SAT/SMT Solvers and Axiomatic Semantics
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

- Non-standard type inference (SAT, MaxSAT)
- 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., \((p \lor q) \rightarrow !p\) is sat or unsat?
  - E.g., \((p \rightarrow q) \rightarrow !(p \land !q)\) 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., \((p \lor q) \land \neg p \land \neg q\)

- **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: \( x < y \land !(x < y + 0) \)
  - 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

- \((z > 5 \land x > 0) \lor (z < -5 \land x \leq 0)\)
- \((x > 0 \land x + 5 > 5) \lor (x \leq 0 \land (x = 0 \implies x + 5 + x = 5))\)

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
- 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…
You Already Know The Basics

- Hoare triples \( \{ P \} \text{ stmt } \{ Q \} \)
  - \( P \) 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

<table>
<thead>
<tr>
<th>Rule</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>((e,\sigma) \rightarrow n)</td>
<td>((x:=e,\sigma) \rightarrow \sigma[x\leftarrow n])</td>
</tr>
<tr>
<td>((e,\sigma) \rightarrow \text{True})</td>
<td>((c_1,\sigma) \rightarrow \sigma')</td>
</tr>
<tr>
<td>((\text{if} \ (e) \ \text{then} \ c_1 \ \text{else} \ c_2, \ \sigma) \rightarrow \sigma')</td>
<td>((c_1 ; c_2, \ \sigma) \rightarrow \sigma'')</td>
</tr>
<tr>
<td>((e,\sigma) \rightarrow \text{False})</td>
<td>((c_2,\sigma) \rightarrow \sigma'')</td>
</tr>
<tr>
<td>((\text{while} \ (e) \ \text{do} \ c, \ \sigma') \rightarrow \sigma'')</td>
<td>((\text{while} \ (e) \ \text{do} \ c, \ \sigma') \rightarrow \sigma'')</td>
</tr>
<tr>
<td>((e,\sigma) \rightarrow \text{False})</td>
<td>((\text{while} \ (e) \ \text{do} \ c, \ \sigma) \rightarrow \sigma'')</td>
</tr>
</tbody>
</table>

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

\{ P \} c \{ Q \}

- Let \( P \) be 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., \( \sigma = [x \rightarrow 5, y \rightarrow 10, z \rightarrow 0] \vdash x < y \)
  - Does \( \sigma' = [x \rightarrow 10, y \rightarrow 10, z \rightarrow 0] \vdash x < y \)?

- Partial correctness \( \{ P \} c \{ Q \} \) therefore is
  - \( \forall \sigma, \forall \sigma'. (\sigma \vdash P \land (c, \sigma) \rightarrow \sigma') \Rightarrow \sigma' \vdash Q \)
\[
\{ P[e/x] \} \ x := e \ { \ P } \\
\{ P \} \ c_1 \ { \ Q } \quad \{ \ Q \} \ c_2 \ { \ R } \\
\{ P \} \ c_1 ; c_2 \ { \ R } \\
\{ P \land e \} \ c_1 \ { \ Q } \quad \{ P \land \neg e \} \ c_2 \ { \ Q } \\
\{ \ P \} \text{ if } (e) \text{ then } c_1 \text{ else } c_2 \ { \ Q } \\
\{ P \land e \} \ c \ { \ P } \\
\{ P \} \ \text{while } (e) \ \text{do } c \ { \ P \land \neg e } \\
\text{Rule of consequence:} \\
P \Rightarrow P' \quad \{ P' \} \ c \ { \ Q' } \quad Q' \Rightarrow Q \\
\{ P \} \ c \ { \ Q } 
\]
Soundness

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

t = x - y;

\begin{align*}
\text{while} \ (t > 0) \ & \{ \\
& x = x - 1; \\
& y = y + 1; \\
& t = t - 1; \}
\end{align*}

\{ 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) 
{
    x = x - 1; \\
y = y + 1; \\
t = t - 1;
}

\{ x=y_0+t \text{ and } y=x_0-t \text{ and } t\geq0 \text{ and } !(t>0) \} \implies \{ x = y_0 \text{ and } y = x_0 \} \\

P \implies P' \quad \{ P' \} \subset \{ Q' \} \\
Q' \implies Q \\
\{ P \} \subset \{ Q \} \quad (?)
Example

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

\[
t = x - y;
\]

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

\[
x = x - 1;
\]

\[
y = y + 1;
\]

\[
t = t - 1;
\]

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

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

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

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 \}
\begin{align*}
&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
\end{align*}
}

