Model Checking

Boris Feigin
b.feigin@cs.ucl.ac.uk

University College London

March 9, 2005

Outline

Introduction

In a nutshell...

Model checking is a collection of techniques for *automated formal verification* of finite-state concurrent systems.

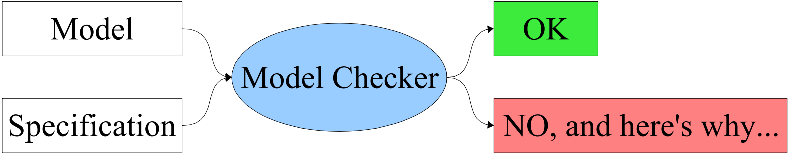
Best suited for analysis of...

***Reactive* systems**

Characterized by continuous interaction with environment.
Control-oriented.

- Hardware controllers
- Protocols of various kinds

Basic Operation



- *Model* — usually, an abstraction of the system being analysed (for example, using some process algebra or even UML)
- *Specification* — properties the system must satisfy (e.g absence of deadlocks, liveness, invariants etc.), expressed in some suitable formalism

Transition Systems (Model)

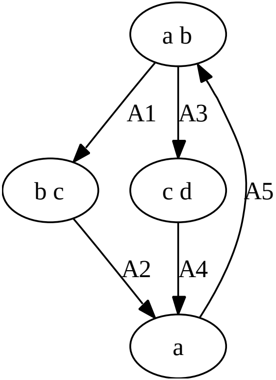
We reason about reactive systems in terms of their *state* and hence model their behaviour using *state transition systems*.

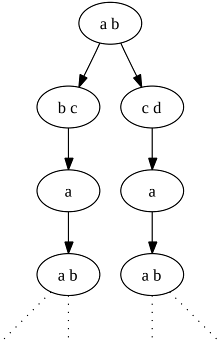
Definition [Müller-Olm et al.]

A *Kripke transition system* T over a set of *atomic propositions* AP is a four-tuple $(S, \mathrm{Act}, \rightarrow, I)$ where

- S — set of states
- Act — set of actions (e.g. program statements)
- $\rightarrow \subseteq S \times \mathrm{Act} \times S$ — transition relation
- $I : S \rightarrow 2^{\mathrm{AP}}$ — *interpretation*; $I(s)$ for some $s \in S$ is the set of propositions which are true in s (e.g. $a = 1$)

Illustration





We can *root* T with an initial state $s_0 \in S$ and unfold it into an infinite *execution tree* (you'll see later why we might want to do this)

Example

We'll use the *finite state processes* (FSP) process algebra from concurrency theory to describe a *Labeled Transition System* for the Dining Philosophers Problem.

Demo using LTSA example for Dining Philosophers

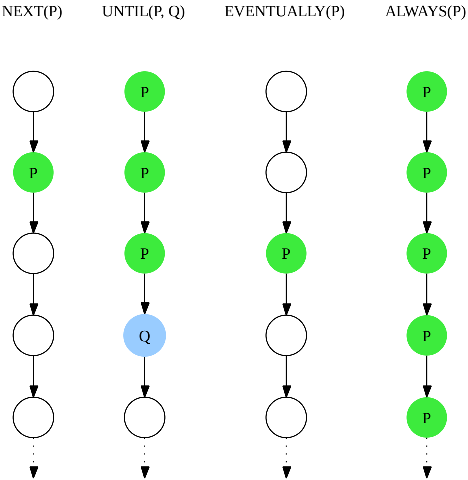
Temporal Logics (Spec)

- Natural formalism for expressing assertions about system evolution
- Differentiate between
  - *Linear-time* — consider *linear* paths in execution tree
  - *Branching-time* — quantify over paths
- Construct formulae from *atomic propositions* and *boolean and temporal connectives*

Propositional Linear-Time Logic (PLTL)

- Important operators:
  - **X** φ ("next φ")
  - φ **U** ψ ("φ until ψ")
  - **F** φ ("eventually φ") — liveness
  - **G** φ ("always φ") — safety
- Used to express *correctness* properties of the system

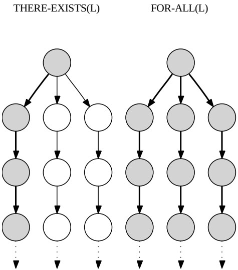
Illustration



Computation Tree Logic (CTL)

- Introduce *selectivity*: "there exists" **E** and "for all" **A**
- Path quantifiers above are combined with PLTL operators

Illustration and Examples



Examples

(from Merz) **AG** $\neg(owns_1 \wedge owns_2)$ — mutual exclusion

(from Clarke et al.) **AG** (**EF** *Restart*) — can always restart

Model Checking

Given a transition system T and a temporal logic formula φ, the model checker decides whether $T \models \varphi$ is true ("T is a model of φ")

Algorithms fall into two categories:

- Local — used with **PLTL**
- Global — **CTL**

The State Explosion Problem

Major Obstacle

Complex systems have astronomical numbers of states

The problem of *state explosion* plagues approaches relying on explicit construction of states

Also worth noting that,

- In practice, descriptions in high-level modeling languages are used in preference to low-level constructs such as those discussed
- HOWEVER, the size of transition systems derived from such descriptions grows *exponentially* with the length of the description

Techniques

- *Partial Order Reduction*
  - Consider concurrent processes — T contains all possible interleavings of actions of individual processes
  - Provided processes are loosely coupled, may be able to exploit commutativity of actions to remove a lot of redundancy
- *Abstraction*
  - Omit superfluous detail which does not affect the property being checked
- *Symmetry Reduction*
  - Exploit structural regularities in the system

Symbolic Model Checking

Thinking out of the box

Can explicit construction of states be avoided altogether?

- *Binary Decision Diagrams (BDDs)* — an efficient encoding of boolean formulae — serve as implicit representation of states and transitions
- Temporal formulae can be model-checked on BDDs directly
- Hence, can handle much larger numbers of states

Example Applications

- Fluke microkernel IPC subsystem
- BB84 quantum key distribution scheme — see issue 11.3 of *ACM Crossroads*
- "Remote Agent" spacecraft controller

Software

- SPIN — accepts PROMELA as the input language
- Java PathFinder — works on level of Java bytecode; authors report successes with up to 100KLOCs
- Bandera tool set
- LTSA — uses FPS process algebra

Model Checking Vs. Deductive Verification

Deductive verification (or *theorem proving*) takes a different route: correctness proofs are constructed from axioms by application of inference rules.

Model Checking

- Automated — almost a “black box” tool
- Limited by state explosion
- “Brute force”

Deductive Verification

- Manual — requires substantial knowledge and experience
- Can handle infinite-state systems
- “Intelligent”

Summary

In a nutshell...

Model checking is a collection of techniques for *automated formal verification* of finite-state concurrent systems.

Pros

- Automagic
- Gives counterexample on violation of spec.
- Widely used: controllers, protocols, operating systems...

Cons

- State explosion limits applicability
- Beware human errors: poor modeling, incomplete spec. etc

Further Reading

E. M. Clarke, O. Grumberg, D. A. Peled
Model Checking
MIT Press, 1999

S. Merz
Model Checking: A Tutorial Overview
Lecture Notes in Computer Science 2067, pp. 3–38, 2001

M. Müller-Olm, D. Schmidt, B. Steffen
Model-Checking: A Tutorial Introduction
Lecture Notes in Computer Science 1694, pp. 330–354, 1999

P. Wolper
An Introduction to Model Checking, 1995