Model Checking 2

Lecture 18
Reasoning About Concurrent Systems Is Hard

- Very hard to predict all possible ways in which thread execution steps can be interleaved.
- Often hard to determine/predict what sequences of actions the environment of a system may generate.
- If you’re not convinced, let’s consider a few very small examples...
**Model Checking**

Finite-state Program

```
void add(Object o) {
    buffer[head] = o;
    head = (head+1) % size;
}

Object take() {
    tail = (tail+1) % size;
    return buffer[tail];
}
```

Property 1: ...
Property 2: ...
...

Requirement

\[ \Box (\Phi \rightarrow \Diamond \Omega) \]

Temporal logic formula

Finite-state model

OK or Error trace

Line 5: ...
Line 12: ...
Line 15: ...
Line 21: ...
Line 25: ...
Line 27: ...
...
Line 41: ...
Line 47: ...

Line 1:

Line 2:

Line 3:

Line 4:

Line 5:

Line 6:

Line 7:

Line 8:

Line 9:

Line 10:

Line 11:

Line 12:

Line 13:

Line 14:

Line 15:

Line 16:

Line 17:

Line 18:

Line 19:

Line 20:

Line 21:

Line 22:

Line 23:

Line 24:

Line 25:

Line 26:

Line 27:

Line 28:

Line 29:

Line 30:

Line 31:

Line 32:

Line 33:

Line 34:

Line 35:

Line 36:

Line 37:

Line 38:

Line 39:

Line 40:

Line 41:

Line 42:

Line 43:

Line 44:

Line 45:

Line 46:

Line 47:

Line 48:

Line 49:

Line 50:

Line 51:

Line 52:

Line 53:

Line 54:

Line 55:

Line 56:

Line 57:

Line 58:

Line 59:

Line 60:

Line 61:

Line 62:

Line 63:

Line 64:

Line 65:

Line 66:

Line 67:

Line 68:

Line 69:

Line 70:

Line 71:

Line 72:

Line 73:

Line 74:

Line 75:

Line 76:

Line 77:

Line 78:

Line 79:

Line 80:

Line 81:

Line 82:

Line 83:

Line 84:

Line 85:

Line 86:

Line 87:

Line 88:

Line 89:

Line 90:

Line 91:

Line 92:

Line 93:

Line 94:

Line 95:

Line 96:

Line 97:

Line 98:

Line 99:

Line 100:
How to express requirements?

Different kinds of temporal logics

Syntax:
What are the formulas in the logic?

Semantics:
What does it mean for model M to satisfy formula $\phi$ in the logic?
Invariant

φ:

atomic propositions (e.g. x = 5, b = true)

¬φ

φ ∧ ψ

φ ∨ ψ

satisfaction:

M satisfies φ if all the reachable states satisfy φ
Invariant checking algorithm

Does M satisfy $\varphi$?

- Start at the initial states and explore the states of M using DFS or BFS.
- At any state, if $\varphi$ is violated then print an “error trace”
- If all reachable states have been visited then say “yes”
CTL - a branching time logic

\( \phi: \)

atomic propositions (e.g. \( x = 5, \ b = \text{true} \))

\( \neg \phi \)

\( \phi \lor \psi \)

\( \exists X \ \phi \)

\( \exists \phi \ U \ \psi \)

\( \exists G \ \phi \)

What is “satisfy”?
CTL - semantics

CTL is a state logic. Each formula is true or false in a state

s satisfies:
- atomic propositions: if $\Gamma(s)$ satisfies $\varphi$
- $\neg \varphi$: if s does NOT satisfy $\varphi$
- $\varphi \lor \psi$: (s satisfies $\varphi$) or (s satisfies $\psi$)
CTL - semantics

s satisfies:

- $\exists X \varphi$ : if there exists $t$ such that $R(s,t)$ and $t$ satisfies $\varphi$
- $\exists \varphi U \psi$ : if there exists a sequence of states $s,t,...v$ such that
  - $V$ satisfies $\psi$
  - $s,t,...$ all satisfy $\varphi$
- $\exists G \varphi$ : if there exists an infinite sequence of states starting from $s$ where $\varphi$ is always true
LTL - a linear time logic

φ:

atomic propositions

¬φ

φ ∧ ψ

φ ∨ ψ

φ U ψ
Why use Temporal Logics?

- Difficult to formalize a requirement in temporal logic

“Between the window open and the window close, button X can be pushed at most twice.”

…is rendered in LTL as...

\[
\begin{align*}
&((\text{open} \lor \neg \text{close}) \rightarrow \\
&\quad(((\neg \text{pushX} \lor \neg \text{close}) \lor \\
&\quad\quad(\text{close} \lor (((\neg \text{pushX} \lor \neg \text{close}) \lor \\
&\quad\quad\quad(\text{close} \lor (((\neg \text{pushX} \lor \neg \text{close}) \lor \\
&\quad\quad\quad\quad(\text{close} \lor (\neg \text{pushX} \lor \text{close}))))))))))))
\end{align*}
\]
Requirements for programmers

- Express correctness as assertions in code
- ...
- or in domain-specific language that gets translated into assertions in code

- Goal: Find assertion violations
State Explosion
State Explosion

- 2 threads with n instructions
  - between $2^n$ and $2^{n+1}$ traces
  - $n^2$ reachable states

- n threads with 1 instructions
  - n! traces
  - $2^n$ reachable states
Avoiding State Explosion

- Partial order reduction
- Symmetry reduction
- Garbage collection
- Fingerprinting
- Symbolic techniques
Partial Order Reduction

- Naive MC approach: Explore all interleavings

- Optimizations:
  - What if we know some data is
    - thread local
    - lock protected
  - What if we *think* some data is
    - thread local
    - lock protected
Symmetry Reduction

- Suppose each state is
  - (Global data, local data of T1, local data of T2)
  - and each thread executes same code

- Consider states
  - (G, L, L’) and (G, L’, L)
  - suppose we’ve done DFS from (G, L, L’)
  - and then we see (G, L’, L)

- What do we do?
  - How do we generalize this idea?
Garbage Collection

- Suppose we’ve seen state $S$ during DFS
- and then we later see a different state $S'$
- that only differs from $S$ in that it has different garbage

- What do we do?
  - How do we do this in general?
Fingerprinting

- During DFS, we’ve seen many states
  - S1, S2, S2, ..., s23229, ...
  - and then we get to a state S

- How do we (efficiently) check if we’ve seen S before?
Symbolic Model Checking
Ordered decision tree for $f = a \oplus b \oplus c \oplus d$
OBDD reduction (hash consing)

\[ f = a \oplus b \oplus c \oplus d \]
OBDD Properties

- Often compact
  - variable order strongly affects size
  - can blow up
- Canonical representation
  - for given variable order
- Efficiently represent a set of states
- Fast, industrial-strength BDD packages
OBDD Challenge

- Write code to perform boolean operations on BDDs!
  - and, or, not, exists, forall
  - what is complexity
    - of operation?
    - of result?