

A.A. 2009-2010, CDLS in Ing. Informatica

Introduction to Formal Methods

05: Symbolic CTL Model Checking

Roberto Sebastiani – rseba@disi.unitn.it

Some material (text, figures) displayed in these slides is courtesy of M. Benerecetti, A. Cimatti, P. Pandya, M. Pistore, M. Roveri, S.Tonetta.

Content

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., 100 boolean vars $\Longrightarrow$ up to $2^{100} > 10^{30}$ states!)
- State Space Explosion:
  - too much memory required $(10^{30} \times 100bit = 10^{23}Gbit)$
  - too much CPU time required to explore each state $(10^{30} \times 1ns > 10^{12}anni)$.

▷ A solution: Symbolic Model Checking

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);

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
- ...

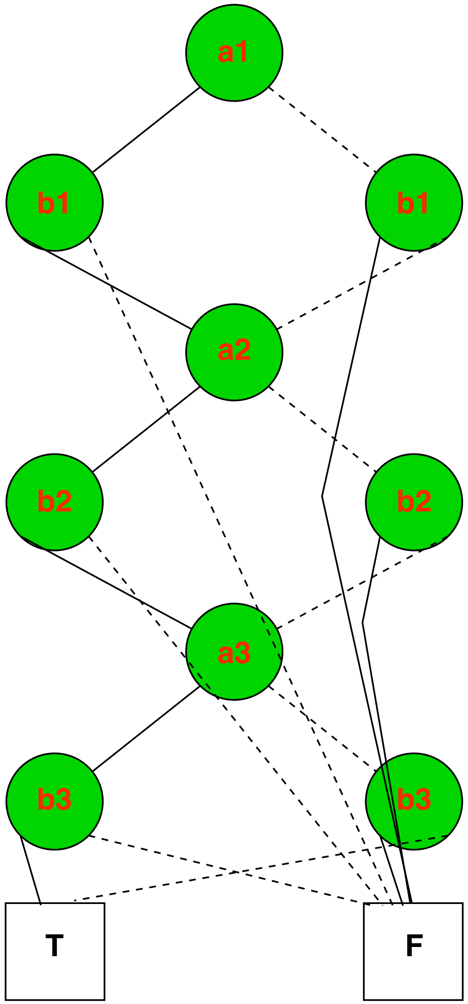
Content

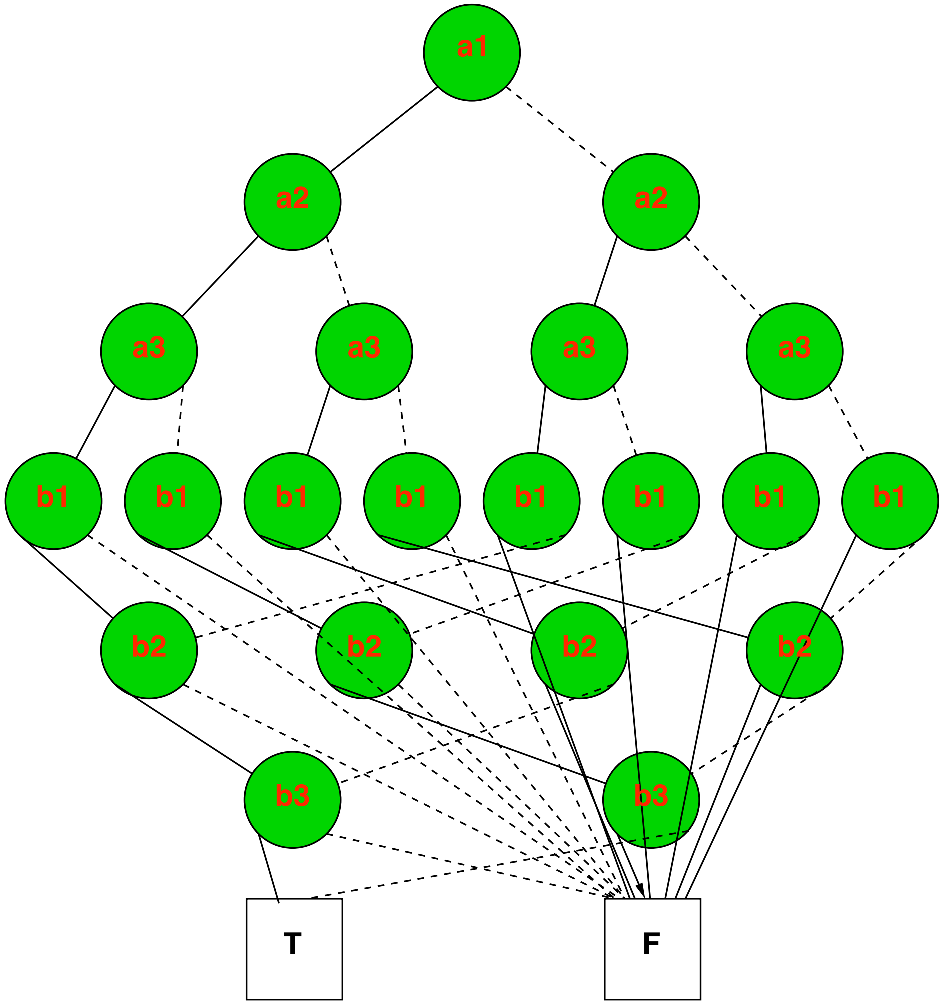
Ordered Binary Decision Diagrams (OBDDs) [Bryant, '85]

Canonical representation of Boolean formulas

- "If-then-else" binary DAGs with two leaves: 1 and 0
- Paths leading to 1 represent models
  Paths leading to 0 represent counter-models
- Variable ordering $A_1, A_2, ..., A_n$ imposed a priori.

OBDD - Examples

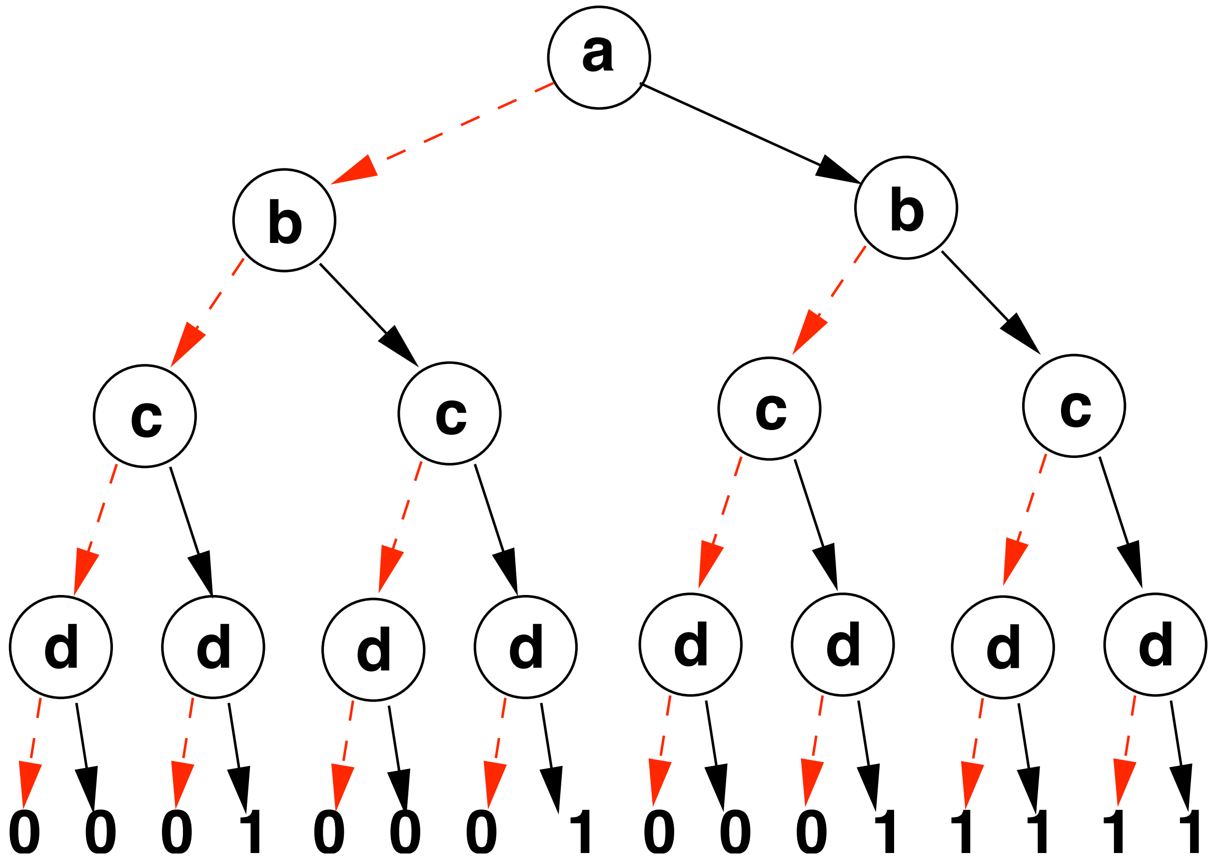




OBDDs of $(a_1 \leftrightarrow b_1) \wedge (a_2 \leftrightarrow b_2) \wedge (a_3 \leftrightarrow b_3)$ with different variable orderings

Ordered Decision Trees

- Ordered Decision Tree: from root to leaves, variables are encountered always in the same order
- Example: Ordered Decision tree for $\varphi = (a \wedge b) \vee (c \wedge d)$

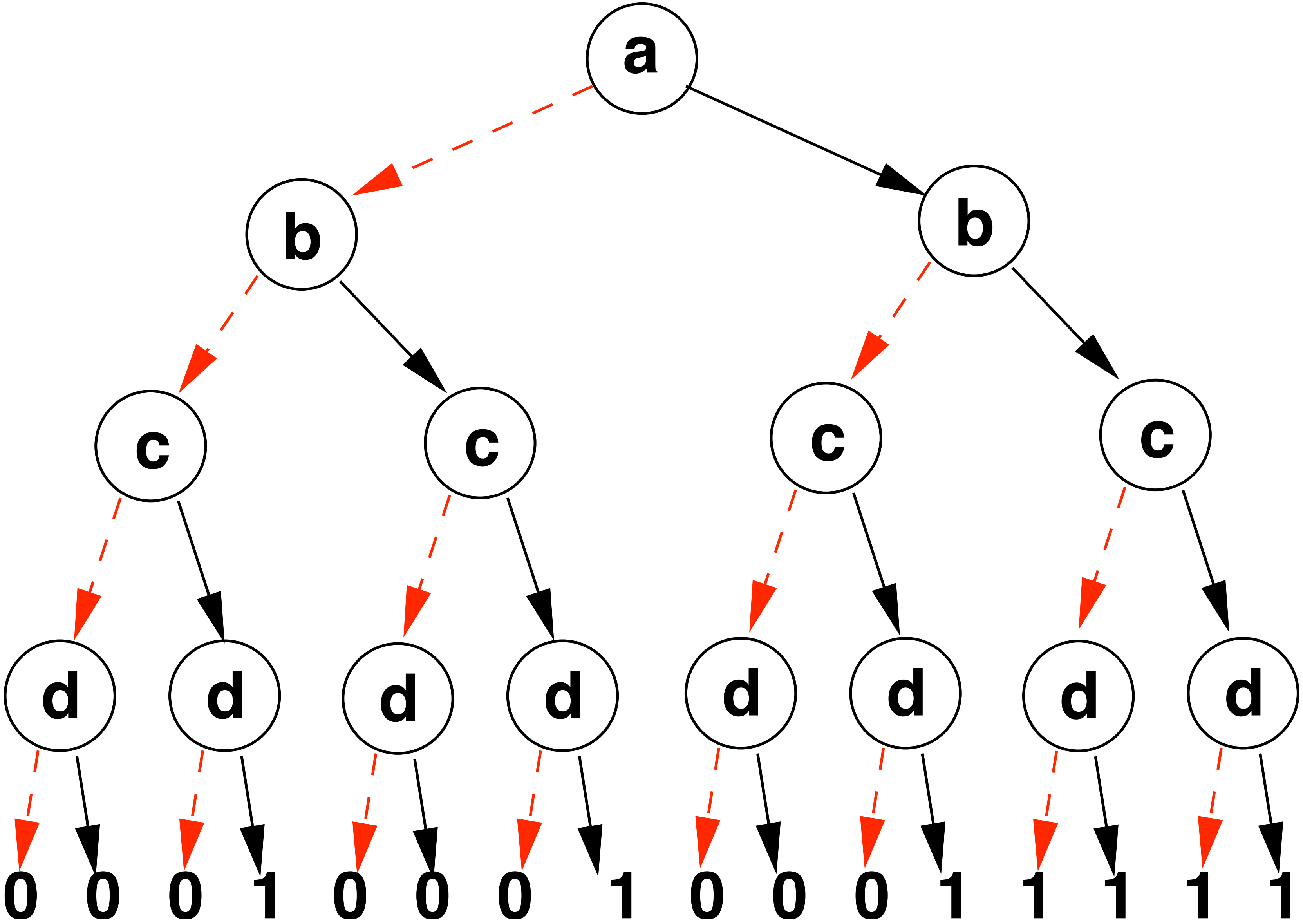


From Ordered Decision Trees to OBDD's: reductions

▷ Recursive applications of the following reductions:

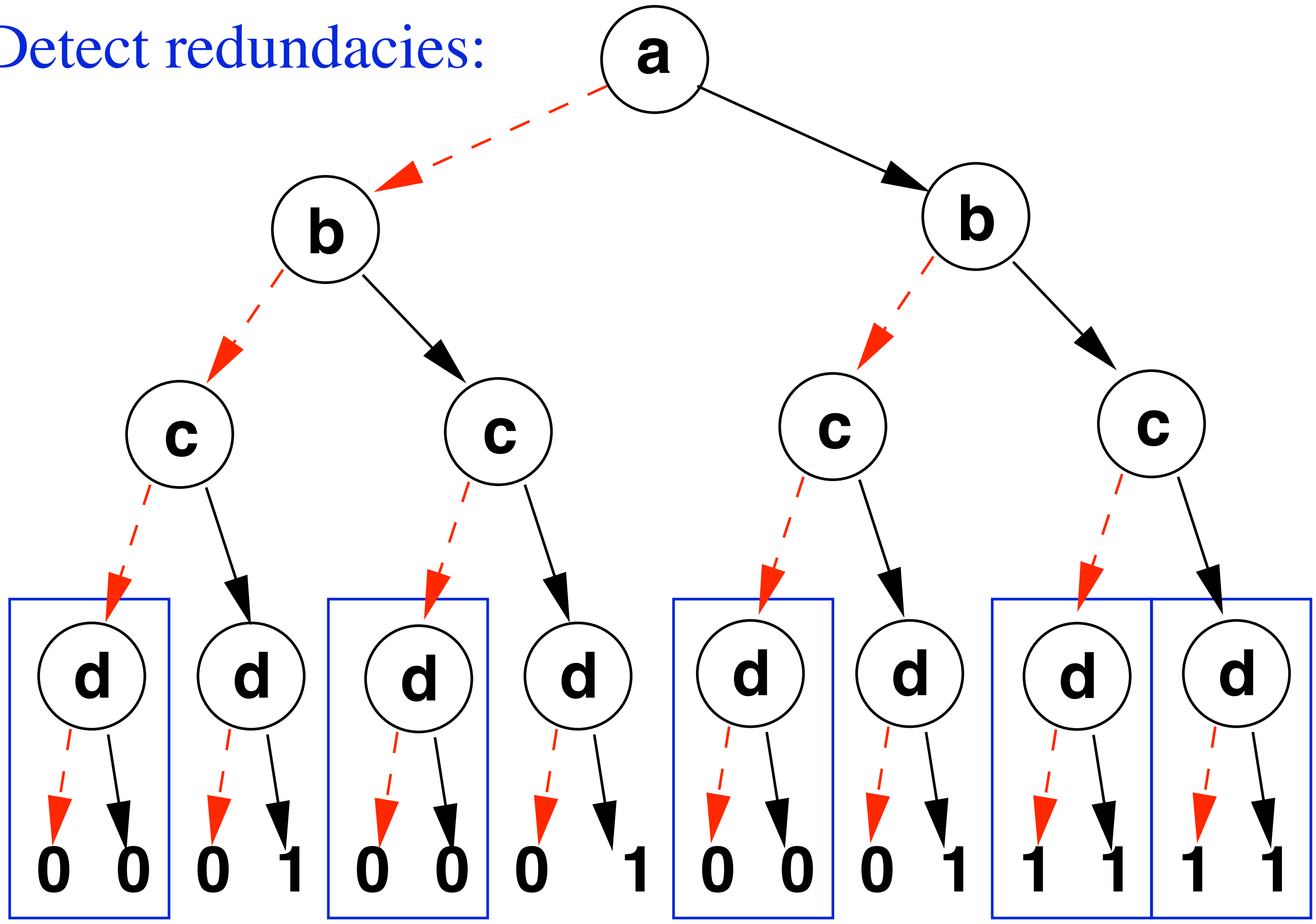
- share subnodes: point to the same occurrence of a subtree
- remove redundancies: nodes with same left and right children can be eliminated

