Introduction to Model Checking

Angelo Montanari

Dipartimento di Matematica e Informatica,
Università di Udine, Italy

Formal verification and model checking

Unlike simulation and testing techniques, formal verification conducts an EXHAUSTIVE EXPLORATION of all possible behaviors of a system

The use of theorem proves, term rewriting systems, and proof checkers in formal verification: they are time consuming and they often require a great deal of manual intervention

(Temporal logic) MODEL CHECKING: an alternative verification technique developed independently by Clarke and Emerson in USA and by Queille and Sifakis in France at the beginning of the '80s

SPECIFICATIONS are expressed as formulae in a propositional temporal logic to be checked against STATE-TRANSITION SYSTEMS (is the transition system a model of the specification?) by using efficient search procedures

The distinctive features of model checking

The method of MODEL CHECKING: a desired property of the system is verified over a given system (the model) through an exhaustive (explicit or implicit) enumeration of all the state reachable by the system and the behaviors that traverse through them

Distinctive features of model checking:

- it is FULLY AUTOMATIC
- when the design fails to satisfy a given property, a COUNTEREXAMPLE is produced that exhibits a behavior which falsifies the property (useful insight to understanding the reason for the failure and clues for fixing the problem)

Model Checking

In its original form, model checking is a simple (and very successful) method to decide the relation

$$\mathcal{M}, s \models \phi$$

where $\mathcal{M}$ is a Kripke structure (set of states plus accessibility relation), s is a designated state in $\mathcal{M}$, and ϕ is a formula of some temporal logic

Format of Kripke structures (over $P_1, \ldots, P_n$)

$$\mathcal{M} = (S, R, L)$$

where S is the state set, $R \subseteq S \times S$ is the transition relation, and $L : S \to 2^{\{P_1,\ldots,P_n\}}$ is the labeling function

Linear Time Logic (LTL)

Constructed from

- atomic propositions $P_1, \ldots, P_n$
- propositional connectives $\neg, \wedge, \vee, \rightarrow$, and $\leftrightarrow$
- unary temporal operators $X\cdot$ (next time), $F\cdot$ (eventually), $G\cdot$ (always), and the binary temporal operator $\cdot\mathcal{U}\cdot$ (until)

example of LTL formula:

$$G(P_1 \rightarrow X(\neg P_2 \mathcal{U} P_1))$$

"after any time where P_1 holds, P_2 is false until P_1 holds again"

$$\mathcal{M}, s \models \phi \Leftrightarrow \textbf{each path } \Pi = ss_1s_2\ldots \textbf{ through } \mathcal{M} \textbf{ satisfies } \phi$$

Computation Tree Logic (CTL)

Constructed from

- atomic propositions $P_1, \ldots, P_n$
- propositional connectives $\neg, \wedge, \vee, \rightarrow$, and $\leftrightarrow$
- unary temporal operators $EX\cdot$, $EF\cdot$, $EG\cdot$, $AX\cdot$, $AF\cdot$, $AG\cdot$, and the binary temporal operators $E(\cdot \mathcal{U} \cdot)$ and $A(\cdot \mathcal{U} \cdot)$

example of CTL formula:

$$AG(P_1 \rightarrow AX\ EFP_2)$$

"in each tree rooted at a son of a node where P_1 holds, there exists a path such that there exists a node in the path where P_2 holds"

$\mathcal{M}, s \models \phi \Leftrightarrow$ **the tree unraveling of $\mathcal{M}$ at state s satisfies ϕ**

Model Checking I: the Case of LTL

(Manna, Pnueli) Given the behavior graph for $\mathcal{M}$ and the tableau $\mathcal{T}_{\neg\phi}$ for $\neg\phi$, look for initial (starting from s), fulfilling (satisfying all promises), and fair (satisfying fairness conditions) trails in $\mathcal{M} \times \mathcal{T}_{\neg\phi}$.

