 SAT/SMT Solvers, Axiomatic Semantics

Announcements

- HW6 deadline is extended
- Now due next Thursday
- HW7 coming up next week
- Grades available for HW1-5 and Q1-5
  - Rainbow grades not updated yet
- Questions on HW6?

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

Reading

- If you are interested in the field
SAT Solvers

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

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, \(lp\) is a soft clause with weight 2, and \(lq\) is soft with weight 1
  - What assignment maximizes \((p \lor q) \land lp \land lq\)?

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 \(\{ b.f = 5 \} \ a.f = 5 \ a.f + b.f = 10 \} \) leads to formula \(select(f,b) = 5 \land store(f,a,5) \Rightarrow select(f,b) + select(f,a) = 10\)

 examples

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

- Lots of SMT solvers, e.g., Z3
  - My goal: become somewhat competent users of SMT solvers; be able to encode problems
  - Axiomatic Semantics --- key motivation for work on SMT!

Axiomatic Semantics

- Consider program fragment
  
    \[
    \begin{align*}
    t &= x - y; \\
    \text{while} \ (t > 0) \{} \\
    & \hspace{1cm} x = x - 1; \\
    & \hspace{1cm} y = y + 1; \\
    & \hspace{1cm} t = t - 1; \\
    \text{\} }
    \end{align*}
    \]

  - We are interested in proving these claims:
    - When \(x > y\), program terminates
    - When \(x > y\), values of \(x\) and \(y\) are swapped

Spring 19 CSCI 4450/6450, A. Milanova
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

Key idea:

- $\{ P \} \text{ code } \{ Q \}$
- Semantics of a program construct is defined in terms of logical assertions and the effect of the construct on these assertions
- Language of assertions
- Deductive reasoning

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...

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
  - $e ::= n \ | \ x \ | \ e_1 + e_2 \ | \ e_1 = e_2$
- Commands (i.e., statements, change state):
  - $c ::= x := e \ | \ c_1 ; c_2 \ | \ \text{if } (e) \ \text{then } c_1 \ \text{else } c_2 \ | \ \text{while } (e) \ \text{do } c \ | \ text{skip}$
- A big-step operational semantics
  - Judgments for expressions: $(e, \sigma) \rightarrow n$
  - Judgments for commands: $(c, \sigma) \rightarrow \sigma'$

Operational Semantics

- $\{ e, \sigma \} \rightarrow n$
- $(c_1, \sigma) \rightarrow \sigma' \ (c_2, \sigma') \rightarrow \sigma''$
- $(x := e, \sigma) \rightarrow \sigma[x := n]$
- $(c_1 ; c_2, \sigma) \rightarrow \sigma''$
- $(\text{if } (e) \ \text{then } c_1 \ \text{else } c_2, \sigma) \rightarrow \sigma''$
- $(e, \sigma) \rightarrow \text{True} \ (c_1, \sigma) \rightarrow \sigma'$
- $(e, \sigma) \rightarrow \text{False} \ (c_2, \sigma) \rightarrow \sigma''$
- $(\text{if } (e) \ \text{then } c_1 \ \text{else } c_2, \sigma) \rightarrow \sigma''$
- $(\text{while } (e) \ \text{do } c, \sigma) \rightarrow \sigma''$
- $(\text{while } (e) \ \text{do } c, \sigma) \rightarrow \sigma'$
Meaning of Assertions

\( \{ P \} c \{ Q \} \)

- Let \( P \) be logical assertion
  - E.g. \( x < y \) or \( x + y = 5 \)
  - \( P \) “references” state \( \sigma \)
- \( \sigma \models P \) (read: \( \sigma \) entails \( P \)) means that assertion \( P \) holds on state \( \sigma \)
  - E.g., \( \sigma = [x \rightarrow 5, y \rightarrow 10, z \rightarrow 0] \models x < y \)
  - Does \( \sigma' = [x \rightarrow 10, y \rightarrow 10, z \rightarrow 0] \models x < y \) ?
- Partial correctness \( \{ P \} c \{ Q \} \) therefore is
  - \( \forall \sigma, \forall \sigma'. (\sigma \models \text{P} \land (c, \sigma) \rightarrow \sigma') \Rightarrow \sigma' \models \text{Q} \)

Soundness

- For each Hoare triple \( \{ P \} c \{ Q \} \) deduced by the static semantics
  - \( \forall \sigma, \forall \sigma'. (\sigma \models \text{P} \land (c, \sigma) \rightarrow \sigma') \Rightarrow \sigma' \models \text{Q} \)
- Notice how in each one of our theories, AI, types, AS we have
  - Dynamic semantics
  - Static semantics
  - Soundness (connecting the two)

Static Semantics

<table>
<thead>
<tr>
<th>Rule</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>[x := e (P)]</td>
<td>({ P } c_1 (Q) \land c_2 (R))</td>
</tr>
<tr>
<td>((P \land e) c_1 (Q) \land c_2 (Q))</td>
<td>((P \land c_1) c_2 (Q))</td>
</tr>
<tr>
<td>((P \land \text{if} (e) \text{then} c_1 \text{else} c_2) c_3 (Q))</td>
<td>((P \land \text{if} (e) \text{do} c_1) c_3 (Q))</td>
</tr>
</tbody>
</table>