Reduction: example



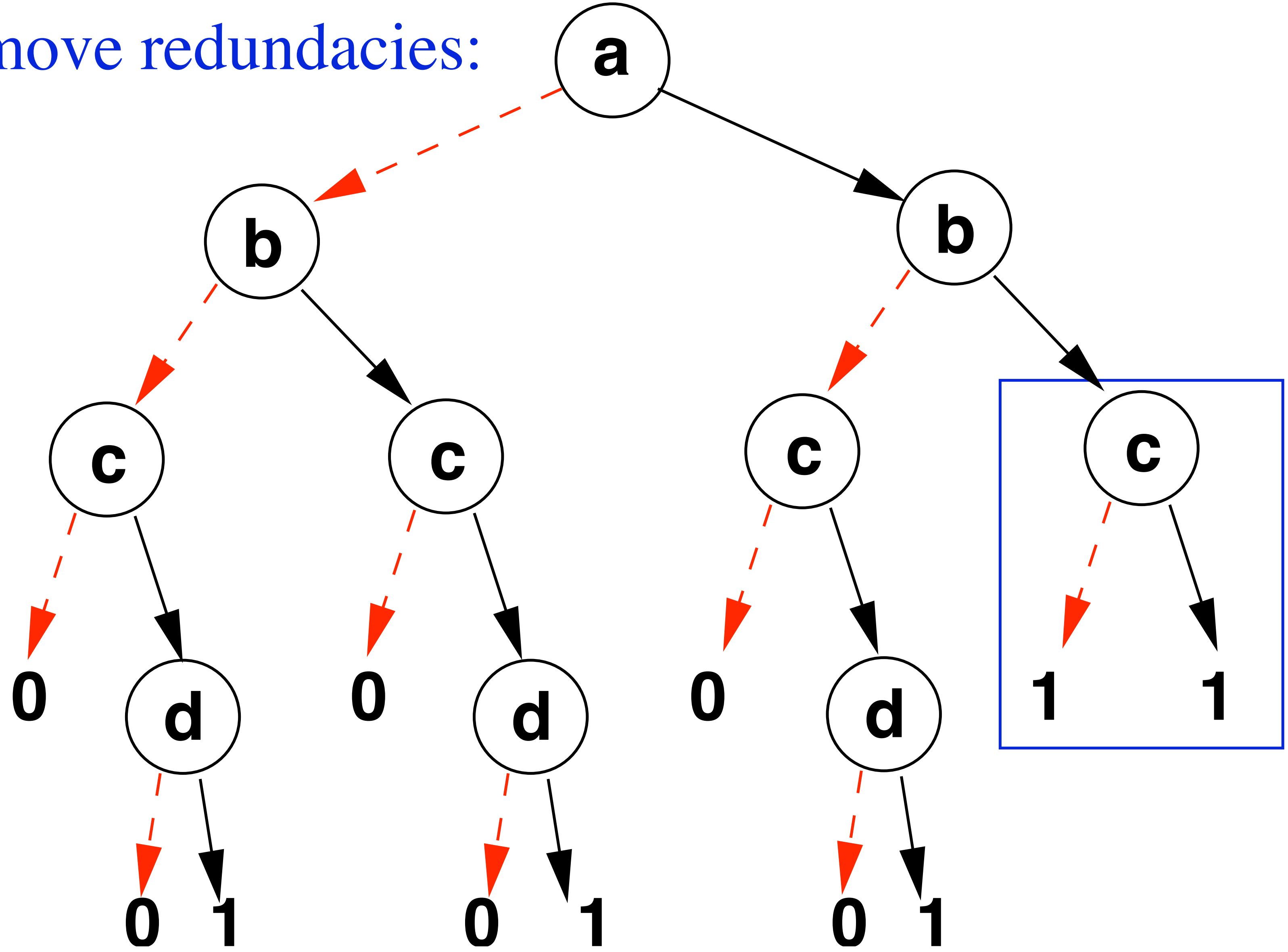
Reduction: example [cont.]

Detect redundacies:



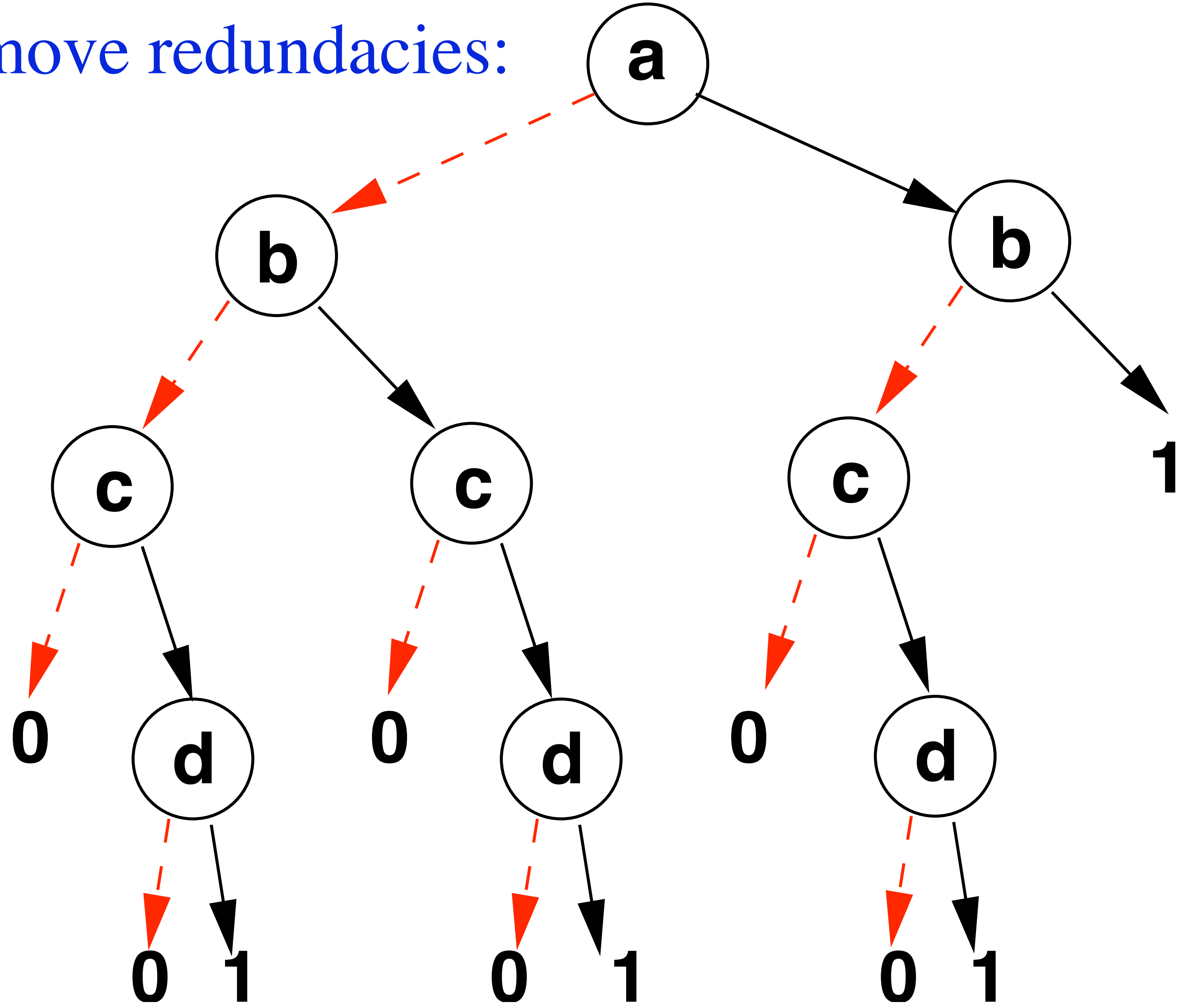
Reduction: example [cont.]

Remove redundacies:



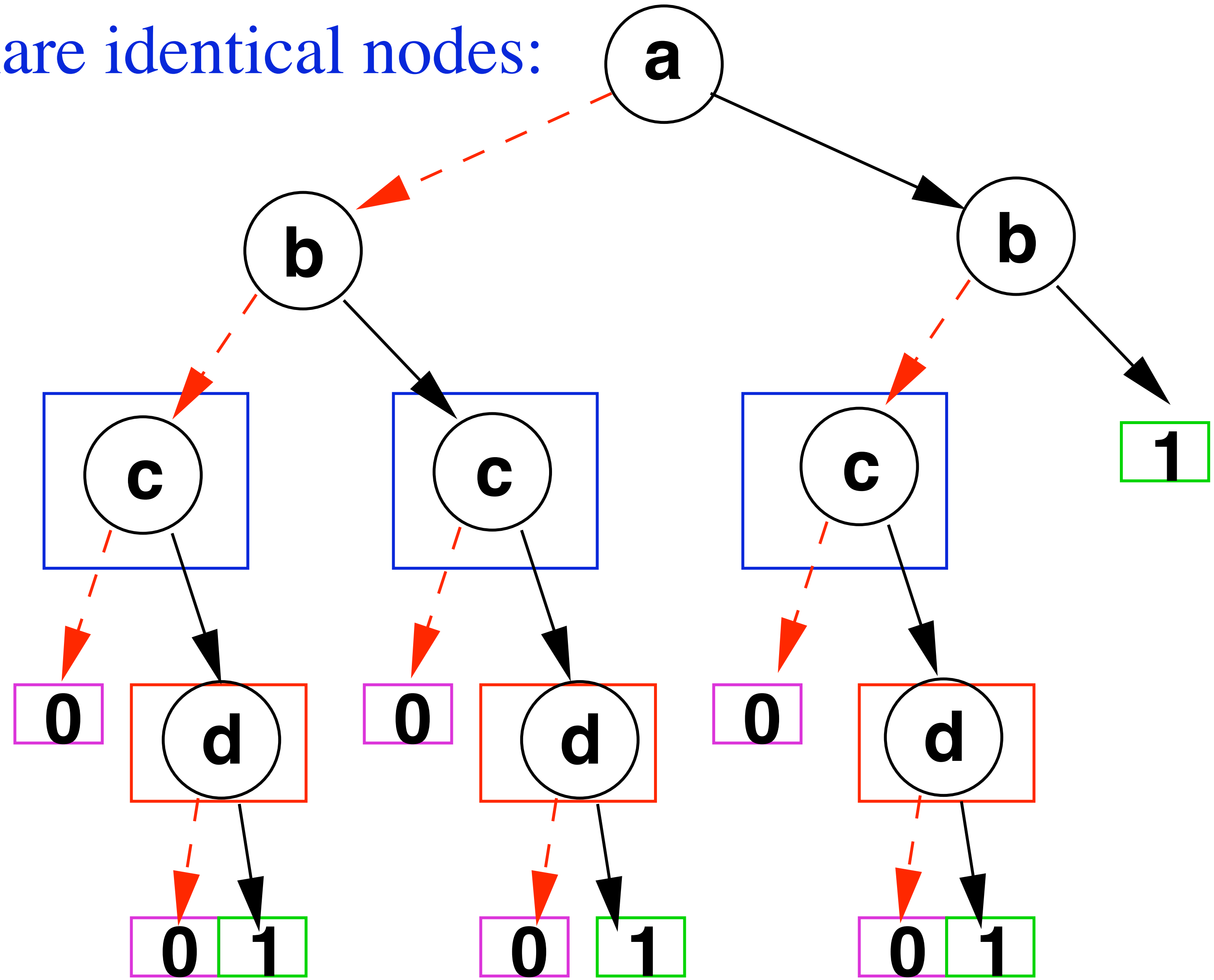
Reduction: example [cont.]