(Vardi, Wolper) Given $(\mathcal{M}, s)$ and the LTL formula ϕ, construct a Büchi automaton $\mathcal{A}$ which accepts those paths $\Pi = ss_1s_2\ldots$ through $\mathcal{M}$ which violates ϕ

$$\mathcal{A} = \mathcal{A}_{(\mathcal{M},s)} \times \mathcal{A}_{\neg\phi}$$

Check for nonemptiness of $\mathcal{A}$ ($L(\mathcal{A})$ is the set of bug scenarios)

Complexity: PSPACE-complete

linear in $|\mathcal{M}|$ $(= |S| + |R|)$
exponential in $|\phi|$

Model Checking II: the Case of CTL

(Clarke, Emerson) Given $\mathcal{M}$ and the CTL formula ϕ, do the following for the subformulas ψ of ϕ (according to an increasing order of their complexities):

$$\text{compute } [\psi] = \{r \in S | \mathcal{M}, r \models \psi\}$$

Finally, Check whether $s \in [\phi]$

Remark. If $\mathcal{M}$ is finite, this can be done effectively by fixed-point computations.

Complexity: Polynomial

linear in $|\mathcal{M}|$ $(= |S| + |R|)$
linear in $|\phi|$

Practical Applications

$\mathcal{M}$: representation of circuits, protocols, ...

ϕ: formulation of desired properties

Task: Find states (or paths) in $\mathcal{M}$ which violate ϕ (bugs, error scenarios)

Use model checking as a debugging method

Problem: state space explosion

Prominent tools: SMV (McMillan), SPIN (Holzmann)

Conferences: CAV, TACAS, and many others

The state explosion problem

The main disadvantage of model checking is STATE EXPLOSION that can occur if the system being verified has many components that can make transitions in parallel

The parallel composition of two concurrent components is indeed modeled by taking the Cartesian product of the corresponding state spaces: the global state space of a reactive system may grow EXPONENTIALLY with the number of concurrent processes running in the system.

The exploration of such a a huge state space may be prohibitive even for an algorithm running in linear time in the size of the model

Possible solutions to the state explosion problem

Two main approaches have been proposed to cope with the state explosion problem:

- Symbolic model checking
- Partial order reduction

The role of BINARY DECISION DIAGRAMS (for synchronous circuits): data structures for the representation of Boolean functions, that make it possible to obtain concise representations for transition systems and to manipulate them quickly

To deal with asynchronous protocols, the size of the search space can be reduced by using the PARTIAL ORDER REDUCTION

Symbolic model checking

Symbolic model checking has been proposed by McMillan (1991). It replaces fixed-point computations over individual states by manipulations of definitions of state sets. It allows an exhaustive implicit enumeration of a huge number of states

states:	bit vectors of fixed length m
$S \subseteq \{0,1\}^m$	defined by a Boolean formula $\beta_S(x_1, \ldots, x_m)$
$R \subseteq S \times S$	defined by a Boolean formula $\beta_R(x_1, \ldots, x_m, x'_1, \ldots, x'_m)$
$L : S \rightarrow 2^{\{P_1, \ldots, P_n\}}$	defined by Boolean formulas $\beta_1(x_1, \ldots, x_m), \ldots, \beta_n(x_1, \ldots, x_m)$ (where β_i defines the set $\{s \in S : P_i \in L(s)\}$)

An Example

Let $S = \{0,1\}^3$ and let the "target" state be defined by $x_1 \wedge x_2 \wedge \neg x_3$. Furthermore, let R be defined by $\bigwedge_{i=1}^{3}(x_i \rightarrow x_i') \wedge (x_1 \leftrightarrow x_1' \vee x_2 \leftrightarrow x_2')$.

Compute the Boolean formula which defines $[EF\ target]$.

