

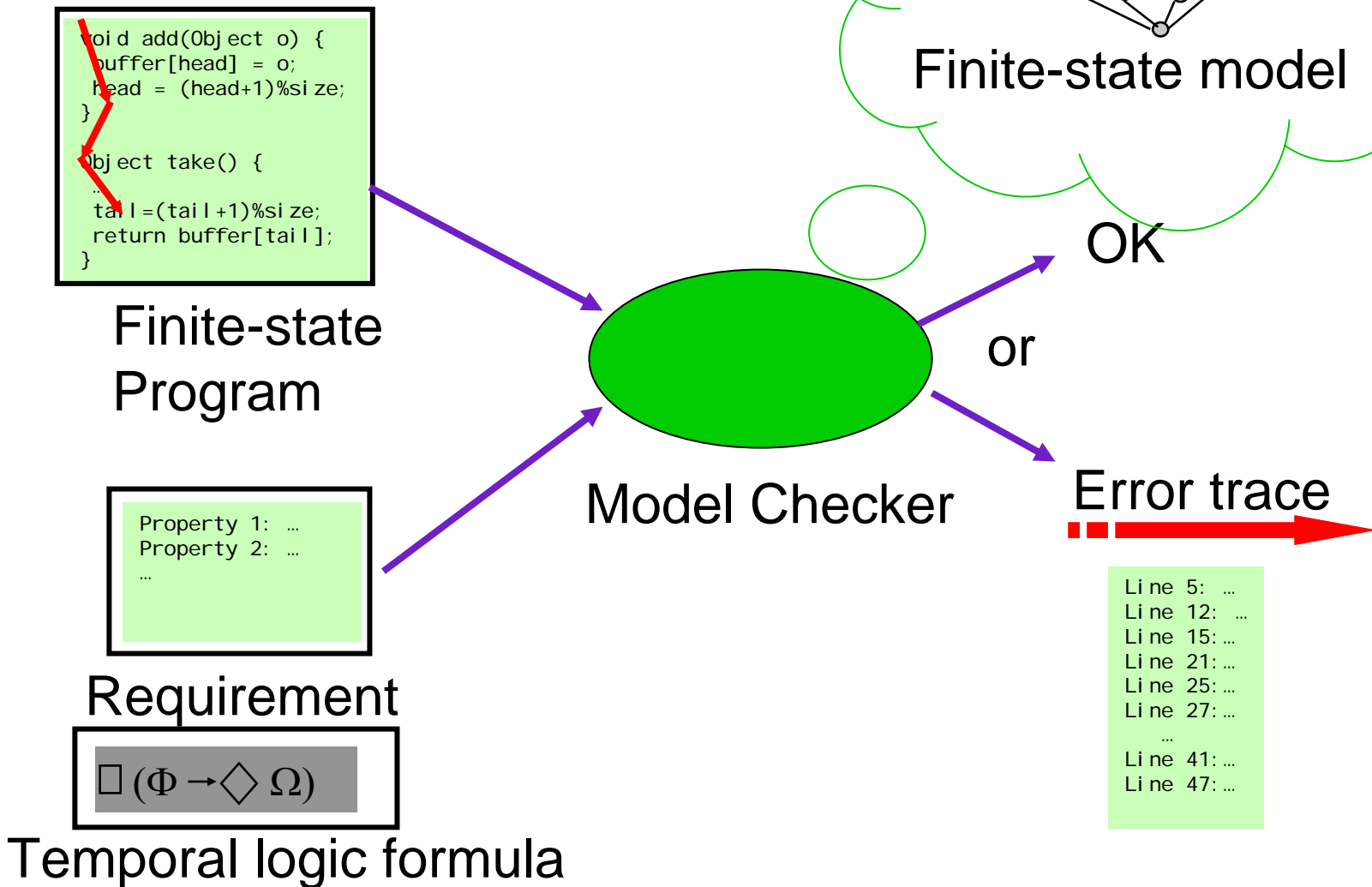
# Model Checking 2

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



# 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  $\varphi$  in the logic ?

# Invariant

$\varphi$ :

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

$\neg\varphi$

$\varphi \wedge \psi$

$\varphi \vee \psi$

satisfaction:

M satisfies  $\varphi$  if all the reachable states satisfy  $\varphi$

# 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

$\varphi$ :

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

$\neg\varphi$

$\varphi \vee \psi$

$\exists X \varphi$

$\exists \varphi \mathbf{U} \psi$

$\exists \mathbf{G} \varphi$

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  $\varphi$
- $\varphi \vee \psi$  : (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  $s,t,\dots,v$  such that
  - $v$  satisfies  $\psi$
  - $s,t,\dots$  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