Remove redundacies:



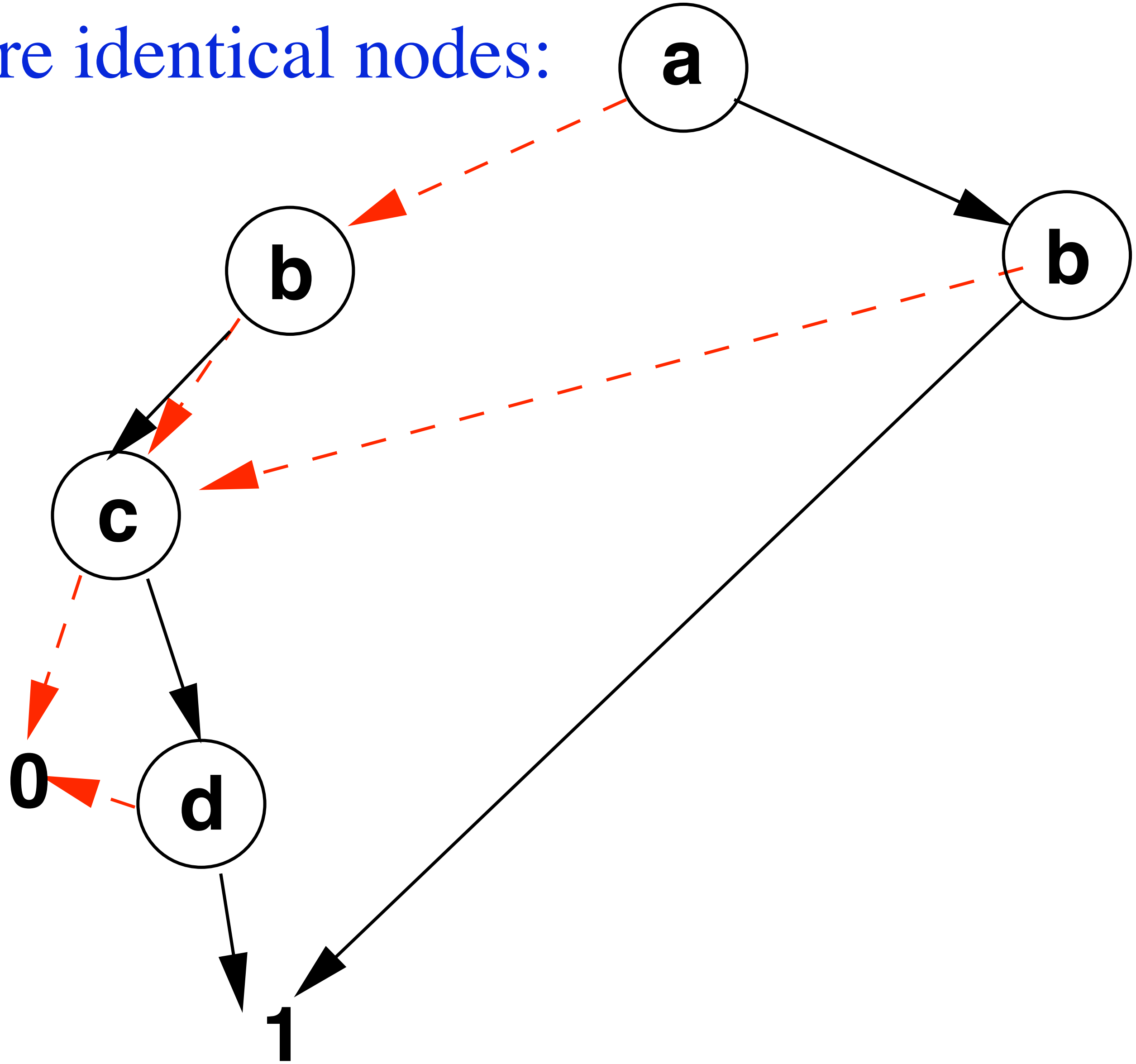
Reduction: example [cont.]

Share identical nodes:



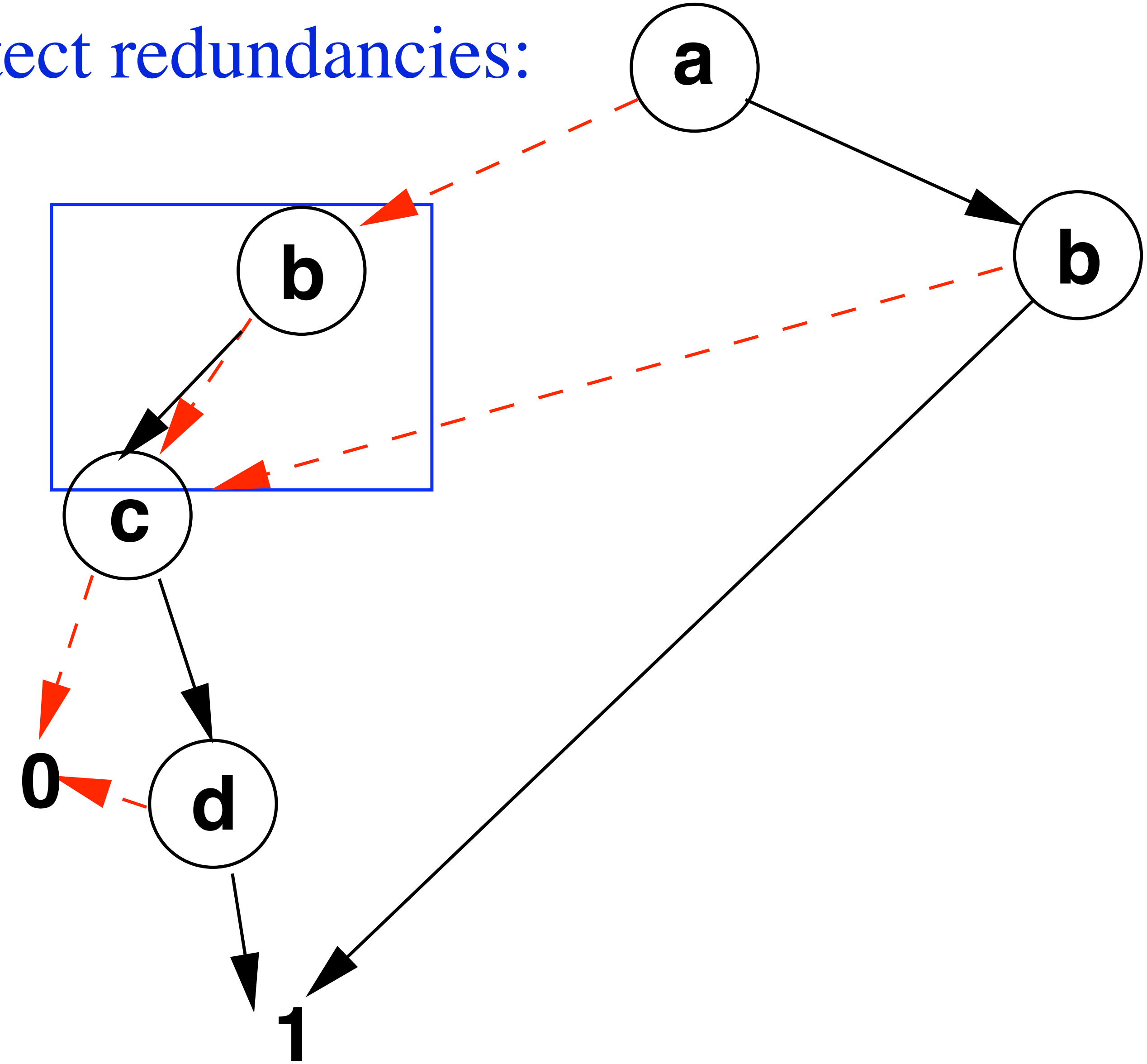
Reduction: example [cont.]

Share identical nodes:



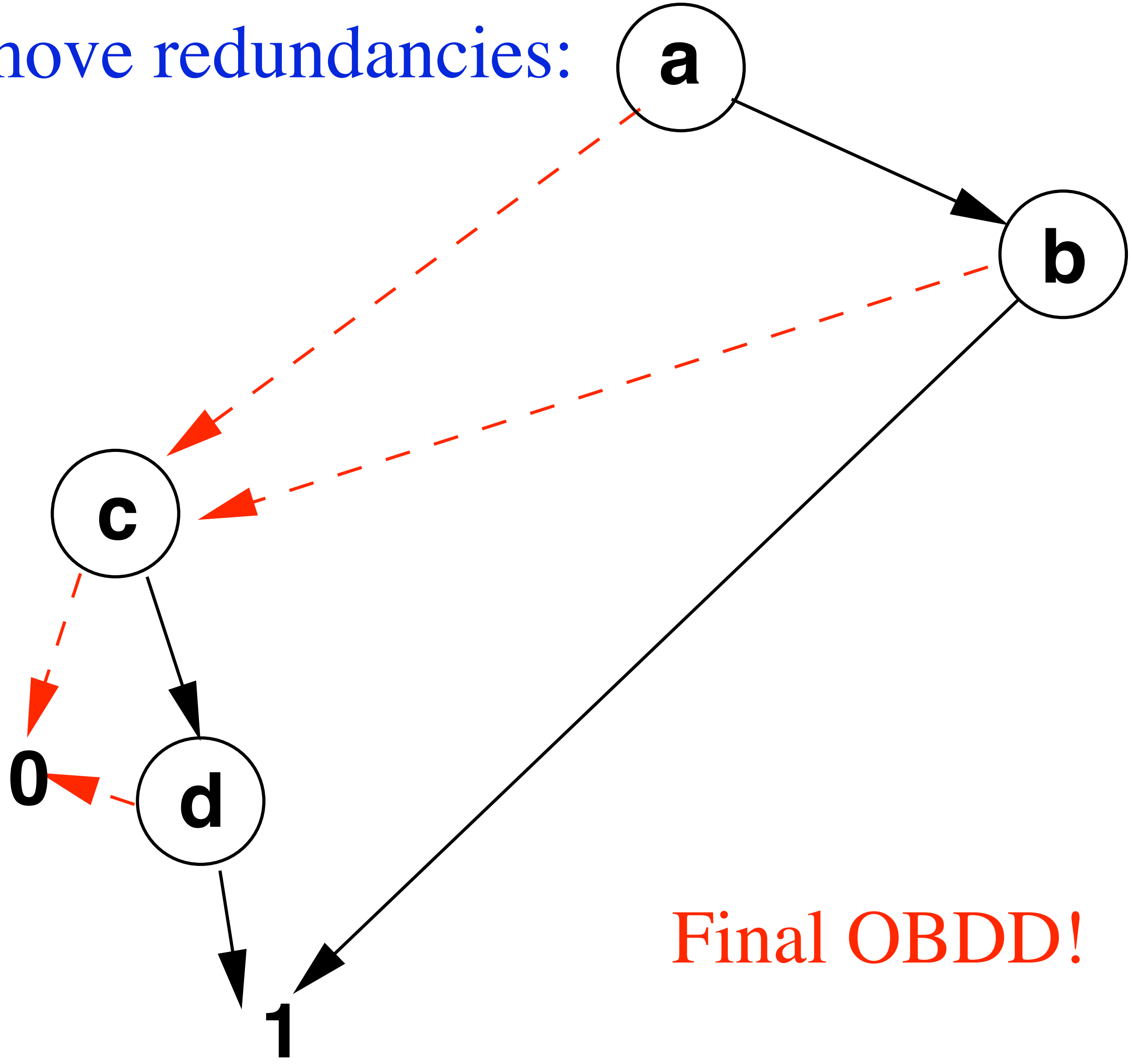
Reduction: example [cont.]

Detect redundancies:



Reduction: example [cont.]

Remove redundancies:



Final OBDD!

Recursive structure of an OBDD

- $OBDD(\top, \{...\}) = 1$,
- $OBDD(\bot, \{...\}) = 0$,
- $OBDD(\varphi, \{A_1, A_2, ..., A_n\}) =$
  $if\ A_1$
  $then\ OBDD(\varphi[A_1|\top], \{A_2, ..., A_n\})$
  $else\ OBDD(\varphi[A_1|\bot], \{A_2, ..., A_n\})$

Incrementally building an OBDD

- $obdd\_build(\top, \{...\}) := 1,$
- $obdd\_build(\bot, \{...\}) := 0,$
- $obdd\_build((\varphi_1 \; op \; \varphi_2), \{A_1, ..., A_n\}) :=$

$$
\begin{array}{l}
reduce(\\
\quad apply(\quad op, \\
\qquad\qquad obdd\_build(\varphi_1, \{A_1, ..., A_n\}), \quad op \in \{\wedge, \vee, \rightarrow, \leftrightarrow\} \\
\qquad\qquad obdd\_build(\varphi_2, \{A_1, ..., A_n\}) \\
)\,)
\end{array}
$$

Incrementally building an OBDD: reduce

reduce traverses the OBDD from the leaves to the root assigning an identifier id(n) to each node n. Two nodes have the same id if the sub-OBDD describe the same Boolean function:

- $id(0) = 0$, $id(1) = 1$
- if $id(lo(n)) = id(hi(n))$, then $id(n) = id(lo(n))$
- if for the node n there exists another node m with the same var and having $id(lo(n)) = id(lo(m))$ and $id(hi(n)) = id(hi(m))$, then $id(n) = id(m)$
- otherwise, $id(n)$ is a new integer.

Incrementally building an OBDD: apply

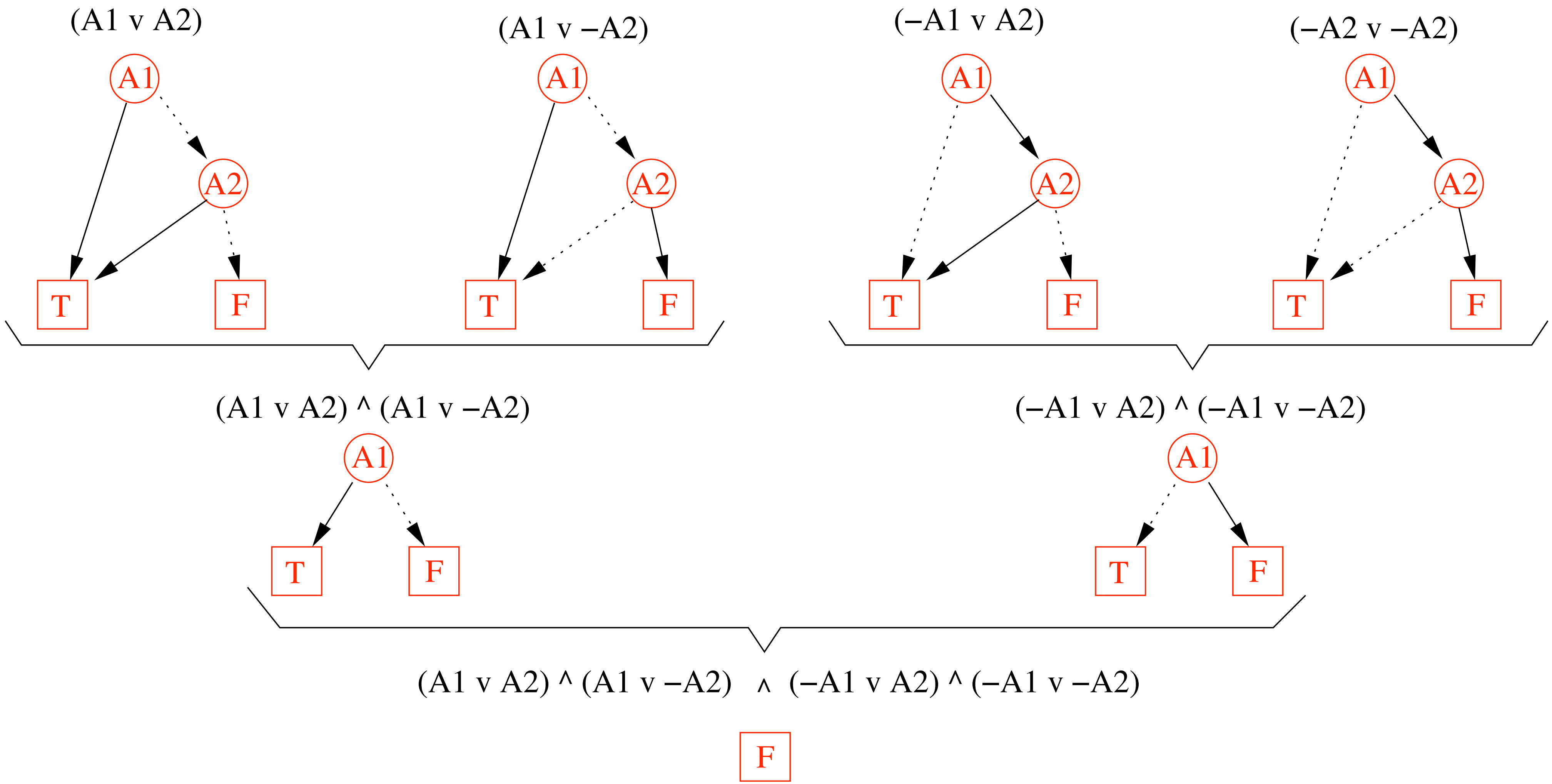
- $apply\ (op,\ O_i, O_j)) := (O_i\ op\ O_j)$
- $apply\ (op,\ ite(A, \varphi^\top, \varphi^\bot), O) :=\ ite(A, apply\ (op,\ \varphi^\top, O), apply\ (op,\ \varphi^\bot, O))$
- $apply\ (op,\ O,\ ite(A, \varphi^\top, \varphi^\bot)) :=\ ite(A, apply\ (op,\ O, \varphi^\top), apply\ (op,\ O, \varphi^\bot))$
- $apply\ (op,\ ite(A_i, \varphi_i^\top, \varphi_i^\bot),\ ite(A_j, \varphi_j^\top, \varphi_j^\bot)) :=$

  $$\begin{array}{lll} \textbf{if}\ (A_i < A_j) & ite(A_i, & apply\ (op, \varphi_i^\top, ite(A_j, \varphi_j^\top, \varphi_j^\bot)), \\ & & apply\ (op, \varphi_i^\bot, ite(A_j, \varphi_j^\top, \varphi_j^\bot)))) \\ \textbf{if}\ (A_i > A_j) & ite(A_j, & apply\ (op, ite(A_i, \varphi_i^\top, \varphi_i^\bot), \varphi_j^\top), \\ & & apply\ (op, ite(A_i, \varphi_i^\top, \varphi_i^\bot), \varphi_j^\bot))) \\ \textbf{if}\ (A_i = A_j) & ite(A_i, & apply\ (op, \varphi_i^\top, \varphi_j^\top), \\ & & apply\ (op, \varphi_i^\bot, \varphi_j^\bot)) \end{array}$$

