Announcements

- HW6 due today, HW7 coming up
- Presentation guidelines and papers up on Schedule page
  1. Select available paper/slot from list (first-come-first-serve)
  2. If available, I’ll assign and update, otherwise goto 1

Outline

- SAT/SMT solvers
- Axiomatic semantics
  - IMP
  - Semantics
  - Verification condition generation
    - Similar to what we called “rules for backwards reasoning” in Principles
- SMT-LIB

Logical Reasoning

- A lot of recent PLSE research uses some form of automated logical reasoning
  - Non-standard type inference (SAT, MaxSAT)
  - Software verification (e.g., Dafny using SMT)
  - Symbolic execution (SMT)
  - Program synthesis

Reading

- If you are interested in the field
  - “The Calculus of Computation: Decision Procedures with Application to Verification” by Aaron Bradley and Zohar Manna, Spring 2007
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., \((p \rightarrow q) \rightarrow (p \land q)\) 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., \((p \lor q) \land lp \land lq\)
- 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

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

SMT Solvers

- Satisfiability Modulo Theories extends assertions/satisfiability beyond propositional logic and even beyond first-order 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 \(select(f1,b) = 5 \land store(f1,a,5) \Rightarrow select(f2,b) + select(f2,a) = 10\)

SMT Solvers

- Examples
  - \((z>5 \land x>0) \lor (z<5 \land x\leq0)\)
  - \((x>5 \land x+5>5) \lor (x\leq5 \land (x=0 \Rightarrow x+5+x=5))\)

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

Axiomatic Semantics

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

You Already Know This 😊

- 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

Operational Semantics

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

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

- Let \( P \) be a logical assertion
  - E.g. \( x < y \) or \( x + y = 5 \)
  - \( P \) “references” mappings in state \( \sigma \)
- \( \sigma \ |- \ P \) (read: \( \sigma \) entails \( P \)) means that assertion \( P \) holds on state \( \sigma \)
  - E.g., \( \sigma = [x\rightarrow5,y\rightarrow10,z\rightarrow0] \ |- \ x < y \)
  - Does \( \sigma' = [x\rightarrow10,y\rightarrow10,z\rightarrow0] \ |- \ x < y \) ?
- Partial correctness \( \{ P \} \ c \ \{ Q \} \) therefore is
  - \( \forall \sigma,\forall \sigma' \). \( \sigma |- P \wedge (c,\sigma) \rightarrow \sigma' \) \( \Rightarrow \) \( \sigma' |- Q \)

Soundness

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

Static Semantics

\[
\begin{align*}
\{ P[e/x] \} x:=e \{ P \} \\
\{ \{ P \} c \{ Q \} \} c \{ R \} \\
\{ P \wedge e \} c \{ Q \} \\
\{ P \wedge \neg e \} c \{ Q \} \\
\{ P \} \text{ if (e) then } c \{ Q \} \\
\{ P \} \text{ while (e) do } c \{ Q \} \\
\{ P \wedge e \} c \{ P \} \\
\{ P \wedge \neg e \} c \{ Q \} \\
\{ P \} \text{ } c \{ Q \} \\
\{ P \} \text{ if (e) then } c \{ Q \} \\
\{ P \} \wedge e \text{ do } c \{ P \} \wedge !e \\
\{ \{ P \} \} c \{ Q \} \\
\end{align*}
\]

Example

\[
\begin{align*}
\{ x>y \text{ and } x=x_0 \text{ and } y=y_0 \} \\
\text{while } (t > 0) \ { } \{ x \text{ = } x - 1; \\
y \text{ = } y + 1; \\
t \text{ = } t - 1; \\
\} \\
\{ x = y_0 + t \text{ and } y = x_0 + t \} \!
\end{align*}
\]
Example

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

Example

\( P \Rightarrow P' \)
\( h \{ P' \} \iling \{ Q' \} \)
\( Q' \Rightarrow Q \)
\( h \{ P \} \curring \{ Q \} \)
\( h \{ P \} \while (e) \dow h \{ P \wedge \neg e \} \{}
\( \{ x>y \text{ and } x=x_0 \text{ and } y=y_0 \} \Rightarrow \{ x=y_0+x-y \text{ and } y=x0-x+y \text{ and } x-y \geq 0 \} \)
\( t = x - y; \)
while (\( t > 0 \)) {
\( \{ x=y0+t-1 \text{ and } y=x0-t+1 \text{ and } t>0 \} \Rightarrow \{ x=y0+t \text{ and } y=x0-t \text{ and } t \geq 0 \} \)
\( x = x - 1; \)
\( \{ x=y0+t-1 \text{ and } y=x0-t+1 \text{ and } t \geq 0 \} \)
\( y = y + 1; \)
\( \{ x=y0+t-1 \text{ and } y=x0-t+1 \text{ and } t \geq 0 \} \)
\( t = t - 1; \)
\( \{ x=y0+t \text{ and } y=x0-t \text{ and } t \geq 0 \} \)
}\]
\( \{ x=y0+t \text{ and } y=x0-t \text{ and } t \geq 0 \text{ and } !(t>0) \} \Rightarrow \{ x=y0 \text{ and } y=x0 \} \)

Outline

- SAT/SMT solvers
- Axiomatic semantics
  - IMP
  - Semantics
  - Verification condition generation
    - Similar to what we called “rules for backwards reasoning” in Principles
- SMT-LIB

Weakest Precondition

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

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

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

\[
vc(\text{while } (e) \text{ do } c, Q) = \operatorname{Inv} \land \text{// Must hold before loop!}
\]

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

\[(\operatorname{Inv} \land \neg e) \Rightarrow Q
\]

Example

\[
i = 5;
while (i > 0) {
v(while (i>0) \{ i = i-1; \};, (i=0))
\}
\]

\[
vc = \{ i \geq 0 \}
\]

\[
i = i - 1;
\]

\[
\text{check-sat}
\]

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)

\[
(\text{declare-const } x \text{ Int}) \quad \text{declare an integer constant } x
\]

\[
(\text{assert } (> x 0)) \quad \text{add } x>0 \text{ to known facts}
\]

\[
(\text{check-sat}) \quad \text{checks if there exist an assignment}
\]

\[
(\text{get-model}) \quad \text{that makes all known facts true; returns}
\]

\[
\text{print this assignment}
\]

\[
\text{https://rise4fun.com/z3/tutorial}
\]

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

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

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

### Example

**Requires:** \( x \equiv 1 \lor x \equiv -2 \)

**Ensures** \( y \equiv 0 \)

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

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)
```

Another Example

Is this formula valid?

\( (x > 0 \land x + 5 > 5) \lor (x \leq 0 \land (x = 0 \Rightarrow x + x + 5 = 5)) \)

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)
```