CDLM in Informatica, academic year 2014-2015

last update: Tuesday 23rd June, 2015, 13:48

Copyright notice: some material (text, figures) displayed in these slides is courtesy of M. Benerecetti, A. Cimatti, P. Pandya, M. Pistore, M. Roveri, and S. Tonetta, who detain its copyright. Some examples displayed in these slides are taken from [Clarke, Grunberg & Peled, “Model Checking”, MIT Press], and their copyright is detained by the authors. All the other material is copyrighted by Roberto Sebastiani. Every commercial use of this material is strictly forbidden by the copyright laws without the authorization of the authors. No copy of these slides can be displayed in public without containing this copyright notice.
Outline

1. Motivations
2. Ordered Binary Decision Diagrams
3. Symbolic representation of systems
4. Symbolic CTL Model Checking
5. A simple example
6. A relevant subcase: invariants
7. Symbolic CTL M.C: efficiency issues
8. Exercises
The Main Problem of CTL M.C. State Space Explosion

The bottleneck:
- Exhaustive analysis may require to store all the states of the Kripke structure, and to explore them one-by-one
- The state space may be exponential in the number of components and variables
  (E.g., 300 Boolean vars $\Rightarrow$ up to $2^{300} \approx 10^{100}$ states!)
- State Space Explosion:
  - too much memory required
  - too much CPU time required to explore each state

A solution: Symbolic Model Checking
Symbolic representation:

- manipulation of sets of states (rather than single states);
- sets of states represented by formulae in propositional logic;
  - set cardinality not directly correlated to size
- expansion of sets of transitions (rather than single transitions);
Motivations

Symbolic Model Checking [cont.]

- two main symbolic techniques:
  - Binary Decision Diagrams (BDDs)
  - Propositional Satisfiability Checkers (SAT solvers)

- Different model checking algorithms:
  - Fix-point Model Checking (historically, for CTL)
  - Fix-point Model Checking for LTL (conversion to fair CTL MC)
  - Bounded Model Checking (historically, for LTL)
  - Invariant Checking
  - ...

Roberto Sebastiani ()
Ch. 05: Symbolic CTL Model Checking
Tuesday 23rd June, 2015 6 / 78
Ordered Binary Decision Diagrams (OBDDs) [Bryant, ’85]

Canonical representation of Boolean formulas

- “If-then-else” binary DAGs with two leaves: 1 and 0
- Variable ordering $A_1, A_2, \ldots, A_n$ imposed a priori.
- Paths leading to 1 represent models
- Paths leading to 0 represent counter-models
OBDDs of \((a_1 \leftrightarrow b_1) \land (a_2 \leftrightarrow b_2) \land (a_3 \leftrightarrow b_3)\) with different variable orderings
Ordered Decision Trees

- **Ordered Decision Tree:** from root to leaves, variables are encountered always in the same order.
- **Example:** Ordered Decision tree for $\varphi = (a \land b) \lor (c \land d)$
From Ordered Decision Trees to OBDD’s: reductions

Recursive applications of the following reductions:
- *share subnodes*: point to the same occurrence of a subtree
- *remove redundancies*: nodes with same left and right children can be eliminated
Reduction: example

```
0 0 1 0 0 0 1 0 0 0 1 1 1 1 1 1
```

Ordered Binary Decision Diagrams
Assume the variable ordering $A_1, A_2, \ldots, A_n$:

\[
\begin{align*}
\text{OBDD}(\top, \{A_1, A_2, \ldots, A_n\}) &= 1 \\
\text{OBDD}(\bot, \{A_1, A_2, \ldots, A_n\}) &= 0 \\
\text{OBDD}(\varphi, \{A_1, A_2, \ldots, A_n\}) &= \text{if } A_1 \\
&\quad \text{then } \text{OBDD}(\varphi[A_1|\top], \{A_2, \ldots, A_n\}) \\
&\quad \text{else } \text{OBDD}(\varphi[A_1|\bot], \{A_2, \ldots, A_n\})
\end{align*}
\]
Incrementally building an OBDD

- $\text{obdd\_build}(\top, \{\ldots\}) := 1,$
- $\text{obdd\_build}(\bot, \{\ldots\}) := 0,$
- $\text{obdd\_build}(\langle \varphi_1 \ \text{op} \ \varphi_2 \rangle, \{A_1, \ldots, A_n\}) :=$
  \begin{align*}
  & \text{reduce}(
    \text{apply}(\text{op},
    \text{obdd\_build}(\varphi_1, \{A_1, \ldots, A_n\}), \text{obdd\_build}(\varphi_2, \{A_1, \ldots, A_n\}))
  \end{align*}
  \quad \text{op} \in \{\land, \lor, \rightarrow, \leftrightarrow\}$
Ordered Binary Decision Diagrams

Incrementally building an OBDD (cont.)