$[target]$ is defined by $x_1 \wedge x_2 \wedge \neg x_3$

$[target] \cup [EX\ target]$ is defined by $(x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge x_2 \wedge \neg x_3) \vee (x_1 \wedge \neg x_2 \wedge \neg x_3)$

$[target] \cup [EX\ target] \cup [EX\ EX\ target] = [EF\ target]$ is defined by $\neg x_3$

An Example (cont'd)

Key step:

- proceed from a state set T to $\{s \in S \mid \exists (s, s') \in R : s' \in T\}$
- proceed from a formula $\beta_T(\overline{x})$ to $\beta'(\overline{x}) = \exists \overline{y}(\beta_R(\overline{x}, \overline{y}) \wedge \beta_T(\overline{y}))$

Problem:

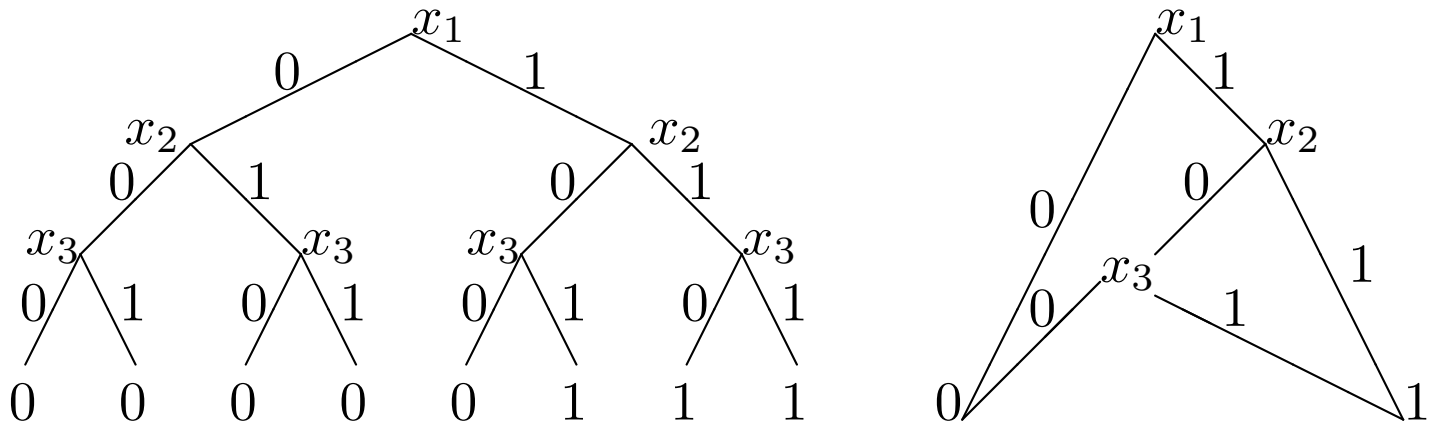
what is a good normal form for the representation of Boolean functions which allows efficient application of $\neg, \wedge, \vee$, and $\exists$?

Ordered Binary Decision Diagrams (OBDDs)

Ordered Binary Decision Diagrams (OBDDs) are based on work by Akers (1978) and Bryant (1986). They are reduced versions of decision trees for Boolean functions.

Example: $x_1 \wedge (x_2 \vee x_3)$

Decision tree vs. OBDD



Basic Results on OBDDs

1. Using the two reduction rules:
   - identify isomorphic "subgraphs"
   - for $x = 0, 1$, replace the paths $x0$ and $x1$ by the arc x, whenever $x0$ and $x1$ lead to the same value (cf. previous example)

   one obtains a canonical (unique) OBDD for a given Boolean function and a given order of variables
2. The operations $\neg, \wedge, \vee, \exists$ can be performed efficiently on OBDDs

Partial order reduction

Partial order reduction can be used to reduce the size of the state space making use of the following observation:

computations that differ in the ordering of
INDEPENDENTLY EXECUTED EVENTS are usually
indistinguishable by the specification and
thus they can be considered equivalent

It suffices to check a reduced state space, which contains (at least) one representative computation for each class of equivalent computations

Textbook

Model Checking, Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled, The MIT Press, 2000.