Model Checking Overview

Sérgio Campos, Edmund Clarke

Model Checking

Model checking is a technique that relies on building a finite model of a system and checking that a desired property holds in that model.

The check is performed by an *exhaustive state space search*.

The challenge is to devise algorithms and data structures that can handle very large models.

It has been used mainly in hardware and protocol verification and, more recently, in software systems.

There are two general approaches:

- *Temporal logic model checking*
- *Behavior conformance checking*

Model Checking Overview
S. Campos, E. Clarke
Model Checking
Model of Computation
Computation Tree Logic
The Model Checking Problem
Advantages & Disadvantages
Notable Examples

Temporal Logic Model Checking

Temporal logic model checking was developed independently by Clarke and Emerson and by Queille and Sifakis in the early 1980's.

In this approach specifications are expressed in a temporal logic and systems are modeled as finite-state transition graphs.

An exhaustive search procedure is used to *check* if a given transition graph is a *model* for the specification.

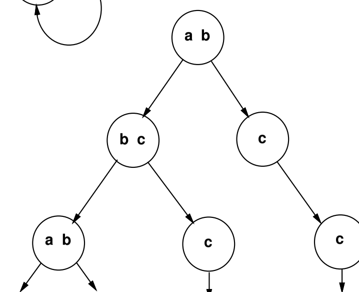
The term "model checking" was coined by Clarke and Emerson.

This work has been awarded the *2007 ACM Turing Award* to Clarke, Emerson and Sifakis!

Model Checking Overview
S. Campos, E. Clarke
Model Checking
Model of Computation
Computation Tree Logic
The Model Checking Problem
Advantages & Disadvantages
Notable Examples

Model of Computation

State Transition Graph or Kripke Model



Infinite Computation Tree

(Unwind State Graph to obtain Infinite Tree)

Model Checking Overview
S. Campos, E. Clarke
Model Checking
Model of Computation
Computation Tree Logic
The Model Checking Problem
Advantages & Disadvantages
Notable Examples

Computation Tree Logic (CTL)

CTL is a logic for reasoning about properties of state-transition graphs. It can succintly express many properties of sequential circuits and communication protocols.

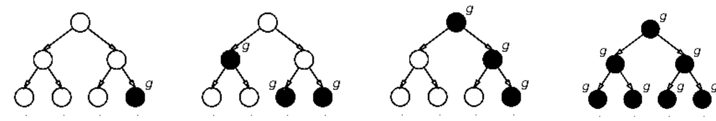
Each operator of the logic has two parts:

- Path quantifier:
  - **A**—"for every path"
  - **E**—"there exists a path"
- State quantifier:
  - **F**p—p holds sometime in the *future*
  - **G**p—p holds *globally* in the future
  - **X**p—p holds *next* time
  - p**U**q—p holds *until* q holds

Model Checking Overview
S. Campos, E. Clarke
Model Checking
Model of Computation
Computation Tree Logic
The Model Checking Problem
Advantages & Disadvantages
Notable Examples

Typical CTL formulas

- **EF**(*started* $\wedge \neg$*ready*): it is possible to get to a state where *started* holds but *ready* does not hold.
- **AG**(*req* $\Rightarrow$ **AF** *ack*): if a *request* occurs, then it will be eventually *acknowledged*.
- **AG**(**AF** *device_enabled*): *enabled* holds infinitely often on every computation path.
- **AG**(**EF** *restart*): from any state it is possible to get to the *restart* state.



Model Checking Overview
S. Campos, E. Clarke
Model Checking
Model of Computation
Computation Tree Logic
The Model Checking Problem
Advantages & Disadvantages
Notable Examples

The Model Checking Problem

Let M be the state–transition graph obtained from the concurrent system.

Let f be the specification expressed in temporal logic.