- \( \text{apply} \ (\text{op}, \ O_i, O_j) := (O_i \ \text{op} \ O_j) \) \iff (O_i, O_j \in \{1, 0\})

- \( \text{apply} \ (\text{op}, \ \text{ite}(A_i, \ \varphi_i^\top, \ \varphi_i^\perp), \ \text{ite}(A_j, \ \varphi_j^\top, \ \varphi_j^\perp)) := \)
  - if \( A_i < A_j \) \( \text{ite}(A_i, \ \text{apply} \ (\text{op}, \ \varphi_i^\top, \ \text{ite}(A_j, \ \varphi_j^\top, \ \varphi_j^\perp)), \ \text{apply} \ (\text{op}, \ \varphi_i^\perp, \ \text{ite}(A_j, \ \varphi_j^\top, \ \varphi_j^\perp))) \)
  - if \( A_i > A_j \) \( \text{ite}(A_j, \ \text{apply} \ (\text{op}, \ \text{ite}(A_i, \ \varphi_i^\top, \ \varphi_i^\perp), \ \varphi_j^\top)), \ \text{apply} \ (\text{op}, \ \text{ite}(A_i, \ \varphi_i^\top, \ \varphi_i^\perp), \ \varphi_j^\perp)) \)
  - if \( A_i = A_j \) \( \text{ite}(A_i, \ \text{apply} \ (\text{op}, \ \varphi_i^\top, \ \varphi_j^\top), \ \text{apply} \ (\text{op}, \ \varphi_i^\perp, \ \varphi_j^\perp)) \)

\( \text{op} \in \{\land, \lor, \rightarrow, \leftrightarrow\} \)
\( \varphi = (A_1 \lor A_2) \land (A_1 \lor \neg A_2) \land (\neg A_1 \lor A_2) \land (\neg A_1 \lor \neg A_2) \)
Critical choice of variable Orderings in OBDD’s

\[(a_1 \leftrightarrow b_1) \land (a_2 \leftrightarrow b_2) \land (a_3 \leftrightarrow b_3)\]
OBDD’s as canonical representation of Boolean formulas

- An OBDD is a **canonical representation** of a Boolean formula: once the variable ordering is established, equivalent formulas are represented by the same OBDD:

  \[ \varphi_1 \iff \varphi_2 \iff \text{OBDD}(\varphi_1) = \text{OBDD}(\varphi_2) \]

- equivalence check requires **constant time**!
- validity check requires constant time! \((\varphi \iff \top)\)
- \((\text{un})\text{satisfiability check requires constant time!} \ (\varphi \iff \bot)\)

- the set of the paths from the root to 1 represent all the **models** of the formula
- the set of the paths from the root to 0 represent all the **counter-models** of the formula
Exponentiality of OBDD’s

- The size of OBDD’s may grow exponentially wrt. the number of variables in worst-case
- Consequence of the canonicity of OBDD’s (unless $P = \text{co-NP}$)
- Example: there exist no polynomial-size OBDD representing the electronic circuit of a bitwise multiplier

**Note**

The size of intermediate OBDD’s may be bigger than that of the final one (e.g., inconsistent formula)
the equivalence check between two OBDDs is simple
  are they the same OBDD? (\(\equiv\) constant time)
the size of a Boolean composition is up to the product of the size of the operands:

\[
|f \circ op g| = O(|f| \cdot |g|)
\]
Boolean quantification

Shannon’s expansion:

- If $v$ is a Boolean variable and $f$ is a Boolean formula, then
  
  $\exists v.f := f|_{v=0} \lor f|_{v=1}$
  $\forall v.f := f|_{v=0} \land f|_{v=1}$

- $v$ does no more occur in $\exists v.f$ and $\forall v.f$ !!