  $O, O_i, O_j \in \{1, 0\}$, $op \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$
- $\neg$? ... reverse the leaves

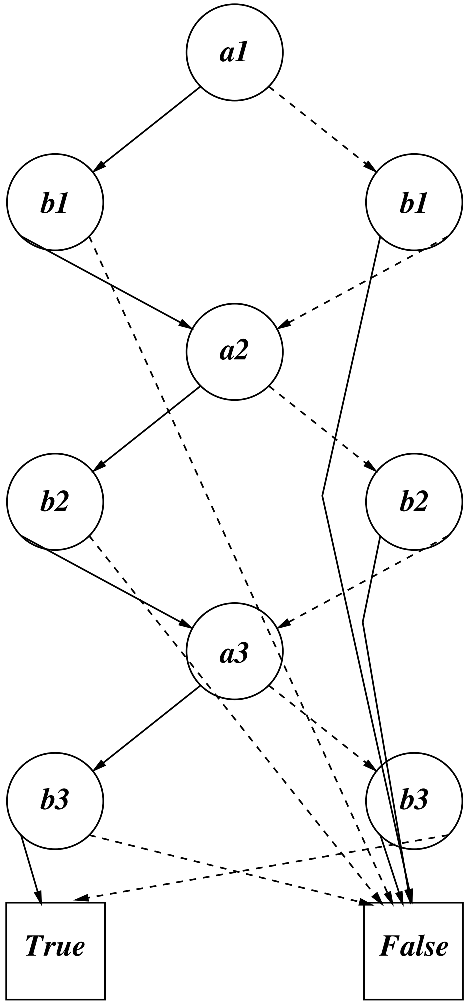
OBBD incremental building – example

$$\varphi = (A_1 \vee A_2) \wedge (A_1 \vee \neg A_2) \wedge (\neg A_1 \vee A_2) \wedge (\neg A_1 \vee \neg A_2)$$

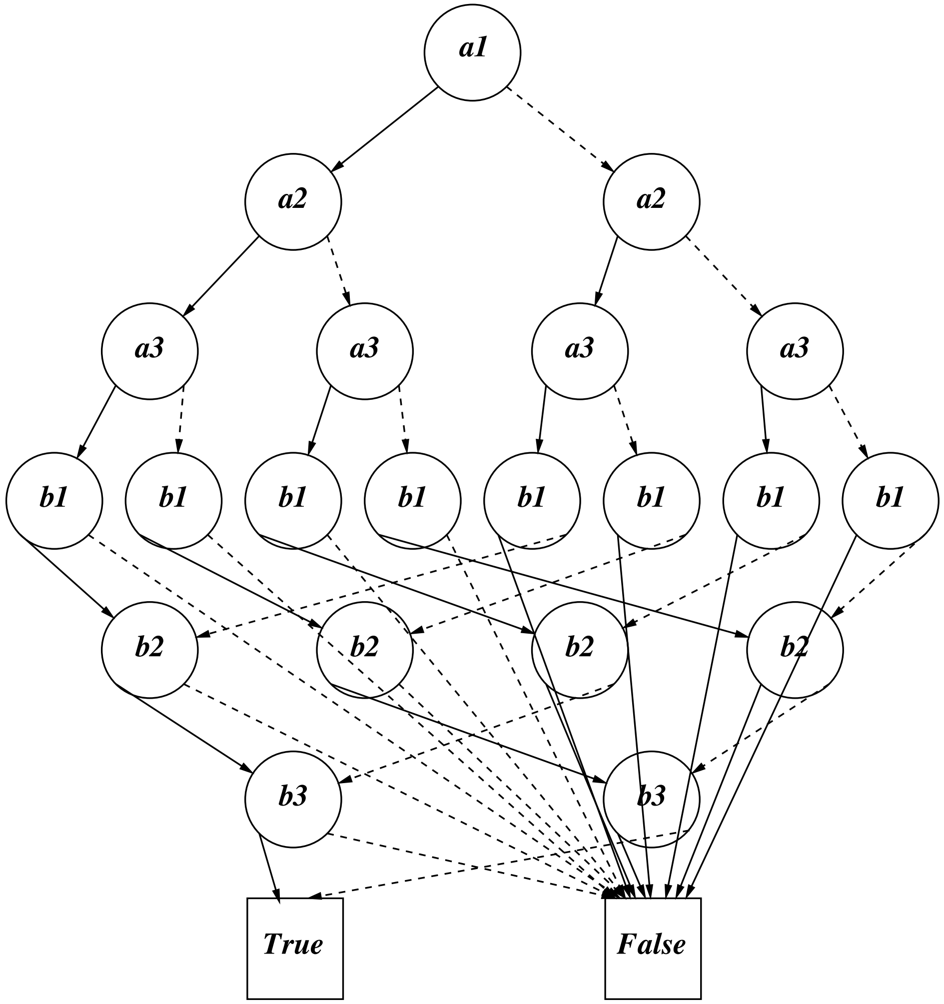


Critical choice of variable Orderings in OBDD's

$$\varphi = (a1 \leftrightarrow b1) \wedge (a2 \leftrightarrow b2) \wedge (a3 \leftrightarrow b3)$$



Linear size



Exponential size

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:

$$\varphi_1 \leftrightarrow \varphi_2 \iff OBDD(\varphi_1) = OBDD(\varphi_2)$$

- equivalence check requires constant time!
  $\Longrightarrow$validity check requires constant time! $(\varphi \leftrightarrow \top)$
  $\Longrightarrow$(un)satisfiability check requires constant time! $(\varphi \leftrightarrow \bot)$
- 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

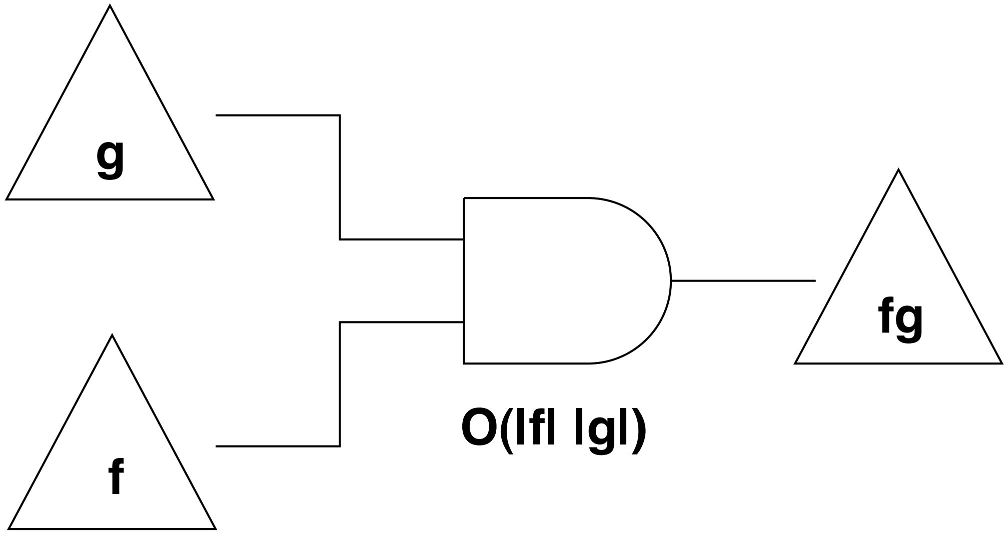
- ▷ The size of OBDD's may grow exponentially wrt. the number of variables in worst-case
- ▷ Consequence of the canonicity of OBDD's (unless P $=$ co-NP)
- ▷ Example: there exist no polynomial-size OBDD representing the electronic circuit of a bitwise multiplier
- ▷ N.B.: the size of intermediate OBDD's may be bigger than that of the final one (e.g., inconsistent formula)

Useful Operations over OBDDs

▷ the equivalence check between two OBDDs is simple

- are they the same OBDD? ($\Longrightarrow$constant time)

▷ the size of a boolean composition is up to the product of the size of the operands: $|f\ op\ g| = O(|f| \cdot |g|)$



(but typically much smaller on average).

Boolean quantification

- If v is a boolean variable, then

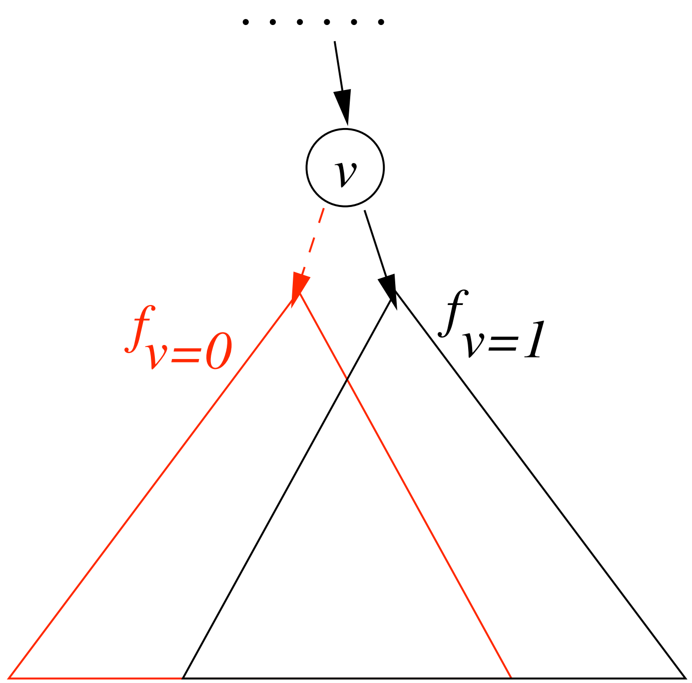
$$\begin{aligned} \exists v.f &:= f|_{v=0} \vee f|_{v=1} \\ \forall v.f &:= f|_{v=0} \wedge f|_{v=1} \end{aligned}$$

- v does no more occur in $\exists v.f$ and $\forall v.f$!!
- 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 $tvalue \in \{\top, \bot\}$, $\mu \cup \{v := tvalue\} \models f$
- Multi-variable quantification: $\exists (w_1, \ldots, w_n).f \ := \ \exists w_1 \ldots \exists w_n.f$
- Example: $\exists (b, c).((a \wedge b) \vee (c \wedge d)) \ = \ a \vee d$
- 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



$\Longrightarrow$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).

Content

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,. . . , 01111 $\}$ still represented as "first bit false": $\neg x_1$

Kripke Structures in Propositional Logic

▷ 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 $(\neg x_1 \wedge x_2 \wedge \neg x_3 \wedge \neg x_4)$
- 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

- any 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
  $\Longrightarrow$ Typically Q can be encoded by much smaller formulas than $\bigvee_{s \in Q} \xi(s)$!
- Example: $Q = \{$ 00000, 00001, 00010, 00011, 00100, 00101, 00110, 00111,. . . , 01111 $\}$ represented as "first bit false": $\neg x_1$

$$\bigvee_{s \in Q} \xi(s) = \left.\begin{array}{l} (\neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_5) \vee \\ (\neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge x_5) \vee \\ (\neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge x_4 \wedge \neg x_5) \vee \\ \ldots \\ (\neg x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5) \end{array}\right\} 2^4 disjuncts$$

Symbolic Representation of Set Operators

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

Symbolic Representation of Transition Relations

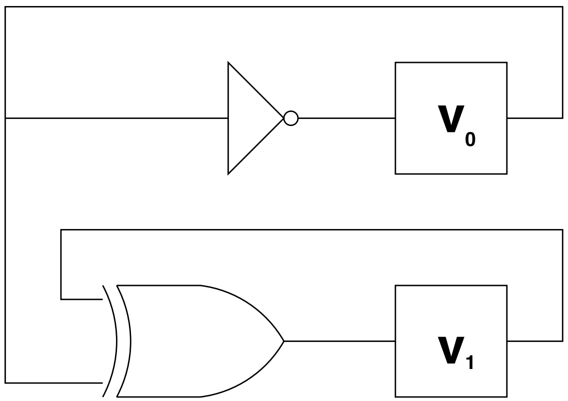
- The transition relation R is a set of pairs of states: $R \subseteq S \times S$
- A transition is a pair of states (s, s')
- A new vector of variables V' (the next state vector) represents the value of variables after the transition has occurred
- $\xi(s, s')$ defined as $\xi(s) \wedge \xi'(s')$
-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

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

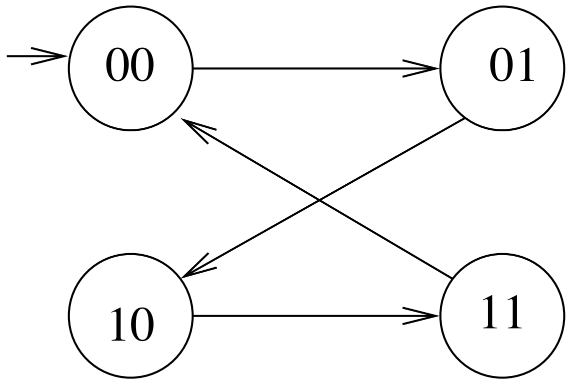
- Note: Each formula equivalent to $\xi(R)$ is a representation of R
  $\Longrightarrow$ Typically R can be encoded by a much smaller formula than $\bigvee_{(s,s')\in R} \xi(s) \wedge \xi(s')$!

