Model Checking: how to cope with the State Explosion Problem

Angelo Montanari

Dipartimento di Matematica e Informatica,
Università di Udine, Italy

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 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 $\beta_R(x_1, \ldots, x_m, x'_1, \ldots, x'_m)$
$L : S \to 2^{\{P_1, \ldots, P_n\}}$	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:

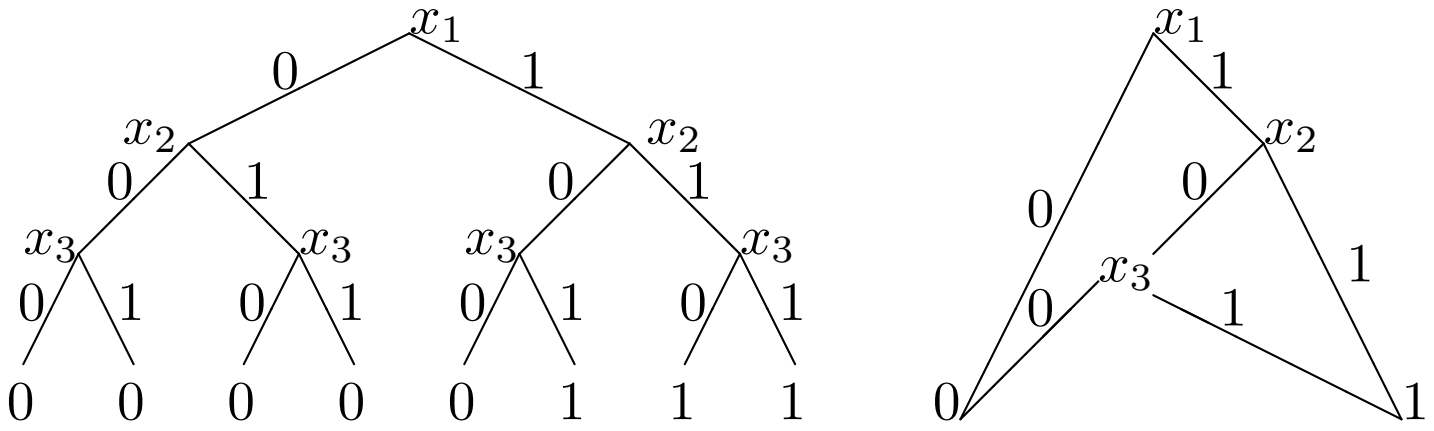
what is a good normal form for the representation of Boolean functions which allows efficient application of $\neg, \wedge, \vee$, and $\exists$?

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

Advanced techniques

- **Bounded model checking**: SAT solvers + symbolic model checking + bounded models
- **Satisfiability Modulo Theories** (SMT solvers): satisfiability of logical formulas with respect to one or more background theories formulated in first-order logic with equality (integers, real numbers, ..)
- **K-Liveness**: a simple but effective technique for LTL verification; it checks the absence of lasso-shaped counterexamples by trying to prove that bad states are visited at most k times, for increasing values of k (liveness checking as a sequence of safety checks)
- **IC3**: a very successful SAT-based model checking algorithm based on induction

Textbook

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