- Multi-variable quantification:
  $\exists (w_1, \ldots, w_n).f := \exists w_1 \ldots \exists w_n.f$

Intuition:

- $\mu \models \exists v.f$ iff exists $tvalue \in \{\top, \bot\}$ s.t. $\mu \cup \{v := tvalue\} \models f$
- $\mu \models \forall v.f$ iff forall $tvalue \in \{\top, \bot\}$, $\mu \cup \{v := tvalue\} \models f$

Example: $\exists (b, c).((a \land b) \lor (c \land d)) = a \lor d$

Note

Naive expansion of quantifiers to propositional logic may cause a blow-up in size of the formulae
OBDD’s and Boolean quantification

- OBDD’s handle quantification operations quite efficiently
  - if \( f \) is a sub-OBDD labeled by variable \( v \), then \( f|_{v=1} \) and \( f|_{v=0} \) are the “then” and “else” branches of \( f \)

\[ f_v=1 \quad f_v=0 \]

\[ \implies \text{lots of sharing of subformulae!} \]
OBDD – summary

- **Factorize** common parts of the search tree (DAG)
- **Require setting a** variable ordering **a priori** (critical!)
- **Canonical representation** of a Boolean formula.
- Once built, logical operations (satisfiability, validity, equivalence) immediate.
- Represents all models and counter-models of the formula.
- **Require** exponential space in worst-case
- **Very efficient** for some practical problems (circuits, symbolic model checking).
Symbolic representation of systems

Symbolic Representation of Kripke Structures

Symbolic representation:
- sets of states as their characteristic function
- provide logical representation and transformations of characteristic functions

Example:
- three state variables $x_1, x_2, x_3$:
  \[
  \{000, 001, 010, 011\}
  \]
  represented as “first bit false”: $\neg x_1$
- with five state variables $x_1, x_2, x_3, x_4, x_5$:
  \[
  \{00000, 00001, 00010, 00011, 00100, 00101, 00110, 00111, \ldots, 01111\}
  \]
  still represented as “first bit false”: $\neg x_1$
Let $M = (S, I, R, L, AF)$ be a Kripke structure.

States $s \in S$ are described by means of an array $V$ of Boolean state variables.

A state is an truth assignment to each atomic proposition in $V$.

- 0100 is represented by the formula $(\neg x_1 \land x_2 \land \neg x_3 \land \neg x_4)$
- we call $\xi(s)$ the formula representing the state $s \in S$
  (Intuition: $\xi(s)$ holds iff the system is in the state $s$)

A set of states $Q \subseteq S$ can be (naively) represented by the formula $\xi(Q)$

$$\bigvee_{s \in Q} \xi(s)$$

Bijection between models of $\xi(Q)$ and states in $Q$. 
Remark

• every propositional formula is a (typically very compact) representation of the set of assignments satisfying it

• Any formula equivalent to \( \xi(Q) \) is a representation of \( Q \)
  \[ \iff \text{Typically } Q \text{ can be encoded by much smaller formulas than } \bigvee_{s \in Q} \xi(s)! \]

• Example: \( Q = \{00000, 00001, 00010, 00011, 00100, 00101, 00110, 00111, \ldots, 01111 \} \) represented as “first bit false”: \( \neg x_1 \)

\[
\bigvee_{s \in Q} \xi(s) = ( \neg x_1 \land \neg x_2 \land \neg x_3 \land \neg x_4 \land \neg x_5 ) \lor ( \neg x_1 \land \neg x_2 \land \neg x_3 \land \neg x_4 \land x_5 ) \lor ( \neg x_1 \land \neg x_2 \land \neg x_3 \land x_4 \land \neg x_5 ) \lor \ldots \lor ( \neg x_1 \land x_2 \land x_3 \land x_4 \land x_5 )
\]

\(2^4\) disjuncts
Symbolic Representation of Set Operators

One-to-one correspondence between sets and Boolean operators

- Set of all the states: $\xi(S) := \top$
- Empty set: $\xi(\emptyset) := \bot$
- Union represented by disjunction: $\xi(P \cup Q) := \xi(P) \lor \xi(Q)$
- Intersection represented by conjunction: $\xi(P \cap Q) := \xi(P) \land \xi(Q)$
- Complement represented by negation: $\xi(S/P) := \neg \xi(P)$
Symbolic Representation of Transition Relations

- The transition relation $R$ is a set of pairs of states: $R \subseteq S \times S$
- A transition is a pair of states $(s, s')$
- A new vector of variables $V'$ (the next state vector) represents the value of variables after the transition has occurred
- $\xi(s, s')$ defined as $\xi(s) \land \xi(s')$
- The transition relation $R$ can be (naively) represented by

$$\bigvee_{(s, s') \in R} \xi(s, s') = \bigvee_{(s, s') \in R} (\xi(s) \land \xi(s'))$$

