Computer-Aided Program Design

Spring 2015, Rice University

Unit 1

Swarat Chaudhuri

January 22, 2015

Reasoning about programs

- A program is a mathematical object with rigorous meaning.
- It should be possible to prove *theorems* about programs.

Reasoning about programs

- A program is a mathematical object with rigorous meaning.
- It should be possible to prove *theorems* about programs.
  - "The program P *always* terminates"
  - "On each input x such that $(x > 0)$, P terminates and outputs $(x + 5)$."
  - "There is an input on which P does not terminate."
  - "There is a way to complete the partial program P such that the resulting program always terminates."

Reasoning about programs

- A program is a mathematical object with rigorous meaning.
- It should be possible to prove *theorems* about programs.
  - "The program P *always* terminates"
  - "On each input x such that $(x > 0)$, P terminates and outputs $(x + 5)$."
  - "There is an input on which P does not terminate."
  - "There is a way to complete the partial program P such that the resulting program always terminates."
- Proof gives us *certainty*, *reliability*...
  - ... to an extent not achieved by testing.

Reasoning about programs: Spot the bug!

```
int computeCurrentYear (int days) {
  /* input: number of days since Jan 1, 1980 */
  int year = 1980;
  while (days > 365) {
    if (isLeapYear(year)){
       if (days > 366) {
          days = days - 366;
          year = year + 1;
       }
    } else {
        days = days - 365;
        year = year + 1;
    }
  }
  return year;
}
```

See http://bit-player.org/2009/the-zune-bug for more details.

Reasoning about programs: Is this program correct?

```
do {
     AcquireSpinLock();
     nPacketsOld = nPackets;
     req = devExt->WLHV;
     if (req && req->status) {
        devExt->WLHV = req->Next;
        ReleaseSpinLock();
       nPackets++;
     }
  } while (nPackets != nPacketsOld);
  ReleaseSpinLock();
```

Challenges

- How to specify correctness properties of programs?

Challenges

- How to specify correctness properties of programs?
  - Mathematical logic
  - $\forall i, j : i < j \;\rightarrow\; A[i] < A[j]$

Challenges

- How to specify correctness properties of programs?
  - Mathematical logic
  - $\forall i, j : i < j \;\rightarrow\; A[i] < A[j]$
- Proofs about programs are complicated and tedious. Won't a human get them wrong?

Challenges

- How to specify correctness properties of programs?
  - Mathematical logic
  - $\forall i, j : i < j \;\rightarrow\; A[i] < A[j]$
- Proofs about programs are complicated and tedious. Won't a human get them wrong?
  - **Machine-checked proofs:** Proofs must be fully formal, and checked by an algorithm.
  - **Automatic proofs:** The proofs must be *generated* by an algorithm.

Automated reasoning about programs

- In principle, proving the correctness or incorrectness of a general program is *undecidable*.

Automated reasoning about programs

- In principle, proving the correctness or incorrectness of a general program is *undecidable*.
- In practice:
  - Focus on solvable special cases (in particular, finite-state programs)
  - Give *semi-algorithms* rather than algorithms.

Automated reasoning about programs

- In principle, proving the correctness or incorrectness of a general program is *undecidable*.
- In practice:
  - Focus on solvable special cases (in particular, finite-state programs)
  - Give *semi-algorithms* rather than algorithms.
- Questions:
  - Program verification
  - Program synthesis

This course: components

- Logic
  - *Decision procedures* for logic
- Verification of finite-state programs
- Verification of infinite-state programs
- Program synthesis

Rules

- No laptops in class.
- Attendance is important
  - No single textbook
  - Few slides
  - In-class activities
- TA: Keliang He
- More information on course webpage:
  `http://www.cs.rice.edu/~swarat/COMP507`

Propositional logic

- Let us first consider finite-state systems:
  - Hardware
  - Network protocols
  - Perhaps not software
- How do you describe correctness properties of such a system?

