

\[
\{ P \} \text{ if (e) then } c_1 \text{ else } c_2 \{ Q \}
\]

\[
\{ P \land e \} \ c \{ P \}
\]

\[
\{ P \} \text{ while (e) do } c \{ P \land !e \}
\]

Rule of consequence:

\[
P \Rightarrow P' \quad \{ P' \} \ c \{ Q' \} \quad Q' \Rightarrow Q
\]

\[
\{ P \} \ c \{ Q \}
\]

Program Analysis CSCI 4450/6450, A Milanova (based on MIT Program Analysis OCW)
Soundness

- For each Hoare triple \( \{ P \} c \{ Q \} \) deduced by the static semantics \( \{ P \} c \{ Q \} \) true iff
  \( \forall \sigma, \forall \sigma'. (\sigma |- P \land (c, \sigma) \rightarrow \sigma') \Rightarrow \sigma' |- Q \)
  holds

- Notice how in each one of our theories, AI, types, AS we have
  - Dynamic semantics
  - Static semantics
  - Soundness (connecting the two)
Example

\{ \ x > y \ \text{and} \ x = x_0 \ \text{and} \ y = y_0 \ \}

\text{t = x – y;}

\text{while (t > 0) \{}

\text{\quad x = x – 1;}

\text{\quad y = y + 1;}

\text{\quad t = t – 1;}

\text{\}}

\{ \ x = y_0 \ \text{and} \ y = x_0 \ \}
Example

\{ x>y \text{ and } x=x_0 \text{ and } y=y_0 \} \\
t = x - y; \\

while \ (t > 0) \ 
\{ \\
\qquad x = x - 1; \\
\qquad y = y + 1; \\
\qquad t = t - 1; \\
\}

\{ x=y_0+t \text{ and } y=x_0-t \text{ and } t\geq 0 \text{ and } !(t>0) \} \Rightarrow \{ x = y_0 \text{ and } y = x_0 \}
Example

\[ \{ x \geq y \text{ and } x = x_0 \text{ and } y = y_0 \} \]

\[ t = x - y; \]

while \((t > 0)\) \{

\[ x = x - 1; \]

\[ y = y + 1; \]

\[ t = t - 1; \]

\( \{ x = y_0 + t \text{ and } y = x_0 - t \text{ and } t \geq 0 \} \]

\}

\[ \{ x = y_0 + t \text{ and } y = x_0 - t \text{ and } t \geq 0 \text{ and } !(t > 0) \} \implies \{ x = y_0 \text{ and } y = x_0 \} \]
Example

\{ x>y \text{ and } x=x_0 \text{ and } y=y_0 \} \\
t = x - y;

\text{while} (t > 0) \{ \\
\{ x-1=y_0+t-1 \text{ and } y+1=x_0-t+1 \text{ and } t-1 \geq 0 \} \text{ simpl. } \{ x=y_0+t \text{ and } y=x_0-t \text{ and } t-1 \geq 0 \} \\
\quad x = x - 1; \\
\{ x=y_0+t-1 \text{ and } y+1=x_0-t+1 \text{ and } t-1 \geq 0 \} \\
\quad y = y + 1; \\
\{ x=y_0+t-1 \text{ and } y=x_0-t+1 \text{ and } t-1 \geq 0 \} \\
\quad t = t - 1; \\
\{ x=y_0+t \text{ and } y=x_0-t \text{ and } t \geq 0 \} \\
\}

\{ x=y_0+t \text{ and } y=x_0-t \text{ and } t \geq 0 \text{ and } !(t>0) \} \Rightarrow \{ x = y_0 \text{ and } y = x_0 \}
Example

\[\{ x > y \text{ and } x = x_0 \text{ and } y = y_0 \}\]

\[t = x - y;\]

\[\text{while } (t > 0) \{\]

\[\{ x = y_0 + t \text{ and } y = x_0 - t \text{ and } t \geq 0 \text{ and } t > 0 \} \Rightarrow \{ x = y_0 + t \text{ and } y = x_0 - t \text{ and } t - 1 \geq 0 \}\]

\[x = x - 1;\]

\[\{ x = y_0 + t - 1 \text{ and } y + 1 = x_0 - t + 1 \text{ and } t - 1 \geq 0 \}\]

\[y = y + 1;\]

\[\{ x = y_0 + t - 1 \text{ and } y = x_0 - t + 1 \text{ and } t - 1 \geq 0 \}\]

\[t = t - 1;\]

\[\{ x = y_0 + t \text{ and } y = x_0 - t \text{ and } t \geq 0 \}\]