Note

Each formula equivalent to $\xi(R)$ is a representation of $R$

$\implies$ Typically $R$ can be encoded by a much smaller formula than $\bigvee_{(s, s') \in R} \xi(s) \land \xi(s')$!
Example: a simple counter

MODULE main
VAR
  v0 : boolean;
  v1 : boolean;
  out : 0..3;
ASSIGN
  init(v0) := 0;
  next(v0) := !v0;

  init(v1) := 0;
  next(v1) := (v0 xor v1);

  out := v0 + 2*v1;

---

# Symbolic representation of systems

Example: a simple counter

```plaintext
MODULE main
VAR
  v0 : boolean;
  v1 : boolean;
  out : 0..3;
ASSIGN
  init(v0) := 0;
  next(v0) := !v0;

  init(v1) := 0;
  next(v1) := (v0 xor v1);

  out := v0 + 2*v1;
```

---

Roberto Sebastiani ()

Ch. 05: Symbolic CTL Model Checking

Tuesday 23rd June, 2015 30 / 78
Example: a simple counter [cont.]

\[
\begin{array}{cccc}
    v_0 & v_1 & v_0' & v_1' \\
    0  & 0  & 0   & 1   \\
    0  & 1  & 1   & 0   \\
    1  & 0  & 1   & 1   \\
    1  & 1  & 0   & 0   \\
\end{array}
\]

\[
\xi(R) = (v_0' \leftrightarrow \neg v_0) \land (v_1' \leftrightarrow v_0 \oplus v_1)
\]

\[
\bigvee_{(s,s') \in R} \xi(s) \land \xi(s') = \\
(\neg v_1 \land \neg v_0 \land \neg v_1' \land v_0') \lor \\
(\neg v_1 \land v_0 \land v_1' \land \neg v_0') \lor \\
(v_1 \land \neg v_0 \land v_1' \land v_0') \lor \\
(v_1 \land v_0 \land \neg v_1' \land \neg v_0')
\]
Pre-Image

(Backward) pre-image of a set:

Set theoretic view:

\[ \text{PreImage}(P, R) := \{ s \mid \text{for some } s' \in P, (s, s') \in R \} \]

Logical view:

\[ \xi(\text{PreImage}(P, R)) := \exists V'.(\xi(P)[V'] \land \xi(R)[V, V']) \]

\( \mu \) over \( V \) is s.t \( \mu \models \exists V'.(\xi(P)[V'] \land \xi(R)[V, V']) \) iff, for some \( \mu' \) over \( V' \), we have: \( \mu \cup \mu' \models (\xi(P)[V'] \land \xi(R)[V, V']) \), i.e., \( \mu' \models \xi(P)[V'] \) and \( \mu \cup \mu' \models \xi(R)[V, V'] \)

Intuition: \( \mu \iff s, \mu' \iff s', \mu \cup \mu' \iff \langle s, s' \rangle \)
Symbolic representation of systems

Example: simple counter

\[
\begin{array}{c|c|c|c}
v_1 & v_0 & v'_1 & v'_0 \\
0 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 \\
\end{array}
\]

\[
\xi(R) = (v'_0 \leftrightarrow \neg v_0) \land (v'_1 \leftrightarrow v_0 \oplus v_1)
\]

\[
\xi(P) := (v_0 \leftrightarrow v_1) \quad \text{(i.e., } P = \{00, 11\})
\]

\[
\xi(PreImage(P, R)) \quad \text{=} \quad \exists V'.(\xi(P)[V'] \land \xi(R)[V, V'])
\]

\[
\exists v'_0 v'_1.((v'_0 \leftrightarrow v'_1) \land (v'_0 \leftrightarrow \neg v_0) \land (v'_1 \leftrightarrow v_0 \oplus v_1))
\]

\[
(\neg v_0 \land v_0 \oplus v_1) \lor \top \lor \top \lor \top \lor (v_0 \land \neg(v_0 \oplus v_1))
\]

\[
v'_0 = \top, v'_1 = \top \quad v'_0 = \top, v'_1 = \bot \quad v'_0 = \bot, v'_1 = \top \quad v'_0 = \bot, v'_1 = \bot
\]

\[
v_1 \quad \text{(i.e., } \{10, 11\})
\]
Symbolic representation of systems

Pre-Image [cont.]

\[ \xi(P) = v_0 \leftrightarrow v_1 \]

\[ \xi(R) = (v'_0 \leftrightarrow \neg v_0) \land (v'_1 \leftrightarrow v_0 \oplus v_1) \]

\[ \xi(\text{PreImage}(P, R)) = \exists V'.((v'_0 \leftrightarrow v'_1) \land (v'_0 \leftrightarrow \neg v_0) \land (v'_1 \leftrightarrow v_0 \oplus v_1)) \]
Forward Image