Propositional logic: Syntax

- Let *Prop* be a set of *propositional variables*. A formula F in propositional logic has the form

$$\begin{aligned} F \quad ::= \quad & p \mid \neg F_1 \mid F_1 \ \vee \ F_2 \mid F_1 \ \wedge \ F_2 \mid \\ & F_1 \ \rightarrow \ F_2 \mid F_1 \ \leftrightarrow \ F_2 \mid \top \mid \bot \end{aligned}$$

  where $p \in Prop$.
- In the above, F_1 and F_2 are *subformulas* of F.
- A *literal* is a formula of the form p or $\neg p$, where $p \in Prop$.

Propositional logic: Semantics

- Interpretation $I : \{P \mapsto \text{true}, Q \mapsto \text{false}, \cdots\}$

$$\begin{array}{ccccc} I & \models & F & \text{if } F \text{ evaluates to} & \text{true} & \text{under } I \\ I & \not\models & F & & \text{false} & \end{array}$$

Propositional logic: Semantics

- Interpretation $I : \{P \mapsto \text{true}, Q \mapsto \text{false}, \cdots\}$

$$\begin{array}{lllll} I \models F & \text{if } F \text{ evaluates to} & \text{true} & \text{under } I \\ I \not\models F & & \text{false} & \end{array}$$

- Inductive definition of semantics:

$$\begin{array}{rcl}
I \models \top & & \\
I \not\models \bot & & \\
I \models \neg F & \text{iff} & I \not\models F \\
I \models F_1 \wedge F_2 & \text{iff} & I \models F_1 \text{ and } I \models F_2 \\
I \models F_1 \vee F_2 & \text{iff} & I \models F_1 \text{ or } I \models F_2 \\
I \models F_1 \rightarrow F_2 & \text{iff} & I \not\models F_1 \text{ or } I \models F_2 \\
I \models F_1 \leftrightarrow F_2 & \text{iff} & I \models F_1 \text{ and } I \models F_2, \\
& & \text{or } I \not\models F_1 \text{ and } I \not\models F_2.
\end{array}$$

Using propositional logic

What does the following program do?

```
bool foo(unsigned int v) {
 unsigned int f;
 f = v & (v - 1);
 return (f == 0);
}
```

Using propositional logic

What does the following program do?

```
bool foo(unsigned int v) {
 unsigned int f;
 f = v & (v - 1);
 return (f == 0);
}
```

Verify this!

Satisfiability

- F is *satisfiable* iff there exists an interpretation I such that $I \models F$.
- F is *valid* iff for all interpretations I, $I \models F$.

Satisfiability

- F is *satisfiable* iff there exists an interpretation I such that $I \models F$.
- F is *valid* iff for all interpretations I, $I \models F$.
- F is valid iff $\neg F$ is unsatisfiable.

- Can you algorithmically check whether a formula is F is satisfiable?

Normal Forms

1. Negation Normal Form (NNF)

   Negations appear only in literals. (only $\neg$, $\wedge$, $\vee$)

2. Disjunctive Normal Form (DNF)

   Disjunction of conjunctions of literals

$$\bigvee_i \bigwedge_j \ell_{ij} \quad \text{for literals } \ell_{ij}$$

3. Conjunctive Normal Form (CNF)

   Conjunction of disjunctions of literals

$$\bigwedge_i \bigvee_j \ell_{ij} \quad \text{for literals } \ell_{ij}$$

The Resolution Procedure

Decides the satisfiability of PL formulae in CNF.

Resolution Rule: For clauses C_1 and C_2 in CNF formula F, derive *resolvent* using the following rule:

$$\frac{C_1[P] \qquad C_2[\neg P]}{C_1[\bot] \vee C_2[\bot]}$$

- Apply resolution and add resolvent to current set of clauses.
- If $\bot$ is ever deduced via resolution, then F must be unsatisfiable, as $F \wedge \bot$ is unsatisfiable.
- If every possible resolution produces an already-known clause, then F is satisfiable.

Resolution

Example:

1. $(P \to Q) \land P \land \neg Q$

Resolution

Example:

1. $(P \to Q) \land P \land \neg Q$
2. $(\neg P \lor Q) \land \neg Q$

Resolution: soundness and completeness

Soundness of resolution: Every unsatisfiability judgment derived by resolution is correct.

Completeness of resolution: Every correct unsatisfiability judgment can be derived by resolution.