Example: a simple counter

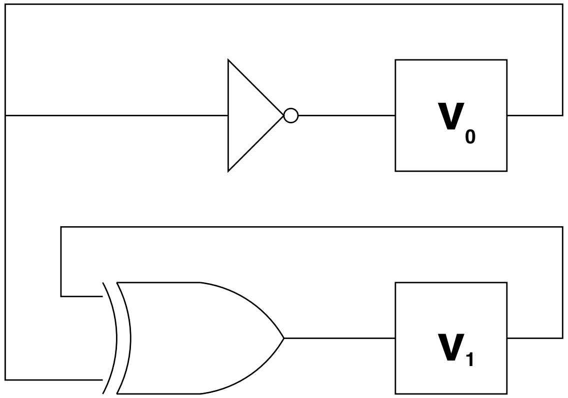
```
MODULE main
 VAR
   v0     : boolean;
   v1     : boolean;
   out    : 0..3;

 ASSIGN
   init(v0) := 0;
   next(v0) := !v0;

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

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



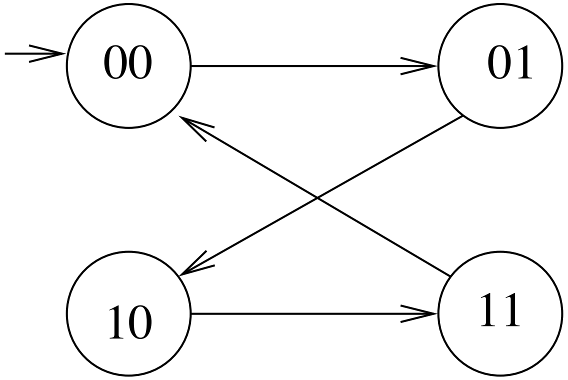
v_1	v_0	v'_1	v'_0
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0



Example: a simple counter [cont.]



v_1	v_0	v'_1	v'_0
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

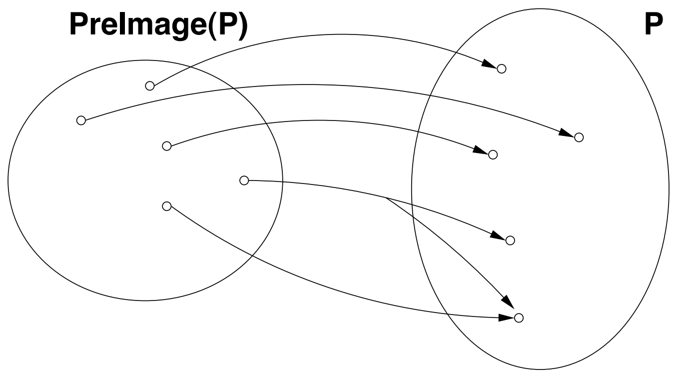


$$\xi(R) = (v'_0 \leftrightarrow \neg v_0) \wedge (v'_1 \leftrightarrow v_0 \bigoplus v_1)$$

$$\bigvee_{(s,s')\in R} \xi(s) \wedge \xi(s') = (\neg v_1 \wedge \neg v_0 \wedge \neg v'_1 \wedge v'_0) \vee (\neg v_1 \wedge v_0 \wedge v'_1 \wedge \neg v'_0) \vee (v_1 \wedge \neg v_0 \wedge v'_1 \wedge v'_0) \vee (v_1 \wedge v_0 \wedge \neg v'_1 \wedge \neg v'_0)$$

Pre-Image

▷ (Backward) pre-image of a set:



▷ Evaluate one-shot all transitions ending in the states of the set

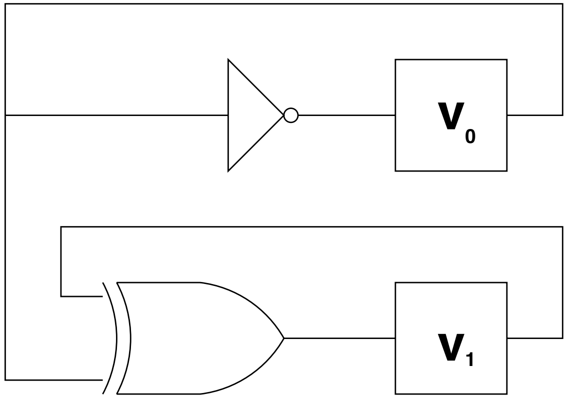
▷ Set theoretic view:
$PreImage(P, R) := \{s |$ for some $s' \in P, (s, s') \in R\}$

▷ Logical view: $\xi(PreImage(P, R)) \ := \ \exists V'.(\xi(P)[V'] \wedge \xi(R)[V, V'])$

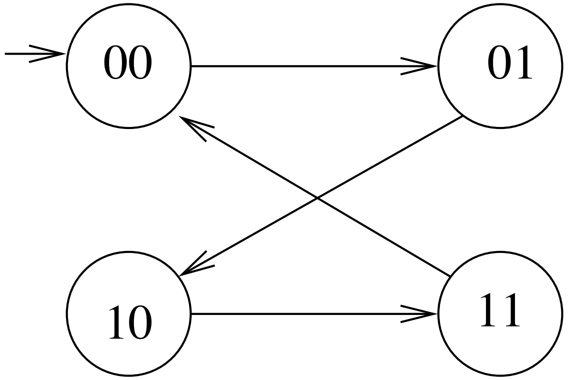
▷ μ over V is s.t $\mu \models \exists V'.(\xi(P)[V'] \wedge \xi(R)[V, V'])$ iff,
for some μ' over V', we have: $\mu \cup \mu' \models (\xi(P)[V'] \wedge \xi(R)[V, V'])$,
i.e., $\mu' \models \xi(P)[V']$ and $\mu \cup \mu' \models \xi(R)[V, V'])$

- Intuition: $\mu \Longleftrightarrow s$, $\mu' \Longleftrightarrow s'$, $\mu \cup \mu' \Longleftrightarrow \langle s, s' \rangle$

Example: simple counter



v_1	v_0	v_1'	v_0'
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

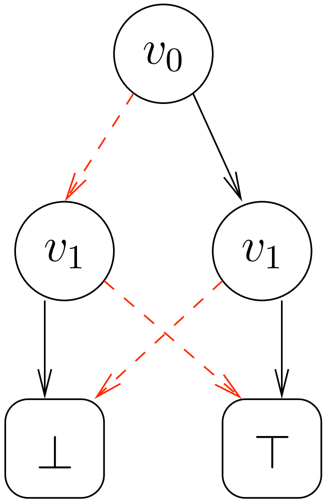


$$\xi(R) = (v_0' \leftrightarrow \neg v_0) \wedge (v_1' \leftrightarrow v_0 \bigoplus v_1)$$

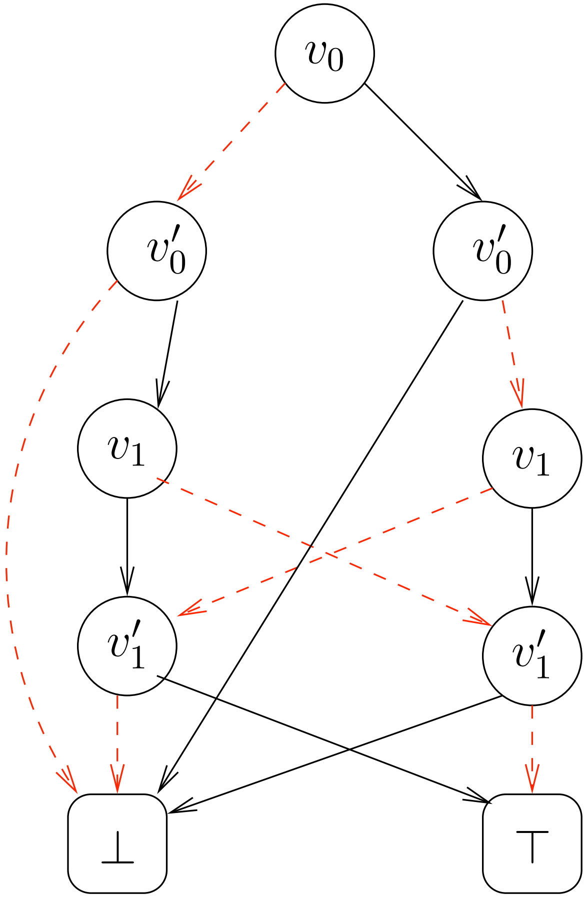
$\xi(P) := (v_0 \leftrightarrow v_1)$ (i.e., $P = \{00, 11\}$)

$$\begin{aligned}
& \xi(PreImage(P,R)) = \exists V'.(\xi(P)[V'] \wedge \xi(R)[V,V']) \\
= \ & \exists v_0' v_1'.((v_0' \leftrightarrow v_1') \wedge (v_0' \leftrightarrow \neg v_0) \wedge (v_1' \leftrightarrow v_0 \bigoplus v_1)) \\
= \ & \underbrace{(\neg v_0 \wedge v_0 \bigoplus v_1)}_{v_0'=\top, v_1'=\top} \vee \underbrace{\bot}_{v_0'=\top, v_1'=\bot} \vee \underbrace{\bot}_{v_0'=\bot, v_1'=\top} \vee \underbrace{(v_0 \wedge \neg(v_0 \bigoplus v_1))}_{v_0'=\bot, v_1'=\bot} \\
= \ & v_1 \quad (i.e., \ \{10, 11\})
\end{aligned}$$

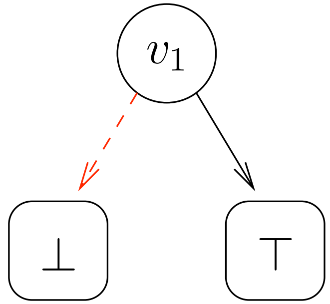
Pre-Image [cont.]



$\xi(P) = v_0 \leftrightarrow v_1$



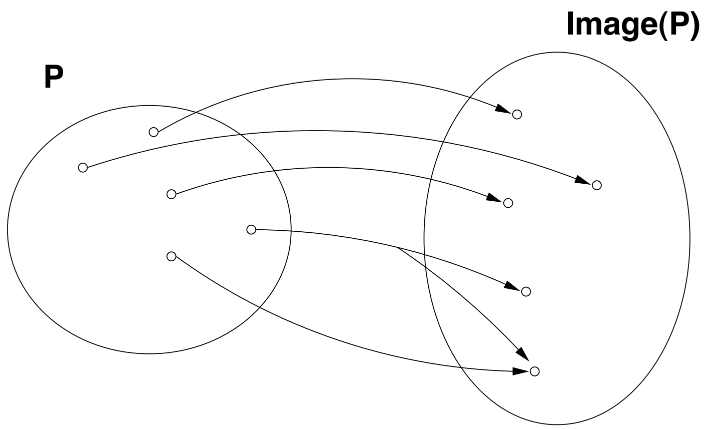
$\xi(R) = (v'_0 \leftrightarrow \neg v_0) \wedge (v'_1 \leftrightarrow v_0 \bigoplus v_1)$



$\xi(PreImage(P, R)) =$
$\exists V'.((v'_0 \leftrightarrow v'_1) \wedge (v'_0 \leftrightarrow \neg v_0) \wedge (v'_1 \leftrightarrow v_0 \bigoplus v_1)) =$
v_1

Forward Image

- Forward image of a set:



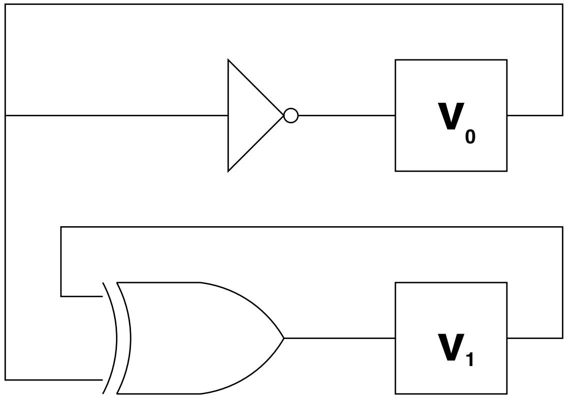
- Evaluate one-shot all transitions from the states of the set
- Set theoretic view

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

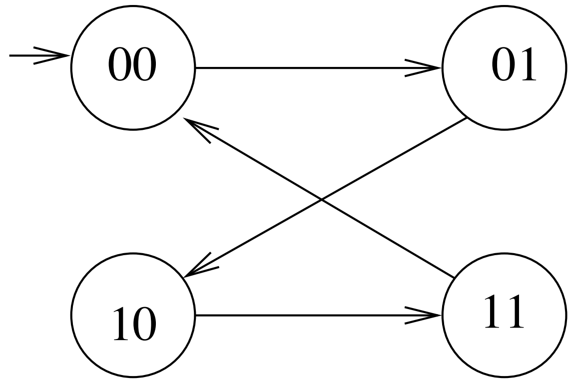
- Logical Characterization

$$\xi(Image(P, R)) \;:=\; \exists V.(\xi(P)[V] \wedge \xi(R)[V, V'])$$

Example: simple counter



v_1	v_0	v'_1	v'_0
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

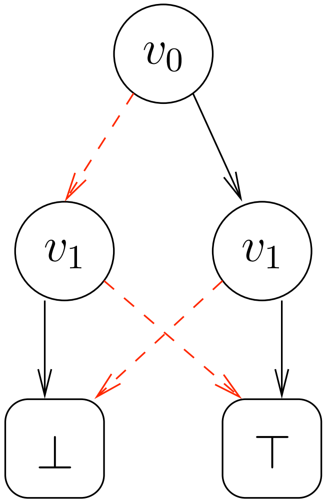


$$\xi(R) = (v'_0 \leftrightarrow \neg v_0) \wedge (v'_1 \leftrightarrow v_0 \bigoplus v_1)$$

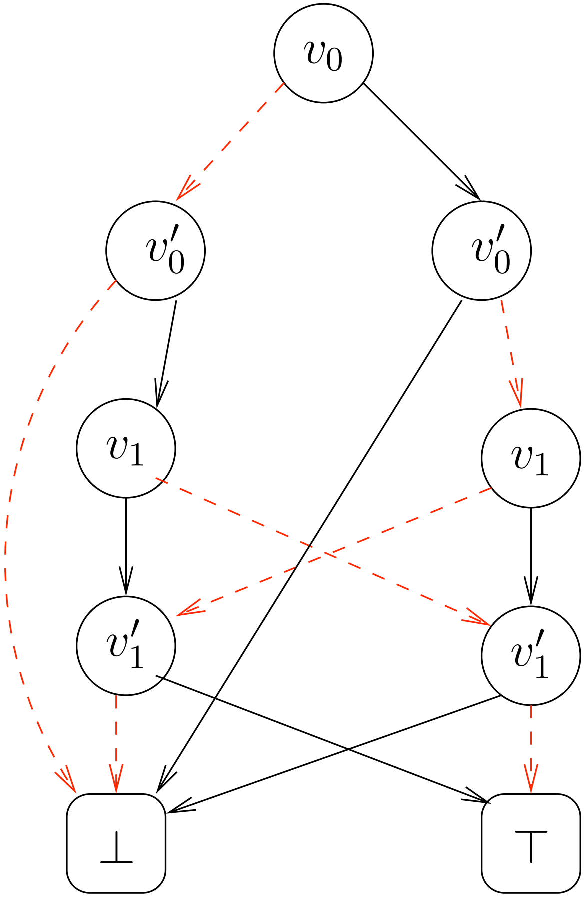
$\xi(P) := (v_0 \leftrightarrow v_1)$ (i.e., $P = \{00, 11\}$)

$$\begin{aligned} \xi(Image(P,R)) &= \exists V.(\xi(P)[V] \wedge \xi(R)[V,V']) \\ &= \exists V.((v_0 \leftrightarrow v_1) \wedge (v'_0 \leftrightarrow \neg v_0) \wedge (v'_1 \leftrightarrow v_0 \bigoplus v_1)) \\ &= \ldots \\ &= \neg v'_1 \qquad (i.e.,\ \{00, 01\}) \end{aligned}$$

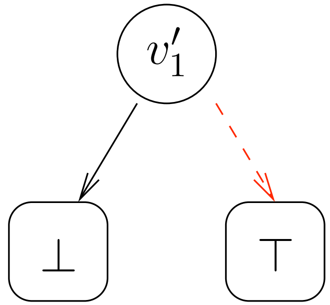
Forward Image [cont.]



$\xi(P) = v_0 \leftrightarrow v_1$



$\xi(R) = (v'_0 \leftrightarrow \neg v_0) \wedge (v'_1 \leftrightarrow v_0 \bigoplus v_1)$



$\xi(Image(P, R)) =$
$\exists V.((v_0 \leftrightarrow v_1) \wedge (v'_0 \leftrightarrow \neg v_0) \wedge (v'_1 \leftrightarrow v_0 \bigoplus v_1)) =$
$\neg v'_1$

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

Content

Symbolic CTL model checking

- Problem: $M \models \varphi$?,
  - $M = \langle S, I, R, L, AP \rangle$ being a Kripke structure and
  - φ 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 φ written in terms of $\neg$, $\wedge$, $\mathbf{EX}$, $\mathbf{EU}$, $\mathbf{EG}$
- A general M.C. algorithm (fix-point):
  1. represent I and R as boolean formulas $\xi(I), \xi(R)$
  2. for every $\varphi_i \in Sub(\varphi)$, find $\xi([\varphi_i])$
  3. Check if $\xi(I) \rightarrow \xi([\varphi])$
- Subformulas $Sub(\varphi)$ of φ are checked bottom-up
- $\xi([\varphi_i])$ computed directly, without computing $[\varphi_i]$ explicitly!!!
  - boolean operators handled directly by OBDDs
  - next temporal operators $\mathbf{EX}$: handled by symbolic PreImage computation
  - other temporal operators $\mathbf{EG}$, $\mathbf{EU}$: handled by fix-point symbolic computation

Symbolic Denotation of a CTL formula φ: $\xi([\varphi])$

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

$$
\begin{array}{lcl}
\xi([false]) & = & \bot \\
\xi([true]) & = & \top \\
\xi([p]) & = & p \\
\xi([\neg\varphi_1]) & = & \neg\xi([\varphi_1] \\
\xi([\varphi_1 \wedge \varphi_2]) & = & \xi([\varphi_1]) \wedge \xi([\varphi_2]) \\
\xi([\mathbf{EX}\varphi]) & = & \exists V'.(\ \xi([\varphi])[V'] \wedge \xi(R)[V, V']) \\
\xi([\mathbf{EG}\beta]) & = & \nu Z.(\ \xi([\beta]) \wedge \xi([\mathbf{EX}Z])\ ) \\
\xi([\mathbf{E}(\beta_1 \mathbf{U} \beta_2)]) & = & \mu Z.(\ \xi([\beta_2]) \vee (\xi([\beta_1]) \wedge \xi([\mathbf{EX}Z])\ )
\end{array}
$$

Notation: if X_1 and X_2 are OBDDs and op is a boolean operator, we write "X_1 op X_2" for "reduce(obdd_merge(op,X_1,X_2))"

General M.C. Procedure

OBDD Check(CTL_formula β) {
    if $(In\_OBDD\_Hash(\beta))$
        return $OBDD\_Get\_From\_Hash(\beta)$;
    case β **of**
    $true$: **return** $obdd\_true$;
    $false$: **return** $obdd\_false$;
    $\neg\beta_1$: **return** $\neg$ Check(β_1);
    $\beta_1 \wedge \beta_2$: **return** (Check(β_1) $\wedge$ Check(β_2));
    $\mathbf{EX}\beta_1$: **return** PreImage(Check(β_1));
    $\mathbf{EG}\beta_1$: **return** Check_EG(Check(β_1));
    $\mathbf{E}(\beta_1\mathbf{U}\beta_2)$: **return** Check_EU(Check(β_1),Check(β_2));
}

PreImage

OBDD PreImage(**OBDD** X) {
    return $\exists V'.(\ X[V'] \wedge \xi(R)[V, V'])$;
}

Check_EG

OBDD Check_EG(**OBDD** X) {

$Y' := X;\ j := 1;$

repeat

$Y := Y';\ j := j + 1;$

$Y' := Y \wedge PreImage(Y));$

until $(Y' \leftrightarrow Y)$;

return Y;

}

Check_EU