- Forward image of a set:

  \[ \text{Image}(P, R) := \{ s' \mid \text{for some } s \in P, (s, s') \in R \} \]

- Set theoretic view:

  \[ \xi(\text{Image}(P, R)) := \exists V. (\xi(P)[V] \land \xi(R)[V, V']) \]

- Logical Characterization:
Example: simple counter

\[ \xi(R) = (v_0' \leftrightarrow \neg v_0) \land (v_1' \leftrightarrow v_0 \oplus v_1) \]
\[ \xi(P) := (v_0 \leftrightarrow v_1) \text{ (i.e., } P = \{00, 11\}) \]

\[ \xi(Image(P, R)) = \exists V. ((v_0 \leftrightarrow v_1) \land (v_0' \leftrightarrow \neg v_0) \land (v_1' \leftrightarrow v_0 \oplus v_1)) \]
\[ = \ldots \]
\[ = \neg v_1' \text{ (i.e., } \{00, 01\}) \]
Forward Image [cont.]

Symbolic representation of systems

\[ \xi(P) = v_0 \leftrightarrow v_1 \]

\[ \xi(R) = (v'_0 \leftrightarrow \neg v_0) \land (v'_1 \leftrightarrow v_0 \oplus v_1) \]

\[ \xi(\text{Image}(P, R)) = \exists V.((v_0 \leftrightarrow v_1) \land (v'_0 \leftrightarrow \neg v_0) \land (v'_1 \leftrightarrow v_0 \oplus v_1)) \]
Application of the Transition Relation

- Image and PreImage of a set of states S computed by means of quantified Boolean formulae
- The whole set of transitions can be fired (either forward or backward) in one logical operation
- The symbolic computation of PreImage and Image provide the primitives for symbolic search of the state space of FSM’s
Symbolic CTL model checking

- Problem: $M \models \varphi$?,
  - $M = \langle S, I, R, L, AP \rangle$ being a Kripke structure and
  - $\varphi$ being a CTL formula

- Solution: represent $I$ and $R$ as Boolean formulas $\xi(I), \xi(R)$ and encode them as OBDDs, and

- Apply fix-point CTL M.C. algorithm:
  - using OBDDs to represent sets of states and relations,
  - using OBDD operations to handle set operations
  - using OBDD quantification technique to compute Preimages
General Schema

Assume $\varphi$ written in terms of $\neg$, $\wedge$, $\text{EX}$, $\text{EU}$, $\text{EG}$

- A general M.C. algorithm (fix-point):
  1. represent $I$ and $R$ as Boolean formulas $\xi(I), \xi(R)$
  2. for every $\varphi_i \in \text{Sub}(\varphi)$, find $\xi([\varphi_i])$
  3. Check if $\xi(I) \rightarrow \xi([\varphi])$

Subformulas $\text{Sub}(\varphi)$ of $\varphi$ are checked bottom-up

- $\xi([\varphi_i])$ computed directly, without computing $[\varphi_i]$ explicitly!!!
  - Boolean operators handled directly by OBDDs
  - next temporal operators $\text{EX}$: handled by symbolic PreImage computation
  - other temporal operators $\text{EG}$, $\text{EU}$: handled by fix-point symbolic computation
Symbolic Denotation of a CTL formula $\varphi$: $\xi([\varphi])$

\[
\xi([\varphi]) := \xi(\{s \in S : M, s \models \varphi\})
\]

\[
\begin{align*}
\xi([false]) &= \bot \\
\xi([true]) &= \top \\
\xi([p]) &= p \\
\xi([\neg \varphi_1]) &= \neg \xi([\varphi_1]) \\
\xi([\varphi_1 \land \varphi_2]) &= \xi([\varphi_1]) \land \xi([\varphi_2]) \\
\xi([EX \varphi]) &= \exists V'. (\xi([\varphi])[V'] \land \xi(R)[V, V']) \\
\xi([EG \beta]) &= \nu Z . (\xi([\beta]) \land \xi([EX Z])) \\
\xi([E(\beta_1 U \beta_2)]) &= \mu Z . (\xi([\beta_2]) \lor (\xi([\beta_1]) \land \xi([EX Z])))
\end{align*}
\]