\}\]

\[\{ x = y_0 + t \text{ and } y = x_0 - t \text{ and } t \geq 0 \text{ and } !(t > 0) \} \Rightarrow \{ x = y_0 \text{ and } y = x_0 \}\]
Example

{ x>y and x=x0 and y=y0 }

\[ t = x - y; \]

{ x=y0+t and y=x0-t and t\geq0 }

while (t > 0) {
    { x=y0+t and y=x0-t and t\geq0 and t>0 } \Rightarrow \{ x=y0+t and y=x0-t and t-1\geq0 \}
        x = x - 1;
    
    { x=y0+t-1 and y+1=x0-t+1 and t-1\geq0 }
        y = y + 1;
    
    { x=y0+t-1 and y=x0-t+1 and t-1\geq0 }
        t = t - 1;
    
    { x=y0+t and y=x0-t and t\geq0 }
}

{ x=y0+t and y=x0-t and t\geq0 and !(t>0) } \Rightarrow \{ x = y0 and y = x0 \}
Example

\{ x>y \text{ and } x=x_0 \text{ and } y=y_0 \} \Rightarrow \{ x=y_0+x-y \text{ and } y=x_0-x+y \text{ and } x-y \geq 0 \}

t = x - y;

\{ x=y_0+t \text{ and } y=x_0-t \text{ and } t \geq 0 \}

while \ (t > 0) \ \{
\{ x=y_0+t \text{ and } y=x_0-t \text{ and } t \geq 0 \text{ and } t>0 \} \Rightarrow \{ x=y_0+t \text{ and } y=x_0-t \text{ and } t-1 \geq 0 \}

\quad x = x - 1;

\{ x=y_0+t-1 \text{ and } y+1=x_0-t+1 \text{ and } t-1 \geq 0 \}

\quad y = y + 1;

\{ x=y_0+t-1 \text{ and } y=x_0-t+1 \text{ and } t-1 \geq 0 \}

\quad t = t - 1;

\{ x=y_0+t \text{ and } y=x_0-t \text{ and } t \geq 0 \}

\}

\{ x=y_0+t \text{ and } y=x_0-t \text{ and } t \geq 0 \text{ and } !(t>0) \} \Rightarrow \{ x = y_0 \text{ and } y = x_0 \}
Outline

- SAT/SMT solvers

- Axiomatic semantics
  - IMP
  - Semantics
  - Verification condition generation
    - Essentially, what we called “backwards reasoning” in Principles of Software

- SMT-LIB
Weakest Precondition

- \(wp(x := e, Q) = Q[e/x]\)
- \(wp(c_1; c_2, Q) = wp(c_1, wp(c_2, Q))\)
- \(wp(\text{if } (e) \text{ then } c_1 \text{ else } c_2, Q) = \)
  \[(e \Rightarrow wp(c_1, Q)) \land (!e \Rightarrow wp(c_2, Q))\]
- \(wp(\text{while } (e) \text{ do } c, Q) = \)
  \[W = (e \Rightarrow wp(c, W)) \land (!e \Rightarrow Q)\]
Verification Condition

Instead of weakest precondition we compute verification condition ($vc$). Stronger

$$vc(\text{while (e) do c, Q}) =$$

\[
\text{Inv} \land (\text{Inv} \Rightarrow ((e \Rightarrow vc(c,\text{Inv})) \land (!e \Rightarrow Q)))
\]