```
OBDD Check_EU(OBDD X1,X2) {
    Y' := X2; j := 1;
    repeat
        Y := Y'; j := j + 1;
        Y' := Y ∨ (X1 ∧ PreImage(Y));
    until (Y' ↔ Y);
return Y;
}
```


Fair CTL MC: Emerson-Lei Algorithm

```
OBDD Check_FairEG(OBDD X) {
    Z':= X;
    repeat
        Z:= Z';
        for each F_i in FT
            Y:= Check_EU(Z,F_i∧Z);
            Z':= Z' ∧ PreImage(Y));
        end for;
    until (Z' ↔ Z);
    return Z;
}
```


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
  $\Longrightarrow$problems of up to 10^{120} states handled!!

Content

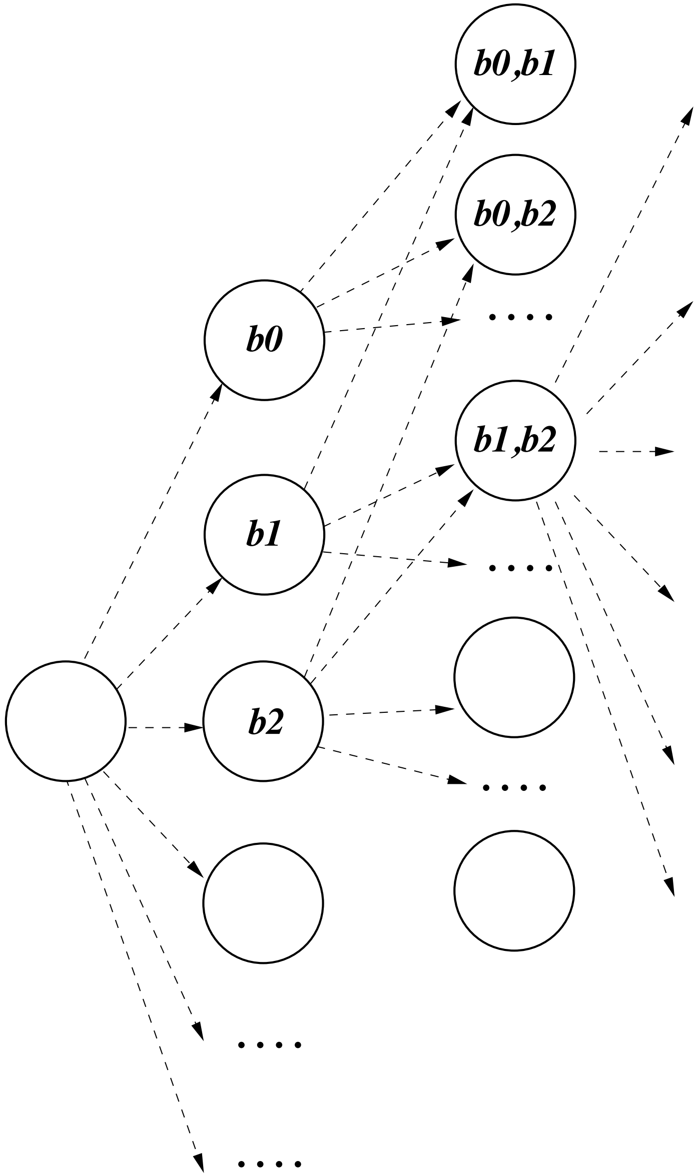
A simple example

```
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 $b0, b1, ...$
- ▷ 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

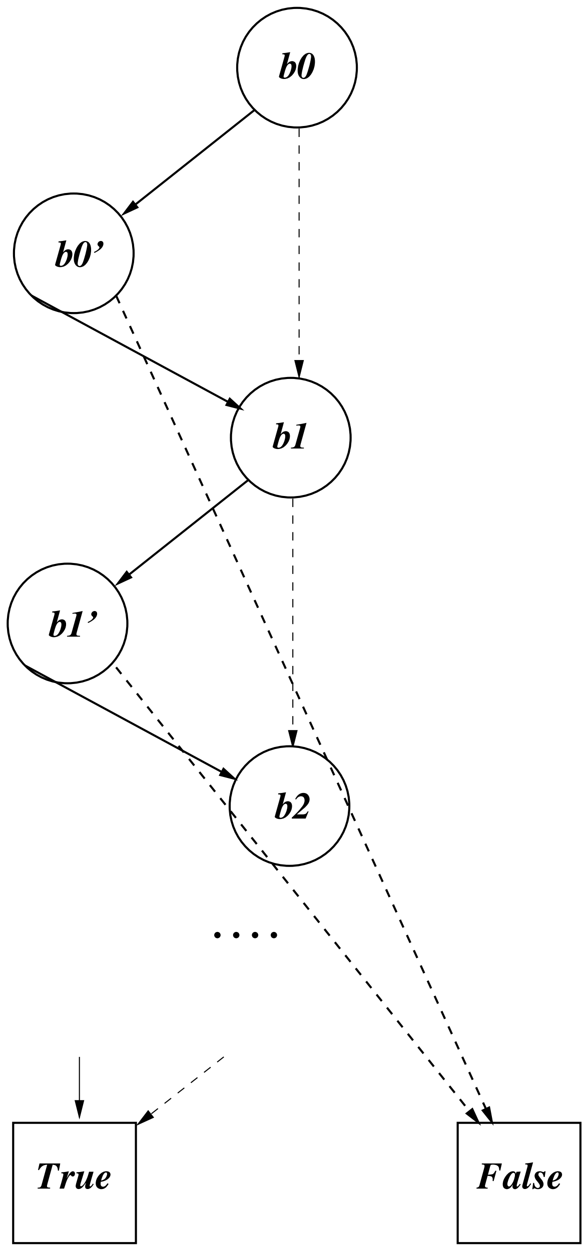


(transitive trans. omitted)

2^N STATES

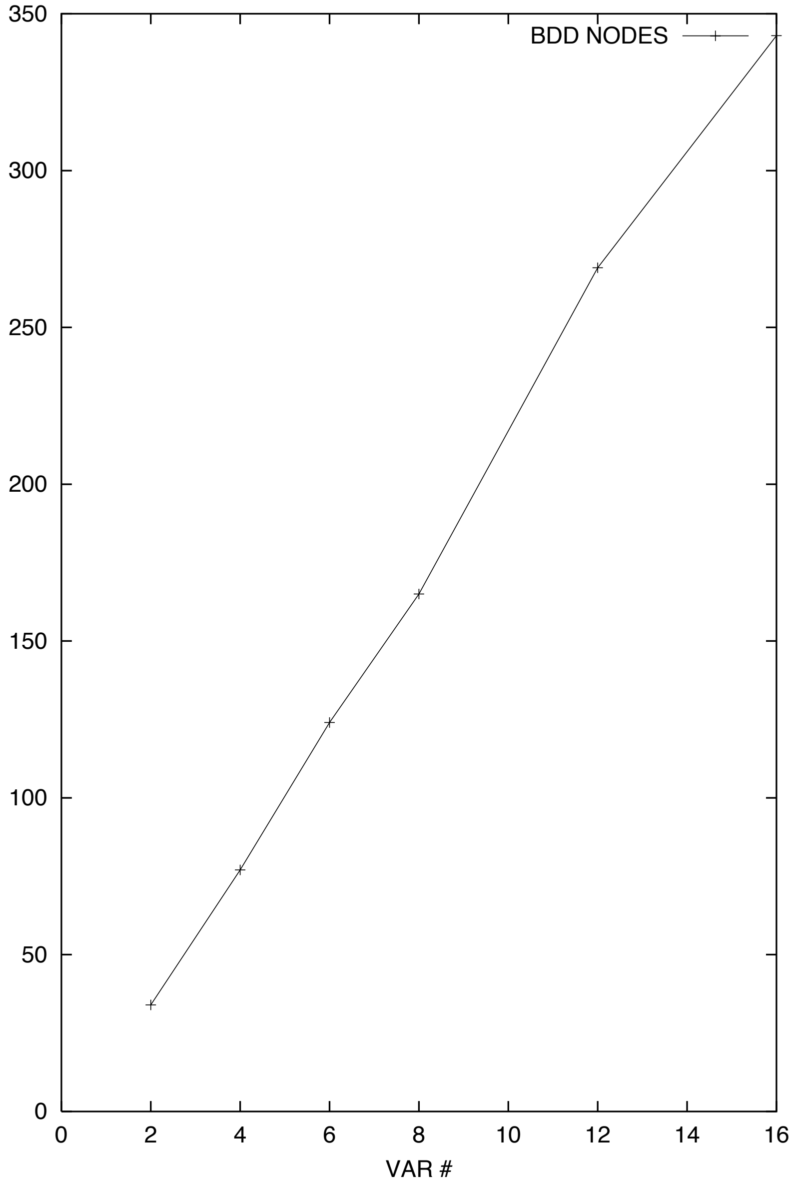
$O(2^N)$ TRANSITIONS

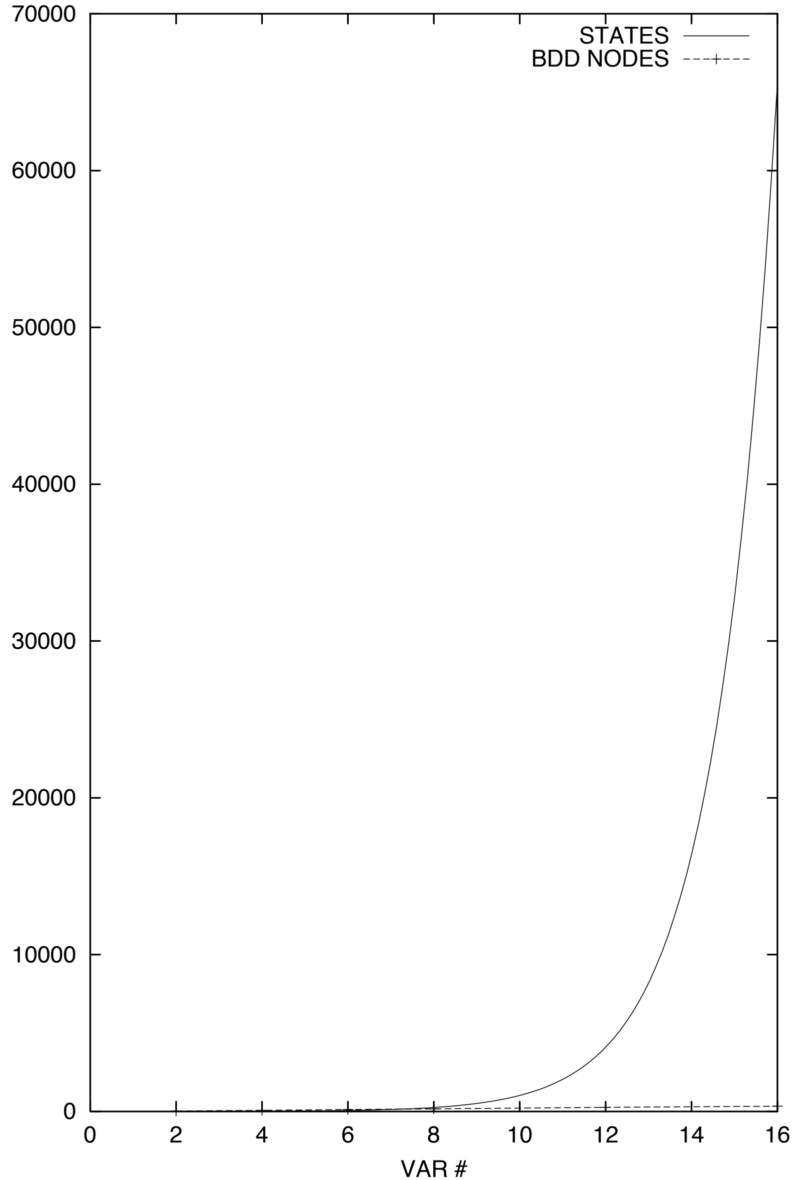
A simple example: $OBDD(\xi(R))$



$2N + 2$ NODES

A simple example: states vs. OBDD nodes [NuSMV.2]