\{ 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 \, \} \\

\begin{align*}
t &= x - y; \\
\text{while } (t > 0) \{ \\
\begin{align*}
\{ \, 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 \quad x = x - 1; \\
\{ \, x & = y_0 + t - 1 \text{ and } y + 1 = x_0 - t + 1 \text{ and } t - 1 \geq 0 \, \} \\
& \quad \quad y = y + 1; \\
\{ \, x & = y_0 + t - 1 \text{ and } y = x_0 - t + 1 \text{ and } t - 1 \geq 0 \, \} \\
& \quad \quad t = t - 1; \\
\{ \, x & = y_0 + t \text{ and } y = x_0 - t \text{ and } t \geq 0 \, \} \\
\end{align*}
\}
\end{align*}

\{ \, 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

\begin{align*}
\{ x > y \text{ and } x = x_0 \text{ and } y = y_0 \} \\
t = x - y; \\
\{ x = y_0 + t \text{ and } y = x_0 - t \text{ and } t \geq 0 \} \\
\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 \} \\
& \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 \}
\end{align*}
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 \} \\
\quad t = x - y; \\
\{ x=y_0+t \text{ and } y=x_0-t \text{ and } t\geq 0 \} \\
while (t > 0) \{ \\
\quad \{ 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 \quad x = x - 1; \\
\quad \{ x=y_0+t-1 \text{ and } y+1=x_0-t+1 \text{ and } t-1\geq 0 \} \\
\quad \quad y = y + 1; \\
\quad \{ x=y_0+t-1 \text{ and } y=x_0-t+1 \text{ and } t-1\geq 0 \} \\
\quad \quad t = t - 1; \\
\quad \{ 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

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

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

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

or

\[
vc(\text{while } (e) \text{ do } c, Q) = \\
\text{Inv} \land \quad // \text{Must hold before loop!} \\
(\text{Inv} \land e) \Rightarrow vc(c,\text{Inv}) \land \quad // \text{Must hold locally for loop} \\
(\text{Inv} \land \neg e) \Rightarrow Q
\]
Example

\[ i = 5; \]
\[
\text{while (} i > 0 \text{) } \begin{cases} 
    
    \text{Inv} = \{ i \geq 0 \}
    
    
    i = i - 1;
    
    
    \{ i = 0 \} 
\end{cases}
\]

\[ \text{vc( while (} i>0 \text{) } \begin{cases} 
    
    \text{Inv} = \{ i \geq 0 \}
    
    
    i = i - 1;
    
    
    \{ i = 0 \} 
\end{cases} \] \]

\[ \text{vc breaks into following assertions:} \]
\[
\text{True } \Rightarrow \text{ wp}(i=5; \{ i\geq0 \})
\]
\[
i\geq0 \land
\]
\[
i\geq0 \land i>0 \Rightarrow \text{ wp}(i=i-1, \{i\geq0\})
\]
\[
\text{equiv. } i\geq0 \land i>0 \Rightarrow i-1\geq0 \land
\]
\[
i\geq0 \land !(i>0) \Rightarrow i=0
\]
Another Example

```
x >= 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 $\{ P \} \mathbf{c} \{ Q \}$ 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: \( x = 1 \) \( \lor \) \( x = -2 \)

ensures: \( y = 0 \)

\{
\begin{align*}
y &= x + 4; \\
\text{if} \ (x > 0) \ {\{}
\quad y &= x^2 - 1; \\
\text{else} \ {\{}
\quad y &= y + x; \\
\}
\}
\}

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

\((x=1 \text{ or } x=-2) \Rightarrow\)
\(( (x>0 \text{ and } x^2-x-1=0) \text{ or } \ (x\leq0 \text{ and } x+4+x=0))\)

\text{SMT-LIB code:}

\begin{align*}
(\text{declare-const } x \text{ Int}) \\
(\text{assert} \ (\text{and} \\
\quad (\text{or} \ (\text{=} \ x \ 1) \ (\text{=} \ x \ -2)) \\
\quad \text{(not} \\
\quad \quad (\text{or} \ (\text{and} \ (\text{=} \ x \ 0) \ (\text{=} \ (+ \ (+ \ x \ 4) \ x) \ 0)) \\
\quad \quad \quad (\text{and} \ (> \ x \ 0) \ (\text{=} \ (- \ (* \ x \ x) \ 1) \ 0))))) \\
(\text{check-sat}) \\
(\text{get-model})
\end{align*}
Example

requires: \( x \equiv 1 \lor x \equiv -5 \)

ensures: \( y \equiv 0 \)

\[
\begin{align*}
\{ \\
y & = x + 4; \\
\text{if} \ (x > 0) \ {\{ \\
\quad y & = x^2 - 1; \\
\} \text{ else} \ {\{ \\
\quad y & = y + x; \\
\} \\
\}
\end{align*}
\]

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

\((x=1 \text{ or } x=-5) \Rightarrow ((x>0 \text{ and } x^2-1 = 0) \text{ or } (x\leq 0 \text{ and } x+4+x=0))\)

SMT-LIB code:

(declare-const x Int)
(assert (and 
  (or (= x 1) (= x -5))
  (not 
    (or (and (\(\leq\) 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:

(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)