Example

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

\(t = x - y;\)

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

\(\{ x = y_0 \text{ and } y = x_0 \}\)

Example

\((P \land e) c (P)\)

\(\{ P \} c (Q)\)

\(Q' \Rightarrow Q\)

Example

\((P \land e) c (P)\)

\(\{ P \} \text{ while } (e) \text{ do } c (P \land \text{le})\)

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

\(t = x - y;\)

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

\(\{ x = y_0 \text{ and } y = x_0 \}\)

Example

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

\(t = x - y;\)

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

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

{ x > y and x = x_0 and y = y_0 } 
\[ t = x - y; \]
while \((t > 0)\) { 
(x = x_0 + t and y = x_0 - t and t < 120) simpl. \{ x = x_0 + t and y = x_0 - t and t < 120 \} 
\begin{align*}
& x = x - 1; \\
& (x = x_0 + t and y = x_0 - t and t < 120) \\
& y = y + 1; \\
& (x = x_0 + t and y = x_0 - t and t < 120) \\
& t = t - 1; \\
& (x = x_0 + t and y = x_0 - t and t < 120)
\end{align*}
} 
\( (x = 0 + t and y = x_0 - t and t < 120 and \neg (t > 0)) \implies \{ x = x_0 and y = x_0 \} \) 25

Example

\( P \implies P' \) 
\( \{ P' \} c Q' \)
\( Q' \implies Q \)
\( \{ R \} \)
\( \{ P \} \)
\( \{ Q \} \)
\( \{ R \} \)
\( \{ P \} c_1 \{ Q \} c_2 \{ R \} \)
\( \{ P \} \)
\( \{ Q \} \)
\( \{ R \} \)

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

\( wp(x := e, Q) = Q[e/x] \)
\( wp(c_1; c_2, Q) = wp(c_1, wp(c_2, Q)) \)
\( wp(if (e) then c_1 else c_2, Q) = e \land wp(c_1, Q) \lor !e \land wp(c_2, Q) \)
\( wp(while (e) do c_1, Q) = W = e \implies wp(c, W) \land !e \implies Q \)
Verification Condition

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

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

or

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

Example

\[ i = 5; \]
\[ \text{while (i > 0) \{} \]
\[ \text{inv = \{} i \geq 0 \}\]
\[ i = i - 1; \]
\[ \text{vc( while (i>0) \{} i = i-1; \}, \{i=0\}) \]
\[ \text{breaks into following assertions:} \]
\[ \text{True} \implies \text{wp(} i=5; \{ i \geq 0 \}) \]
\[ \text{i} \geq 0 \land \text{inv} \implies \text{vc(} c, \text{inv}\} \land \text{Must hold locally for loop} \]
\[ \text{inv} \land \neg e \implies Q \]

Another Example

\[ \{ x >= 0 \} \]
\[ i = x; \]
\[ z = 0; \]
\[ \text{while (i !} = 0 \{ \]
\[ z = z+1; \]
\[ i = i-1; \]
\[ \} \]
\[ \{ x = z \} \]

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 >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

SMT-LIB

- Your homework is to write a Tiny Dafny
  - Given a program \{ P \} c \{ Q \} generate verification conditions in SMT-LIB
  - Verify conditions with Z3
- Yet another programming language, OCaml
Suppose we need to verify \( \{ P \} \rightarrow \{ Q \} \).

Generate \( \text{vc}(c,Q) \).

Program verifies when \( P \Rightarrow \text{vc}(c,Q) \) 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: if \( \{ P \Rightarrow \text{vc}(c,Q) \} \) is satisfiable.
- If (unsat) program is verified!
- If (sat) get model.

\[
\text{Example requires: } x == 1 \text{ or } x == -2 \\
\text{ensures: } y == 0 \\
\{
    y = x + 4; \\
    \text{if } (x > 0) \{ \\
        y = x^2 - 1; \\
    \} \text{ else } \{ \\
        y = y + x; \\
    \}
\}
\]

\[
\text{SMT-LIB code:}
(\text{declare-const } x \text{ Int}) \\
(\text{assert (and (or (not (= x 1)) (= x -2)) (and (> x 0) (= (+ x 4) 0)) (and (<= x 0) (= (+ x x 1) 0)))}) \\
(\text{check-sat}) \\
(\text{get-model})
\]

\[
\text{Example requires: } x == 1 \text{ or } x == -5 \\
\text{ensures: } y == 0 \\
\{
    y = x + 4; \\
    \text{if } (x > 0) \{ \\
        y = x^2 - 1; \\
    \} \text{ else } \{ \\
        y = y + x; \\
    \}
\}
\]

\[
\text{SMT-LIB code:}
(\text{declare-const } x \text{ Int}) \\
(\text{assert (and (or (not (= x 1)) (= x -2)) (and (<= x 0) (= (+ x x 1) 0)))}) \\
(\text{check-sat}) \\
(\text{get-model})
\]

Another Example

Is this formula valid?

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

\[
\text{SMT-LIB code:}
(\text{declare-const } x \text{ Int}) \\
(\text{assert (and (or (not (= x 1)) (= x -2)) (and (<= x 0) (= (+ x x 1) 0)))}) \\
(\text{check-sat})
\]

\[
\text{Example from MIT 2015 Program Analysis OCW}
\]