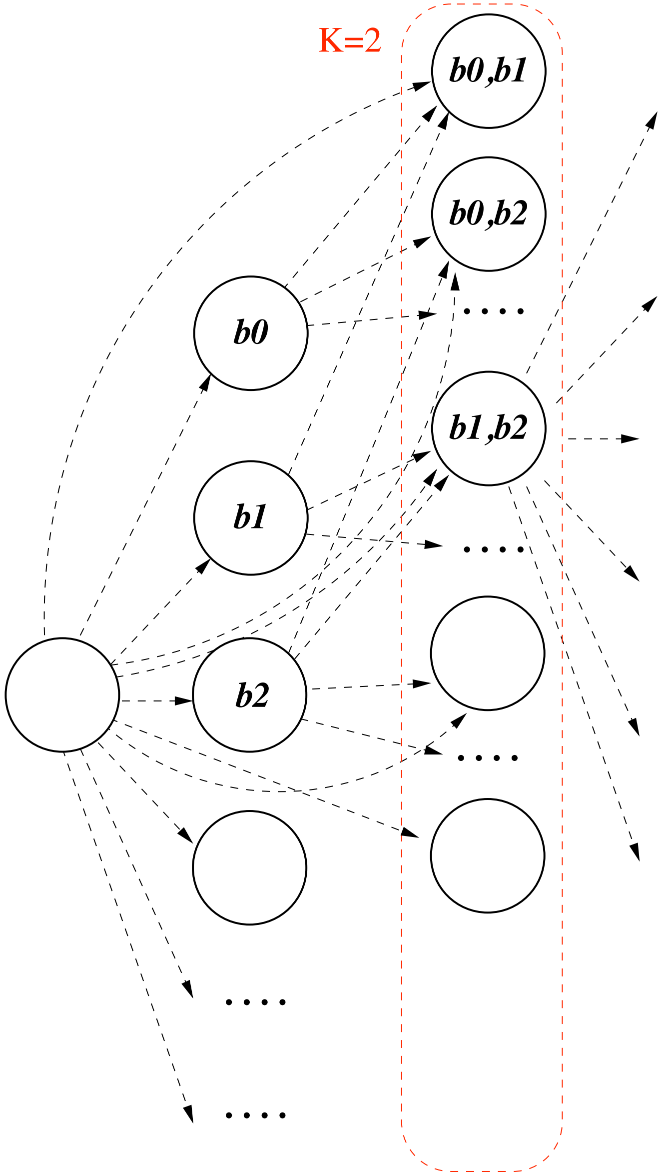




A simple example: reaching K bits true

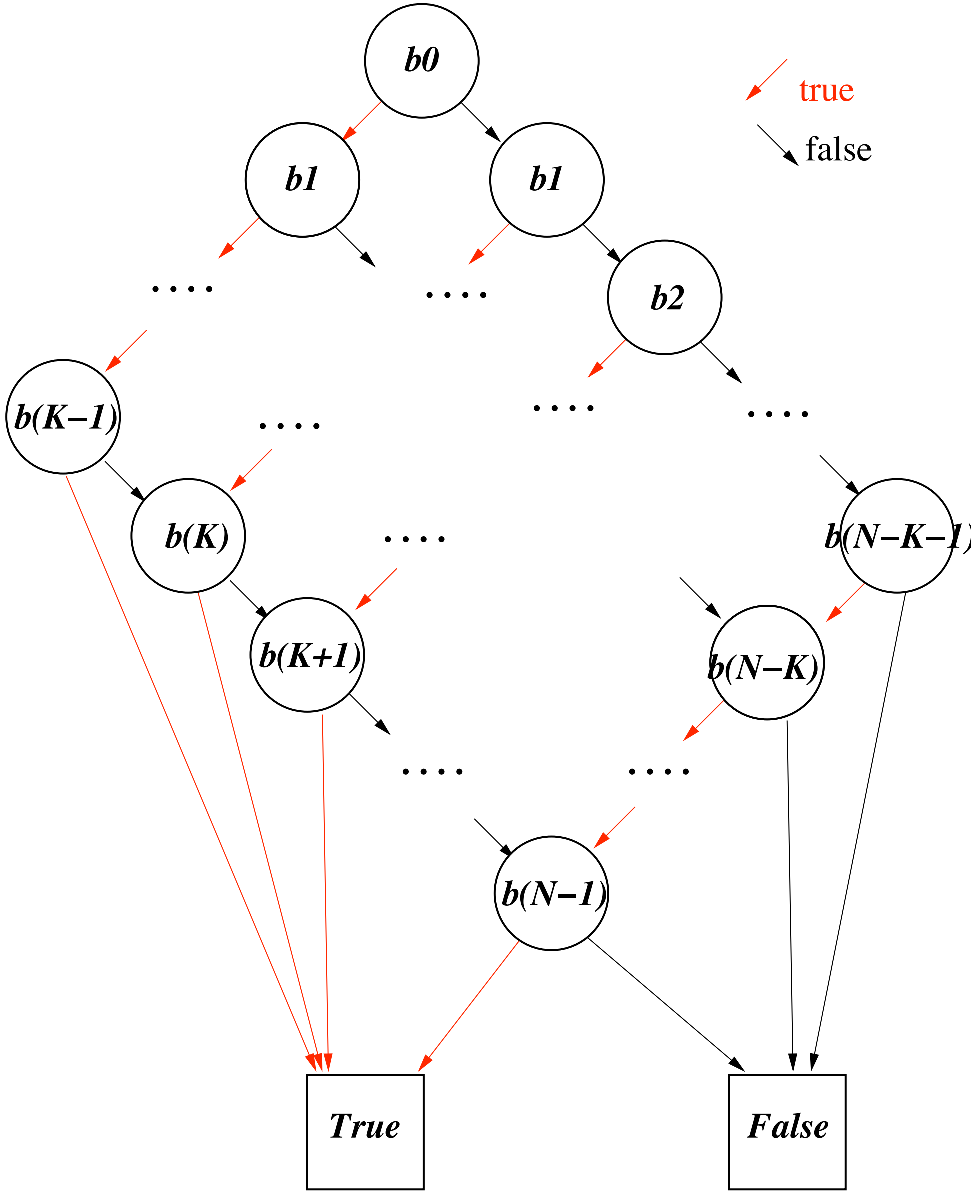
- Property $\mathbf{EF}(b0 + b1 + ... + b(N-1) = K)$ $(K \leq N)$ (it may be reached a state in which K bits are true)
- E.g.: "it is reachable a state where K exams are passed"

A simple example: FSM



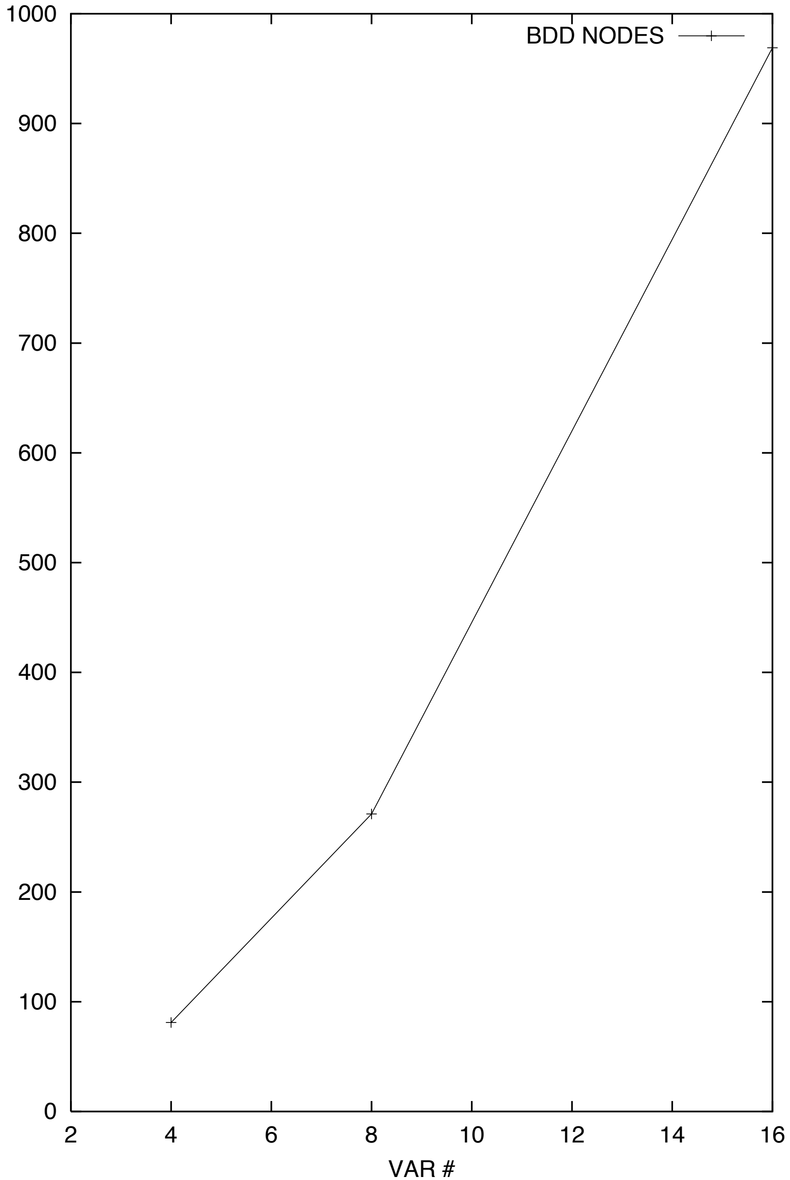
$\binom{N}{K}$ STATES

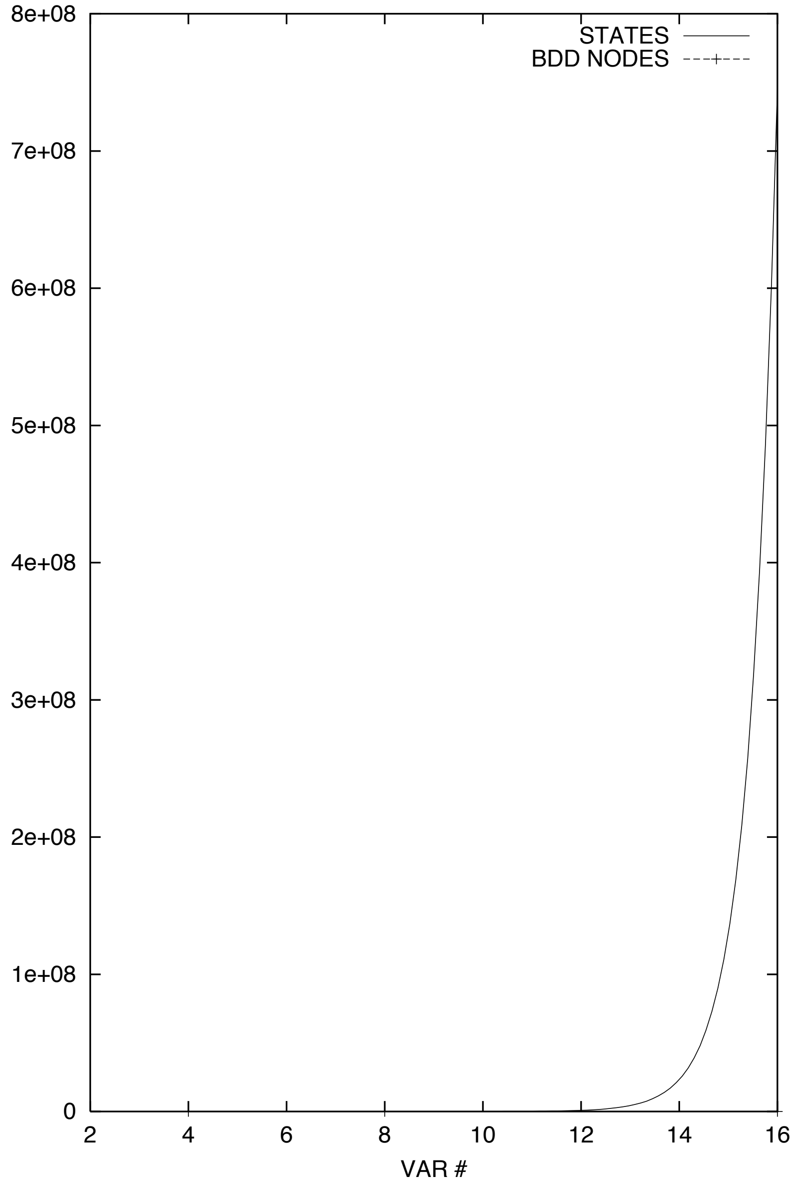
A simple example: $OBDD(\xi(\varphi))$



$(N-K) \cdot K + 2$ NODES

A simple example: states vs. OBDD nodes [NuSMV.2]