$\varphi$ :

atomic propositions

$\neg\varphi$

$\varphi \wedge \psi$

$\varphi \vee \psi$

$\varphi \mathbf{U} \psi$

# 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.”

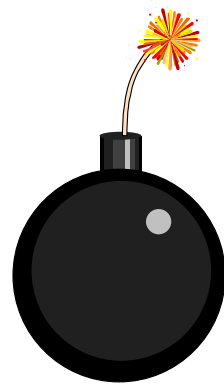
**...is rendered in LTL as...**

```
((open ∧ <>close) ->
  ((!pushX ∧ !close) U
    (close ∨ ((pushX ∧ !close) U
      (close ∨ ((!pushX ∧ !close) U
        (close ∨ ((pushX ∧ !close) U
          (close ∨ (!pushX U close))))))))))
```

# 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

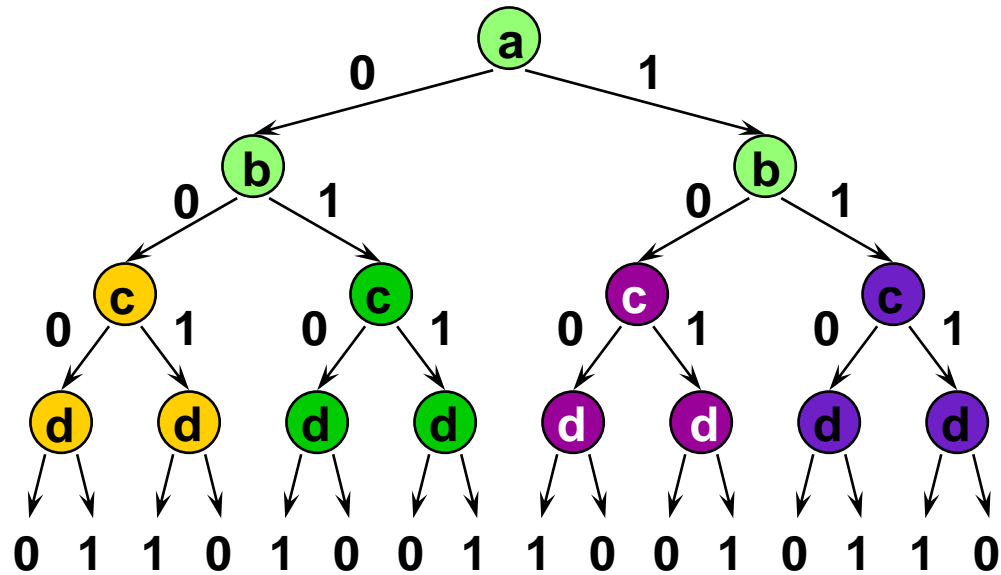
- During DFS, we've seen many states
  - $S_1, S_2, S_2, \dots, s_{23229}, \dots$
  - and then we get to a state  $S$
- How do we (efficiently) check if we've seen  $S$  before?

# Symbolic Model Checking

# Binary Decision Diagrams

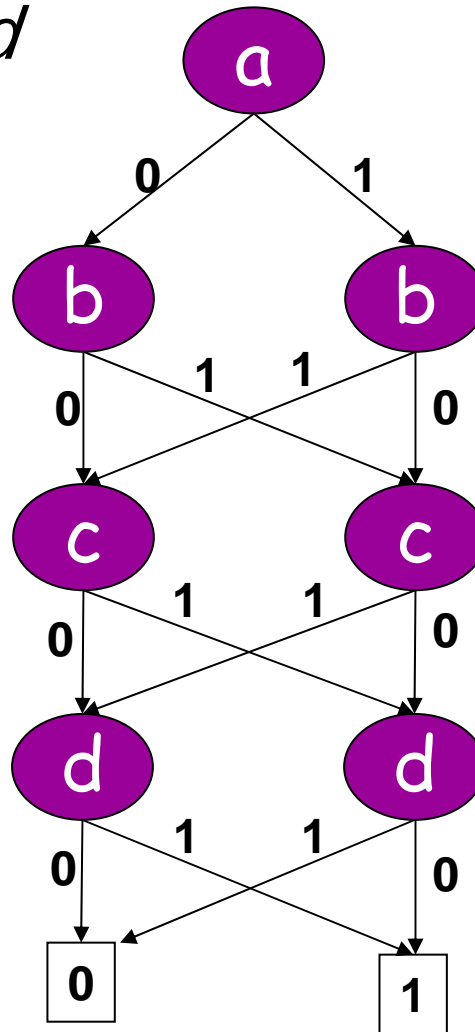
[Bryant]

Ordered decision tree for  $f = a \oplus b \oplus c \oplus d$



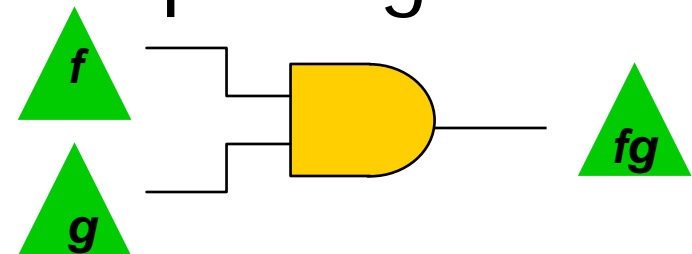
# OBDD reduction (hash consing)

$$f = a \oplus b \oplus c \oplus d$$



# 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?