Notation: if $X_1$ and $X_2$ are OBDDs and $op$ is a Boolean operator, we write “$X_1 \text{ op } X_2$” for “reduce(obdd_merge(op, X_1, X_2))”
Symbolic CTL Model Checking

General M.C. Procedure

OBDD Check(CTL_formula $\beta$) {
    if (In_OBDD_Hash($\beta$))
        return OBDD_Get_From_Hash($\beta$);
    case $\beta$ of
        true: return obdd_true;
        false: return obdd_false;
        $\neg \beta_1$: return $\neg$ Check($\beta_1$);
        $\beta_1 \land \beta_2$: return (Check($\beta_1$) $\land$ Check($\beta_2$));
        EX$\beta_1$: return PreImage(Check($\beta_1$));
        EG$\beta_1$: return Check_EG(Check($\beta_1$));
        E($\beta_1$ U $\beta_2$): return Check_EU(Check($\beta_1$),Check($\beta_2$));
    }
}
**PreImage**

\[ \text{OBDD PreImage( OBDD } X \) \{ \\
\text{ return } \exists V'. ( X[V'] \land \xi(R)[V, V']) \}; \\
\} \]
Check\_EG

\textbf{OBDD Check\_EG(\textbf{OBDD }X)} \{  
\textbf{Y}' := \textbf{X}; \textbf{j} := 1; 
\textbf{repeat}
\quad \textbf{Y} := \textbf{Y}'; \textbf{j} := \textbf{j} + 1; 
\quad \textbf{Y}' := \textbf{Y} \land \text{PreImage}(\textbf{Y}));
\textbf{until} (\textbf{Y}' \leftrightarrow \textbf{Y});  
\textbf{return} \textbf{Y}; 
\}

Roberto Sebastiani ()  
Ch. 05: Symbolic CTL Model Checking  
Tuesday 23rd June, 2015  
45 / 78
### Check\_EU

**OBDD** \( \text{Check\_EU}(\text{OBDD } X_1, X_2) \) {

\[
Y' := X_2; \quad j := 1;
\]

**repeat**

\[
Y := Y'; \quad j := j + 1;
\]

\[
Y' := Y \lor (X_1 \land \text{PreImage}(Y));
\]

**until** \( Y' \leftrightarrow Y \);

**return** \( Y \);

}````
CTL Symbolic Model Checking – Summary

- Based on fixed point CTL M.C. algorithms
- Kripke structure encoded as Boolean formulas (OBDDs)
- All operations handled as (quantified) Boolean operations
- Avoids building the state graph explicitly
- reduces dramatically the state explosion problem
  → problems of up to $10^{120}$ states handled!!
A simple example

MODULE main
VAR
  b0 : boolean;
  b1 : boolean;
  ...
ASSIGN
  init(b0) := 0;
  next(b0) := case
    b0  : 1;
    !b0 : {0,1};
  esac;
  init(b1) := 0;
  next(b1) := case
    b1  : 1;
    !b1 : {0,1};
  esac;
  ...
A simple example [cont.]

- N Boolean variables $b_0, b_1, ...$
- Initially, all variables set to 0
- Each variable can pass from 0 to 1, but not vice-versa
- $2^N$ states, all reachable
- (Simplified) model of a student career behaviour.
A simple example: FSM

(transitive trans. omitted)

$2^N$ STATES

$O(2^N)$ TRANSITIONS
A simple example: $OBDD(\xi(R))$

$2N + 2$ NODES
A simple example: states vs. OBDD nodes [NuSMV.2]
A simple example: reaching $K$ bits true

- Property $\mathbf{EF}(b_0 + b_1 + \ldots + b(N - 1) = K) \ (K \leq N)$
  (it may be reached a state in which $K$ bits are true)
- E.g.: “it is reachable a state where $K$ exams are passed”
A simple example: FSM

$\binom{N}{K}$ STATES

$K=2$

$b_0, b_1$

$b_0, b_2$

$b_1, b_2$

$b_0$

$b_1$

$b_2$
A simple example: \( OBDD(\xi(\varphi)) \)

\[(N - K + 1) \cdot K + 2 \text{ NODES} \]
A simple example: states vs. OBDD nodes [NuSMV.2]
Invariant properties have the form $\mathbf{AG} \ p$ (e.g., $\mathbf{AG}\neg\mathit{bad}$)

Checking invariants is the negation of a reachability problem:
- is there a reachable state that is also a bad state?