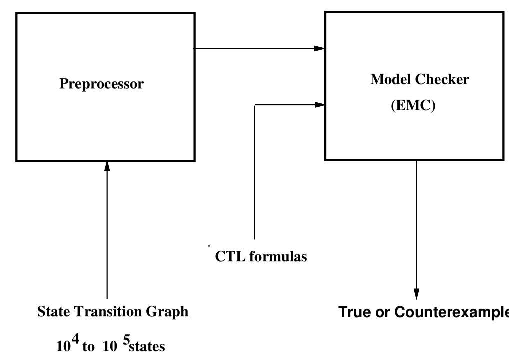
Find all states s of M such that

$$M, s \models f$$

Efficient Algorithms: CE81, CES83

Model Checking Overview
S. Campos, E. Clarke
Model Checking
Model of Computation
Computation Tree Logic
The Model Checking Problem
Advantages & Disadvantages
Notable Examples

The EMC System



Behavior Conformance Checking

Both the system and its specification are modeled as *automata*. These automata are compared to determine if the system behavior conforms to the specification.

Different notions of conformance have been explored, including:

- *Language inclusion*
- *Refinement orderings*
- *Observational equivalence*

Vardi and Wolper showed how temporal-logic model-checking could be recast in terms of automata, thus relating these two approaches.

Checking Language Inclusion

A finite *Büchi automaton* M is a 5-tuple

$$\langle K, p_0, \Sigma, \Delta, A \rangle$$

- K is a finite set of *states*
- $p_0 \in K$ is the *initial state*
- Σ is a finite *alphabet*
- $\Delta \subseteq K \times \Sigma \times K$ is the *transition relation*
- $A \subseteq K$ is the *acceptance set*

An infinite sequence $\sigma \in \Sigma^\omega$ is *accepted* by M iff it corresponds to the labels on the edges of a path that visits states in A infinitely often.

The set of sequences accepted by M is called the *language of* M and is denoted $\mathcal{L}(M)$.

Model Checking Overview
S. Campos, E. Clarke
Model Checking
Model of Computation
Computation Tree Logic
The Model Checking Problem
Advantages & Disadvantages
Notable Examples

Language Acceptance

An infinite sequence of states $p_0 p_1 p_2 \ldots \in K^\omega$ is a *path* in M if there exists an infinite sequence $a_0 a_1 a_2 \ldots \in \Sigma^\omega$ such that $\forall i \geq 0 : \langle s_i, a_i, s_{i+1} \rangle \in \Delta$.

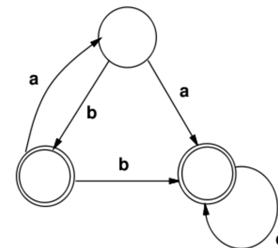
Let $p = p_0 p_1 p_2 \ldots \in K^\omega$ be a path in M. The *infinitary set* of p is the set of states that occur infinitely often on p.

A sequence $a_0 a_1 a_2 \ldots \in \Sigma^\omega$ is *accepted* by M if there is a corresponding path $p = p_0 p_1 p_2 \ldots \in K^\omega$ such that the infinitary set of p contains at least one element of A.

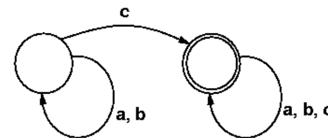
Model Checking Overview
S. Campos, E. Clarke
Model Checking
Model of Computation
Computation Tree Logic
The Model Checking Problem
Advantages & Disadvantages
Notable Examples

Implementation and Specification

M_{imp} corresponds to the *implementation*:



M_{spec} corresponds to the *specification*, "event C must happen at least once":

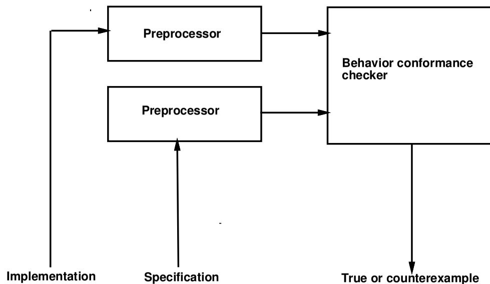


Model Checking Overview
S. Campos, E. Clarke
Model Checking
Model of Computation
Computation Tree Logic
The Model Checking Problem
Advantages & Disadvantages
Notable Examples

The Behavior Conformance Problem

Given two automata M_{imp} and M_{spec}, the *behavior conformance problem* consists of checking if

$$\mathcal{L}(M_{imp}) \subseteq \mathcal{L}(M_{spec})$$



Advantages of Model Checking

In contrast to theorem proving, model checking is *completely automatic* and *fast*, frequently producing an answer in a matter of minutes.