[Look up the textbook The Calculus of Computation, *by Bradley and Manna.]*

Boolean Constraint Propagation (BCP)

Based on unit resolution

$$\frac{\ell \qquad C[\neg\ell]}{C[\bot]} \leftarrow \text{clause} \qquad\qquad \textit{where } \ell = P \textit{ or } \ell = \neg P$$

Boolean Constraint Propagation (BCP)

Based on unit resolution

$$\frac{\ell \qquad C[\neg\ell]}{C[\bot]} \leftarrow \text{clause} \qquad \qquad \textit{where } \ell = P \textit{ or } \ell = \neg P$$

Example:

$$F : P \wedge (\neg P \vee Q) \wedge (R \vee \neg Q \vee S)$$

Davis-Putnam-Logemann-Loveland (DPLL) Algorithm

- Decides the satisfiability of PL formulae in CNF.
- Decision Procedure DPLL: Given F in CNF

$$
\begin{array}{l}
\texttt{let rec } \textsc{dpll}\ F = \\
\quad \texttt{let } F' = \textsc{bcp}\ F \texttt{ in} \\
\quad \texttt{if } F' = \top \texttt{ then true} \\
\quad \texttt{else if } F' = \bot \texttt{ then false} \\
\quad \texttt{else} \\
\qquad \texttt{let } P = \textsc{choose}\ \mathsf{vars}(F') \texttt{ in} \\
\qquad (\textsc{dpll}\ F'\{P \mapsto \top\}) \ \vee\ (\textsc{dpll}\ F'\{P \mapsto \bot\})
\end{array}
$$

Davis-Putnam-Logemann-Loveland (DPLL) Algorithm

- Decides the satisfiability of PL formulae in CNF.
- Decision Procedure DPLL: Given F in CNF

$$
\begin{array}{l}
\texttt{let rec } \textsc{dpll}\ F = \\
\quad \texttt{let } F' = \textsc{bcp}\ F \texttt{ in} \\
\quad \texttt{if } F' = \top \texttt{ then true} \\
\quad \texttt{else if } F' = \bot \texttt{ then false} \\
\quad \texttt{else} \\
\qquad \texttt{let } P = \textsc{choose}\ \mathsf{vars}(F') \texttt{ in} \\
\qquad (\textsc{dpll}\ F'\{P \mapsto \top\})\ \lor\ (\textsc{dpll}\ F'\{P \mapsto \bot\})
\end{array}
$$

Optimization:
Don't CHOOSE only-positive or only-negative variables for splitting.

DPLL Example

$$F : (\neg P \vee Q \vee R) \wedge (\neg Q \vee R) \wedge (\neg Q \vee \neg R) \wedge (P \vee \neg Q \vee \neg R)$$

Exercise

- How does DPLL work on the following example?

$$(P \vee \neg Q \vee \neg R) \wedge (Q \vee \neg P \vee R) \wedge (R \vee \neg Q)$$

Exercise

- How does DPLL work on the following example?

$$(P \vee \neg Q \vee \neg R) \wedge (Q \vee \neg P \vee R) \wedge (R \vee \neg Q)$$

- Solve this example using Z3.

In-class exercise: N-Queens

- You are given an $N \times N$ chessboard. Your goal is to place N queens on the board so that no queen can hit any other.
- Show how to solve this problem using Z3 for $N = 4$.

Discussion

What about formulas that are not in CNF?

Homework exercise

- Use Z3 to check the correctness of

  ```
  bool foo(unsigned int v) {
   unsigned int f;
   f = v & (v - 1);
   return (f == 0);
  }
  ```

  under 4-bit integers.

Homework exercise

- Use Z3 to check the correctness of

  ```
  bool foo(unsigned int v) {
   unsigned int f;
   f = v & (v - 1);
   return (f == 0);
  }
  ```

  under 4-bit integers.
- More precisely, that it checks whether `v` is a power of 2.

There is a bug!

- 0 is incorrectly considered to be a power of 2.

There is a bug!

- 0 is incorrectly considered to be a power of 2.
- Fix:

```
bool foo(unsigned int v) {
 unsigned int f;
 f = v && !(v & (v - 1));
 return (f != 0);
}
```

See more bit hacks at http://graphics.stanford.edu/∼seander/bithacks.html.