$(\mathbf{AG}\neg\mathit{bad} = \neg\mathbf{EF}\mathit{bad})$

Standard M.C. algorithm reasons backward from the $\mathit{bad}$ by iteratively applying PreImage computations:

$$Y' := Y \lor \text{PreImage}(Y)$$

until (i) it intersect $[I]$ or (ii) a fixed point is reached
Symbolic Model Checking of Invariants [cont.]
Symbolic Forward Model Checking of Invariants

Alternative algorithm (often more efficient): forward checking

- Compute (the OBDD of) the set of bad states \([bad]\)
- Compute the set of initial states \(I\)
- Compute incrementally the set of reachable states from \(I\) until (i) it intersect \([bad]\) or (ii) a fixed point is reached
Computing Reachable states: basic

\textbf{OBDD} \texttt{Compute\_reachable()} \{ 
\begin{align*}
Y' & := \bot; Y := \top; j := 1; \\
\text{while} \; \neg (Y' \leftrightarrow Y) & \\
& \quad j := j + 1; \\
& \quad Y := Y'; \\
& \quad Y' := Y \lor \text{Image}(Y); \\
\}
\texttt{return} \; Y;
\}

Y=\text{reachable}
Computing Reachable states: advanced

**OBDD** `Compute_reachable()` {
  
  \[
  Y := F := I; \quad j := 1;
  \]
  
  while \( \neg (F \leftrightarrow \bot) \) do
    \[
    j := j + 1;
    \]
    \[
    F := \text{Image}(F) \land \neg Y;
    \]
    \[
    Y := Y \lor F;
    \]
  
  return \( Y \);
}

\( Y = \text{reachable}; F = \text{frontier} \) (new)
Computing Reachable states [cont.]
A relevant subcase: invariants

Checking of Invariant Properties

```cpp
bool Forward_Check_EF(OBDD BAD) {
    Y := F := l; j := 1;
    while ¬(F ↔ ⊥) and (F ∧ BAD) ↔ ⊥
        j := j + 1;
        F := Image(F) ∧ ¬Y;
    Y := Y ∨ F;
}
if F = ⊥ // fixpoint reached
    return false
else // counter-example
    return true
}
```

Y=reachable;F=frontier (new)
Checking of Invariant Properties [cont.]
Checking of Invariants: Counterexamples

- if layer $n$ intersects with the bad states, then the property is violated
- a counterexample can be reconstructed proceeding backwards
  (i) select any model of $BAD \land F[n]$ (we know it is satisfiable), call it $t[n]$
  (ii) compute $\text{Preimage}(t[n])$, i.e. the states that can result in $t[n]$ in one step
  (iii) compute $\text{Preimage}(t[n]) \land F[n-1]$, and select one model $t[n-1]$
- iterate (i)-(iii) until the initial states are reached
- $t[0], t[1], \ldots, t[n]$ is our counterexample
Checking of Invariants: Counterexamples [cont.]
Back to OBDDs: Efficiency Issues

OBDD packages provides efficient basis for Symbolic Model Checking:

- unique representant for each OBDD via hash tables
- complement-based representation of negation
- memoizing partial computations
- garbage collection mechanisms
- variable reordering algorithms, dynamic activation
- specialized algorithms for relational products for Image/PreImage computations
Most hardware design companies have their own Symbolic Model Checker(s)

- Intel, IBM, Motorola, Siemens, ST, Cadence, ...
- very advanced tools
- proprietary technology!

On the academic side

- CMU SMV [McMillan]
- VIS [Berkeley, Colorado]
- Bwolen Yang’s SMV [CMU]
- NuSMV [CMU, IRST, UNITN, UNIGE]
- ...

Symbolic Model Checkers
Let $\varphi \overset{\text{def}}{=} (A \land (B \lor C))$ and $\varphi' \overset{\text{def}}{=} \exists A. \forall B. \varphi$. Using the variable ordering “A, B, C”, draw the OBDD corresponding to the formulas $\varphi$ and $\varphi'$.

$\varphi \overset{\text{def}}{=} (A \land (B \lor C))$

[ Solution: 

![OBDD Diagram]

]
\[ \varphi' \overset{\text{def}}{=} \exists A. \forall B. (A \land (B \lor C)) \]

[ Solution: ]

\[ \varphi' \overset{\text{def}}{=} \exists A. \forall B. \varphi \]
\[ = \forall B. (A \land (B \lor C)) [A := \top] \]
\[ = \forall B. (B \lor C) \]
\[ = (B \lor C) [B := \top] \land (B \lor C) [B := \bot] \]
\[ = (\top \land C) \lor \bot \]
\[ = C \]

which corresponds to the following OBDD:
Ex: Symbolic CTL Model Checking

Given the following finite state machine expressed in NuSMV input language:

MODULE main
VAR v1 : boolean; v2 : boolean;
INIT (!v1 & !v2)
TRANS (next(v1) <-> !v1) & (next(v2) <-> (v1<->v2))

and consider the property $P \equiv (v_1 \land v_2)$. Write:

- the Boolean formulas $I(v_1, v_2)$ and $T(v_1, v_2, v'_1, v'_2)$ representing respectively the initial states and the transition relation of $M$.
  