It can be used to check *partial specifications* and can provide useful information about correctness even if the system has not been completely specified.

Above all, model checking's *tour de force* is that it produces *counterexamples*, which usually uncover subtle errors in design that would be difficult to find otherwise.

Main Disadvantage

The main disadvantage of model checking is the *state explosion problem*.

BDDs can be used to represent state transition systems efficiently, dramatically increasing the size of the systems that can be verified.

Other techniques for alleviating state explosion include:

- Abstraction
- Compositional reasoning
- Symmetry
- Partial order reduction
- Localization reduction
- Semantic minimization

Model Checker Performance

Model checkers today can routinely handle systems with between 100 and 200 state variables.

They have checked interesting systems with 10^{120} reachable states.

By using appropriate abstraction techniques, systems with an essentially unlimited number of states can be checked.

As a result, model checking is becoming widely used in industry to aid in the verification of newly developed designs.

Notable Examples–IEEE Futurebus$^+$

In 1992 Clarke and his students at CMU used SMV to verify the cache coherence protocol in the IEEE Futurebus+ Standard.

They constructed a precise model of the protocol and showed that it satisfied a formal specification of cache coherence.

They found a number of previously undetected errors in the design of the protocol.

This was the first time that formal methods have been used to find errors in an IEEE standard.

Although the development of the protocol began in 1988, all previous attempts to validate it were based entirely on informal techniques.

Notable Examples–IEEE SCI

In 1992 Dill and his students at Stanford used Murφ to verify the cache coherence protocol of the IEEE Scalable Coherent Interface.

They modeled a typical configuration using the C code in the definition of the SCI standard.

Since the number of states of the model was very large, they verified only small instances of the system.

Nevertheless, they found several errors, ranging from uninitialized variables to subtle logical errors.

The errors also existed in the complete protocol, although it had been extensively discussed, simulated, and even implemented.

Notable Examples–HDLC

A High-level Data Link Controller (HDLC) was being designed at AT&T in Madrid.

In 1996 researchers at Bell Labs offered to check some properties of the design. The design was almost finished, so no errors were expected.

Within five hours, six properties were specified and five were verified, using the FormalCheck verifier.

The sixth property failed, uncovering a bug that would have reduced throughput or caused lost transmissions.

The error was corrected in a few minutes and formally verified using FormalCheck.

Notable Examples–Buildings

In 1995 the Concurrency Workbench was used to analyze an active structural control system to make buildings more resistant to earthquakes.

The control system sampled the forces being applied to the structure and used hydraulic actuators to exert countervailing forces.

The first model had more than 10^{19} states and was not directly analyzable. By using semantic minimization it was possible to derive a much smaller model.

A timing error was discovered that could have caused the controller to worsen, rather than dampen, the vibration experienced during earthquakes.

Temporal logic model checkers

- The very first two model checkers were EMC and CÆSAR.
- SMV is the first model checker to use BDDs.
- The Spin system uses partial order reduction to reduce the state explosion problem.
- Murφ and UV are based on the Unity programming language.
- The Concurrency Workbench verifies CCS processes for properties expressed as mu-calculus formulas.
- SVE, FORMAT, and CV all focus on hardware verification.
- Verus and Kronos check properties of real-time systems.
- HyTech is designed for reasoning about hybrid systems.

Behavior conformance checkers

- The Cospan/FormalCheck system is based on showing inclusion between omega automata.
- FDR checks refinement between CSP programs; recently, it has been used to debug the Needham-Schroeder authentication protocol.
- The Concurrency Workbench can be used to determine if two systems are observationally equivalent.

Directions for Future Research

- Integrate abstraction techniques into current verification systems.
- Investigate techniques for compositional reasoning.
- Continue research on the use of symmetry.
- Develop methods for verifying parameterized designs.
- Develop practical tools for real-time and hybrid systems.
- Combine model checking techniques with deductive verification.
- Study problems involved in verifying hardware-software codesigns.
- Develop tool interfaces that are suitable for system designers.