Or

$$vc(\text{while (e) do c, Q}) =$$

\[
\text{Inv} \land // \text{Must hold before loop!}
\]

\[
(\text{Inv} \land e) \Rightarrow vc(c,\text{Inv}) \land // \text{Must hold locally for loop}
\]

\[
(\text{Inv} \land !e) \Rightarrow Q
\]
Example

```c
i = 5;
while (i > 0) {
    Inv = { i ≥ 0 }
    i = i - 1;
}
{ i = 0 }
```

vc( while (i>0) { i = i-1; }, {i=0})

vc breaks into following assertions:

True => wp(i=5; { i≥0 })

i≥0 ∧

i≥0∧i>0 => wp(i=i-1,{i≥0})

equiv. i≥0∧i>0 => i-1≥0 ∧

i≥0∧!(i>0) => i=0
Another Example

\{ x \geq 0 \} 

i = x;

z = 0;

while (i != 0) {
  z = z+1;
  i = i-1;
}

\{ x = z \}
SMT-LIB

SMT-LIB is a language for specifying input to SMT solvers (e.g., Z3)

- (declare-const x Int) declare an integer constant x
- (assert (> x 0)) add x>0 to known facts
- (check-sat) checks if there exist an assignment that makes all known facts true; returns (sat) or (unsat)
- (get-model) print this assignment

https://rise4fun.com/z3/tutorial
(declare-const a Int)
(declare-fun f (Int Bool) Int)
(assert (> a 10))
(assert (< (f a true) 100))
(check-sat)
Your homework is to write a Tiny Dafny

- Given a program $\{ P \} \mathsf{c} \{ Q \}$ generate verification conditions in SMT-LIB
- Verify conditions with Z3

Yet another programming language, OCaml
Suppose we need to verify \( \{ P \} c \{ Q \} \)

- Generate \( vc(c,Q) \)
- Program verifies when \( P \Rightarrow vc(c,Q) \) is **valid**
  - A logical formula is **valid** when true for all inputs

**Encoding**

- Duality of satisfiability and validity:
  - \( F \) is valid iff \( \neg F \) is unsatisfiable
- Ask: is \( \neg(P \Rightarrow vc(c,Q)) \) satisfiable
  - If (unsat) program is verified!
  - If (sat) get model
Example

requires: x == 1 || x == -2
ensures: y == 0
{
  y = x + 4;
  if (x > 0) {
    y = x*x - 1;
  } else {
    y = y + x;
  }
}

vc(...,{y=0}) = ??
(x=1 or x=-2) =>
((x>0 and x*x-1=0) or
 (x<=0 and x+4+x=0))

SMT-LIB code:
(declare-const x Int)
(assert (and
  (or (= x 1) (= x -2))
  (not
    (or (and (<= x 0) (= (+ (+ x 4) x) 0))
      (and (> x 0) (= (- (* x x) 1) 0)))))))
(check-sat)
(get-model)
Example

requires: \( x = 1 \) || \( x = -5 \)

ensures: \( y = 0 \)

\{
  y = x + 4;
  if (x > 0) {
    y = x*x - 1;
  } else {
    y = y + x;
  }
\}

vc(\ldots,\{y=0\}) = wp(\ldots,\{y=0\}) = ??

(x=1 or x=-5) =>
((x>0 and x*x-1 = 0) or
 (x<=0 and x+4+x=0))

SMT-LIB code:

(declare-const x Int)
(assert (and
  (or (= x 1) (= x -5))
  (not
    (or (and (<= x 0) (= (+ (+ x 4) x) 0))
      (and (> x 0) (= (- (* x x) 1) 0)) ))))
(check-sat)
(get-model)
Another Example

Is this formula valid?

\[(x > 0 \text{ and } x + 5 > 5) \text{ or } (x \leq 0 \text{ and } (x = 0 \Rightarrow x + x + 5 = 5))\]

SMT-LIB code:

```lisp
(declare-const x Int)
(assert (not (and (> x 0) (> (+ x 5) 5))))
(assert (not
  (and (<= x 0) (or (not (= x 0)) (= (+ (+ x x) 5) 5))))))
(check-sat)
```