  [ Solution: $I(v_1, v_2)$ is $(!v_1 \land !v_2)$, $T(v_1, v_2, v'_1, v'_2)$ is $(v'_1 \leftrightarrow !v_1) \land (v'_2 \leftrightarrow (v_1 \leftrightarrow v_2))$ ]

- the graph representing the FSM. (Assume the notation “$v_1v_2$” for labeling the states: e.g. “10” means “$v_1 = 1, v_2 = 0$”.)
  
  [ Solution: ]
the Boolean formula representing symbolically $\text{EX}P$. [The formula must be computed symbolically, not simply inferred from the graph of the previous question!]

[ Solution:

\[
\text{EX}(P) = \exists v_1', v_2'. (T(v_1, v_2, v_1', v_2') \land P(v_1', v_2'))
\]

\[
= \exists v_1', v_2'. ((v_1' \leftrightarrow \neg v_1) \land (v_2' \leftrightarrow (v_1 \leftrightarrow v_2)) \land (v_1' \land v_2')
\]

\[
\Rightarrow v_1' = T, v_2' = T
\]

\[
v_1' = T, v_2' = T
\]

\[
= (\neg v_1 \land \neg v_2) \lor \bot \lor \bot \lor \bot
\]

\[
= (\neg v_1 \land \neg v_2)
\]
Ex: Symbolic CTL Model Checking

Given the following finite state machine expressed in NuSMV input language:

```
VAR v1 : boolean; v2 : boolean;
INIT init(v1) <-> init(v2)
TRANS (v1 <-> next(v2)) & (v2 <-> next(v1));
```

write:

- the Boolean formulas \( I(v_1, v_2) \) and \( T(v_1, v_2, v'_1, v'_2) \) representing the initial states and the transition relation of \( M \) respectively.
  
  [ Solution: \( I(v_1, v_2) \) is \( (v_1 \leftrightarrow v_2) \), \( T(v_1, v_2, v'_1, v'_2) \) is \( (v_1 \leftrightarrow v'_2) \land (v_2 \leftrightarrow v'_1) \) ]

- the graph representing the FSM. (Assume the notation “\( v_1 v_2 \)” for labeling the states. E.g., “10” means “\( v_1 = 1, v_2 = 0 \).”)

[ Solution: ]
the Boolean formula \( R^1(v'_1, v'_2) \) representing the set of states which can be reached after exactly 1 step.

**NOTE:** this must be computed symbolically, not simply deduced from the graph of question b).

[ Solution: ]

\[
R^1(v'_1, v'_2) = \exists v_1, v_2. (I(v_1, v_2) \land T(v_1, v_2, v'_1, v'_2))
= \exists v_1, v_2. ((v_1 \leftrightarrow v_2) \land (v_1 \leftrightarrow v'_2) \land (v_2 \leftrightarrow v'_1))
= (((v_1 \leftrightarrow v_2) \land (v_1 \leftrightarrow v'_2) \land (v_2 \leftrightarrow v'_1))[v_1 = \bot, v_2 = \bot] \lor
  (((v_1 \leftrightarrow v_2) \land (v_1 \leftrightarrow v'_2) \land (v_2 \leftrightarrow v'_1))[v_1 = \bot, v_2 = T] \lor
  (((v_1 \leftrightarrow v_2) \land (v_1 \leftrightarrow v'_2) \land (v_2 \leftrightarrow v'_1))[v_1 = T, v_2 = \bot] \lor
  (((v_1 \leftrightarrow v_2) \land (v_1 \leftrightarrow v'_2) \land (v_2 \leftrightarrow v'_1))[v_1 = T, v_2 = T])
= (\neg v'_1 \land \neg v'_2) \lor \bot \lor \bot \lor (v'_1 \land v'_2)
= (\neg v'_1 \land \neg v'_2) \lor (v'_1 \land v'_2)
= (v'_1 \leftrightarrow v'_2)
\]