Model Checking:
An Introduction
Meeting 2, CSCI 5535, Spring 2010

Announcements

- Moodle problems?
- Looked at the syllabus on the website?

- Next Week: There will be class
  - Guest lecturer: Sriram Sankaranarayanan
  - I will be out of town (e-mail ok)

Feedback

- "Find out what PL is about"
- "PL foundations"
- "Apply PL to ..."

You're in the right place!

Course Summary

Core Topics

- Semantics
  - Operational semantics
    - rules for execution on an abstract machine
    - useful for implementing a compiler or interpreter
  - Axiomatic semantics
    - logical rules for reasoning about the behavior of a program
    - useful for proving program correctness
  - Abstract interpretation
    - application: program analysis

- Types
  - $\lambda$-calculus
    - tiny language to study core issues in isolation

Course At-A-Glance

- Part I: Language Specification
  - Semantics = Describing programs
  - Evaluation strategies, imperative languages
- Part II: Language Design
  - Types = Classifying programs
  - Typed $\lambda$-calculus, functional languages
- Part III: Applications
But first …

Who are we again?
- We’re going to find critical bugs in important bits of software
  - using PL techniques!
- You’ll be enthusiastic about this
  - and thus want to learn the gritty details

Overarching Plan
Model Checking (today)
- Transition systems (i.e., models)
- Temporal properties
- Temporal logics: LTL and CTL
- Explicit-state model checking
- Symbolic model checking

Counterexample Guided Abstraction Refinement
- Safety properties
- Predicate abstraction
- Software model checking
- Counterexample feasibility
- Abstraction refinement weakest pre, thrm prv

Take-Home Message
- Model checking is the exhaustive exploration of the state space of a system, typically to see if an error state is reachable. It produces concrete counterexamples.
- The state explosion problem refers to the large number of states in the model.
- Temporal logic allows you to specify properties with concepts like “eventually” and “always”.

Spoiler
- This stuff really works!
- Symbolic model checking is a massive success in the model-checking field
- SLAM took the PL world by storm
  - Spawned multiple copycat projects
  - Launched Microsoft’s Static Driver Verifier (released in the Windows DDK)
Model Checking

There are complete courses in model checking (see ECEN 5139, Prof. Somenzi).

*Model Checking* by Edmund M. Clarke, Orna Grumberg, and Doron A. Peled.
*Symbolic Model Checking* by Ken McMillan.

We will skim.

Verification vs. Falsification

- What is verification?

- What is falsification?

Verification vs. Falsification

- An automated verification tool
  - can report that the system is verified (with a proof);
  - or that the system was not verified.

- When the system was not verified, it would be helpful to explain why.
  - Model checkers can output an error counterexample: a concrete execution scenario that demonstrates the error.

- Can view a model checker as a falsification tool
  - The main goal is to find bugs

  - So what can we verify or falsify?

Temporal Properties

**Temporal Property**

A property with time-related operators such as "invariant" or "eventually"

*Invariant*(\( p \))
is true in a state if property \( p \) is true in every state on all execution paths starting at that state

\[ G, AG, \square \] ("globally" or "box" or "forall")

*Eventually*(\( p \))
is true in a state if property \( p \) is true at some state on every execution path starting from that state

\[ F, AF, \diamond \] ("future" or "diamond" or "exists")

An Example Concurrent Program

- A simple concurrent mutual exclusion program
  - Two processes execute asynchronously
  - There is a shared variable \( turn \)
  - Two processes use the shared variable to ensure that they are not in the critical section at the same time

  - Can be viewed as a "fundamental" program: any bigger concurrent one would include this one

```plaintext
10: while (true) {
11:  wait(turn == 0); // critical section
12:  work(); turn = 1;
13: }
14: || // concurrently with
20: while (true) {
21:  wait(turn == 1); // critical section
22:  work(); turn = 0;
23: }
```
Reachable States of the Example Program

Example Properties of the Program

Example Properties of the Program

Temporal Logics

Execution Paths
Paths and Predicates

- We write $h \models p$ “the path $h$ makes the predicate $p$ true”
  - $h$ is a path in a transition system
  - $p$ is a temporal logic predicate

- Example:
  $A h. \quad h \models G (\neg \neg \neg \neg (pc1=12 \land pc2=22))$

--

Linear Time Logic (LTL)

- LTL properties are constructed from atomic propositions in AP; logical operators $\land, \lor, \neg$; and temporal operators $X, G, F, U$.
- The semantics of LTL is defined on paths.
  Given a path $h$:
  $h \models p$

--

Linear Time Logic (LTL)

- LTL properties are constructed from atomic propositions in AP; logical operators $\land, \lor, \neg$; and temporal operators $X, G, F, U$.
- The semantics of LTL is defined on paths:
  Given a path $h$:
  $h \models p$ iff $L(h_0, ap)$
  $h \models X p$ iff $h_1 \models p$
  $h \models F p$ iff $\exists i \geq 0. h_i \models p$
  $h \models G p$ iff $\forall i \geq 0. h_i \models p$
  $h \models p U q$ iff $\exists i \geq 0. h_i \models q$ and $\forall j<i. h_j \models p$

--

Satisfying Linear Time Logic

- Given a transition system $T = (S, I, R, L)$ and an LTL property $p$, $T$ satisfies $p$ if all paths starting from all initial states $I$ satisfy $p$

--

Computation Tree Logic (CTL)

- In CTL temporal properties use path quantifiers: $A : $ for all paths, $E : $ there exists a path
- The semantics of CTL is defined on states:
  Given a state $s$
  $s \models ap$ iff $L(s, ap)$
  $s \models EX p$ iff $\exists a path (s_0, s_1, s_2, ...). s_1 \models p$
  $s \models AX p$ iff $\forall paths (s_0, s_1, s_2, ...). s_1 \models p$
  $s \models EG p$ iff $\exists a path (s_0, s_1, s_2, ...). \forall i. o_{2i} \models p$
  $s \models AG p$ iff $\forall paths (s_0, s_1, s_2, ...). \forall i. o_{2i} \models p$
Linear vs. Branching Time

• LTL is a **linear time logic**
  - When determining if a path satisfies an LTL formula we are only concerned with a single path

• CTL is a **branching time logic**
  - When determining if a state satisfies a CTL formula we are concerned with multiple paths
  - In CTL the computation is instead viewed as a computation tree which contains all the paths

The expressive powers of CTL and LTL are incomparable (LTL $\subseteq$ CTL*, CTL $\subseteq$ CTL*)
- Basic temporal properties can be expressed in both logics
- Not in this lecture, sorry! (Take a class on Modal Logics)

For Next Time

• Post about today’s class and reading
• Read “Lazy Abstraction”
  - for the main ideas, ok to skim Sec. 7