Content

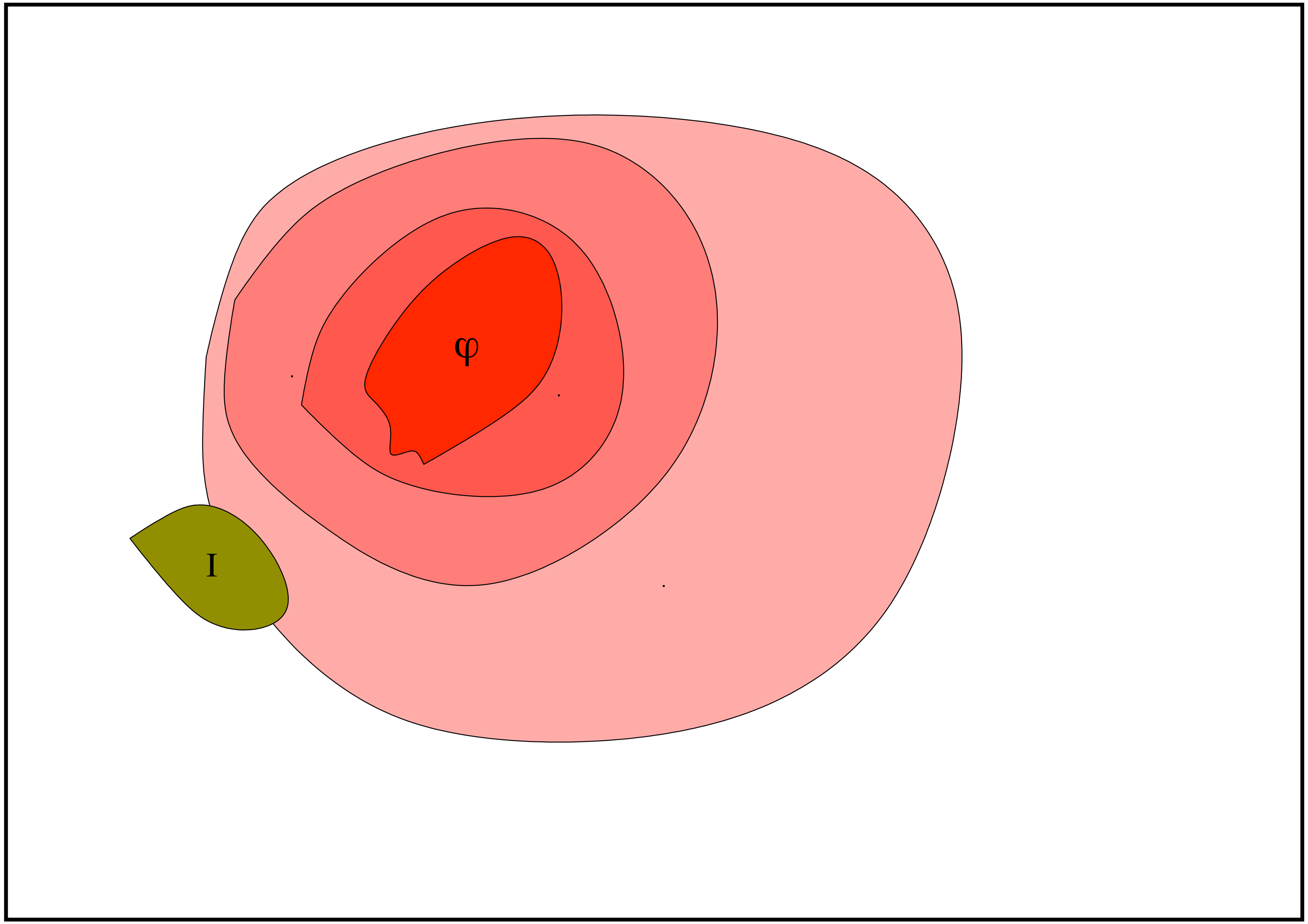
Symbolic Model Checking of Invariants

- Invariant properties have the form **AG p** (e.g., $\mathbf{AG}\neg bad$)
- Checking invariants is the negation of a reachability problem:
  - is there a reachable state that is also a bad state?
    $(\mathbf{AG}\neg bad = \neg\mathbf{EF} bad)$
- Standard M.C. algorithm reasons backward from the $\neg bad$ by iteratively applying PreImage computations:

$$Y' := Y \vee PreImage(Y)$$

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

OBDD Compute_reachable() $\{$

$\quad Y := F := I;\ j := 1;$

$\quad$ **while** $F \neq \bot$

$\qquad j := j + 1;$

$\qquad F := Image(F) \wedge \neg Y;$

$\qquad Y := Y \vee F;$

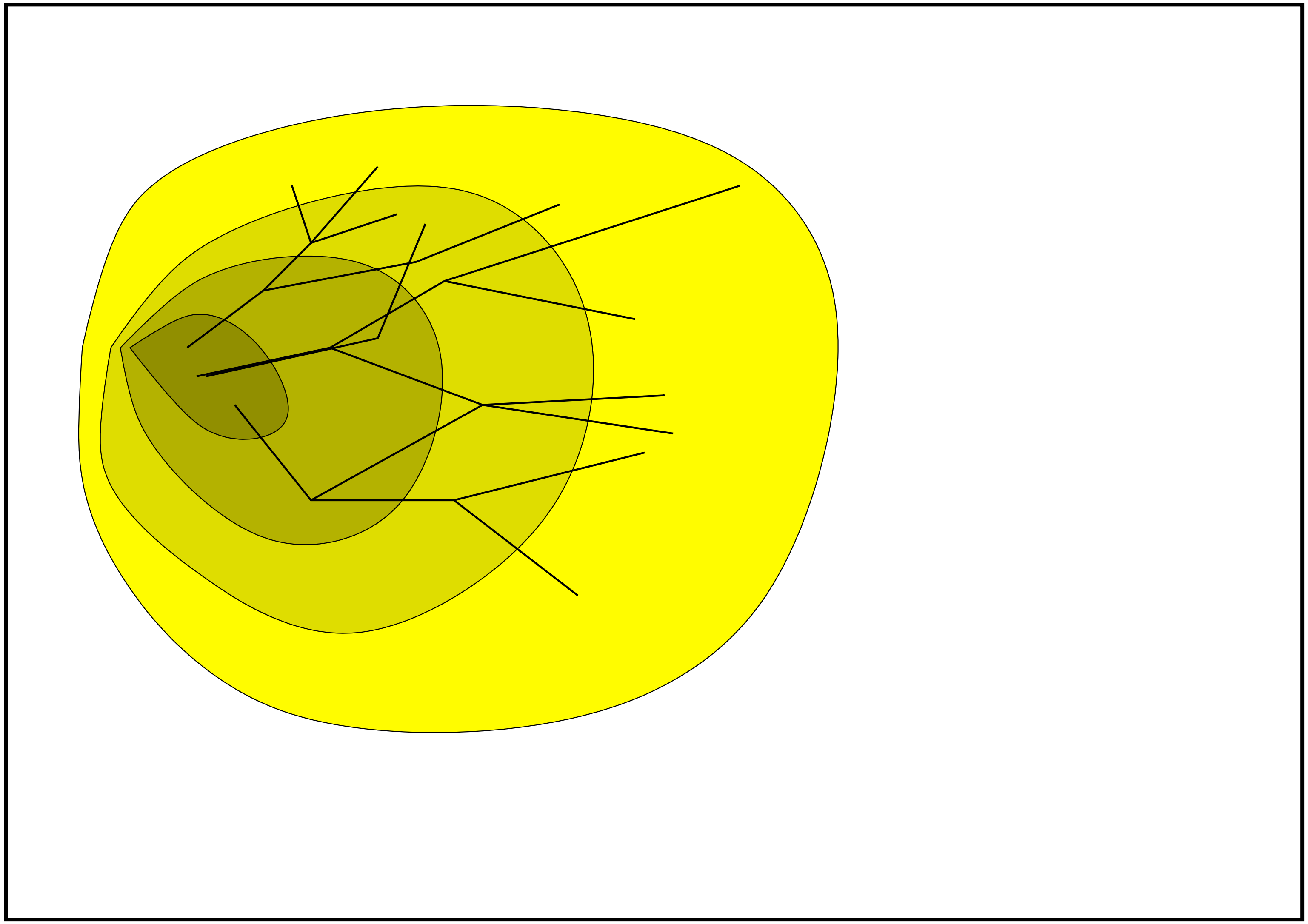
$\quad \}$

return Y;

$\}$

Y=reachable;F=frontier (new)

Computing Reachable states [cont.]

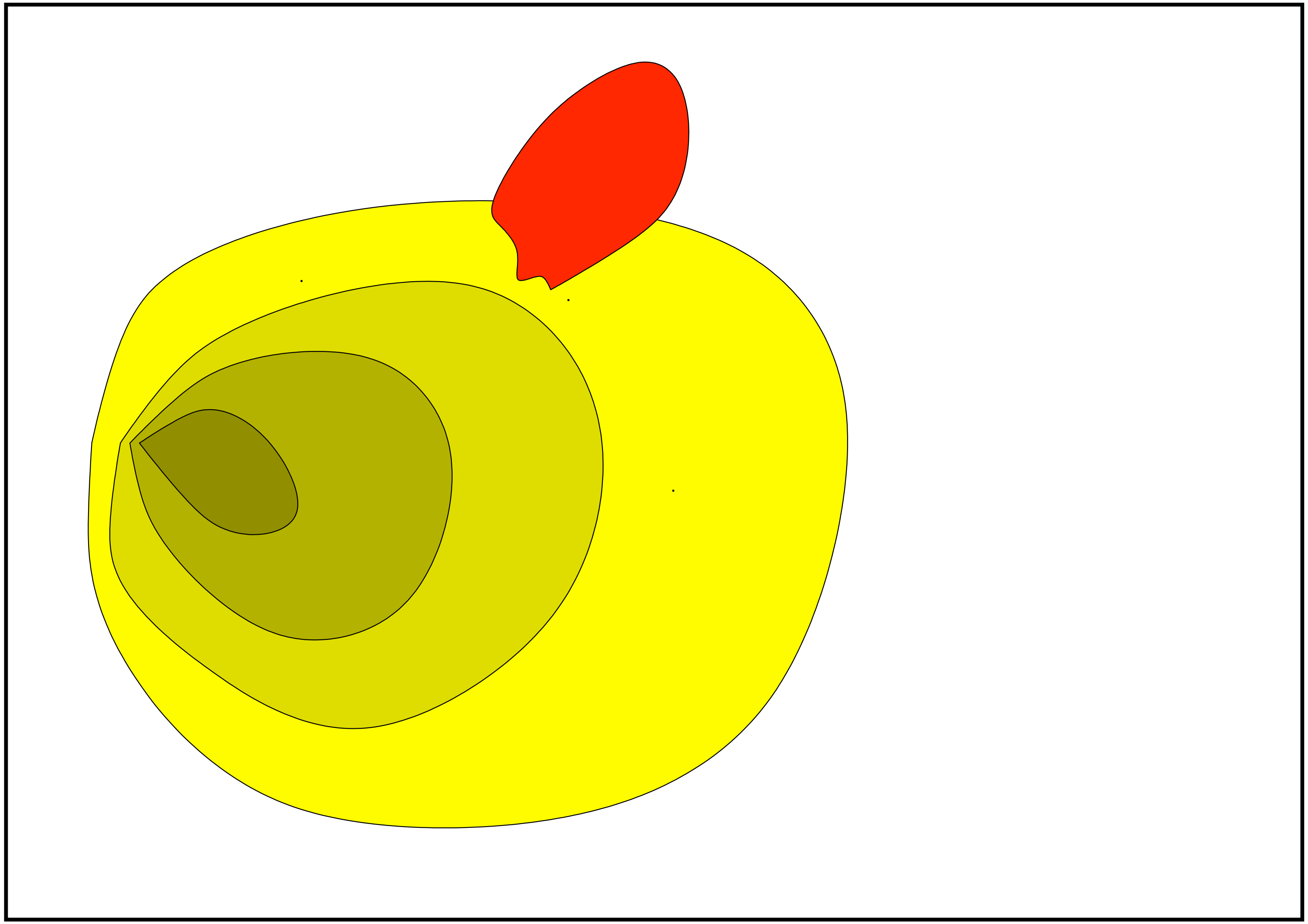


Checking of Invariant Properties

bool Forward_Check_EF(**OBDD** BAD) {
    $Y := F := I;\ j := 1;$
    while $F \neq \perp$ **and** $(F \wedge BAD) = \perp$
        $j := j + 1;$
        $F := Image(F) \wedge \neg Y;$
        $Y := Y \vee F;$
    }
    if $F = \perp$ // fixpoint reached
        return $false$
    else   // counter-example
        return $true$
}

Y=reachable;F=frontier (new)

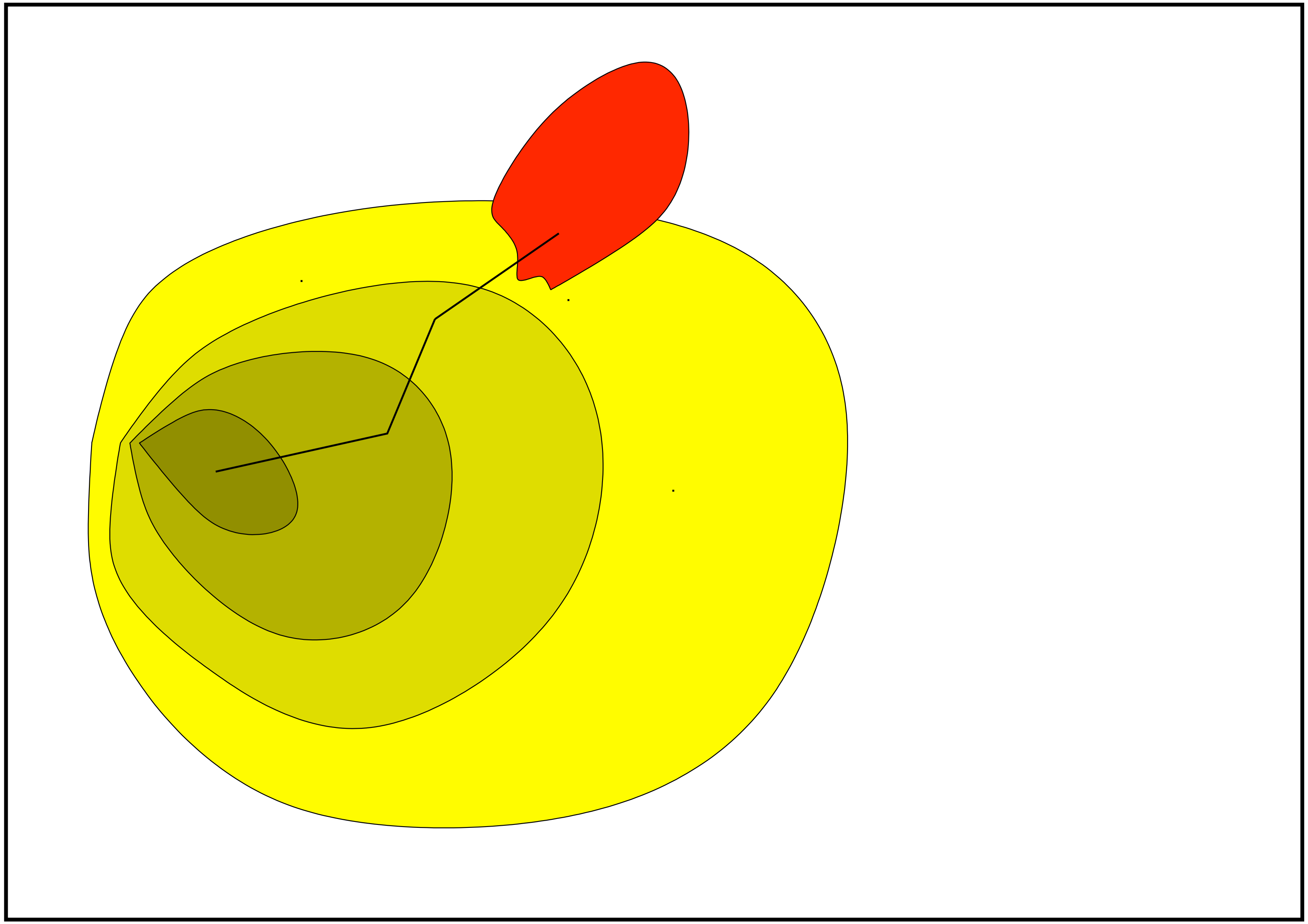
Checking of Invariant Properties [cont.]



Checking of Invariants: Counterexamples

- ▷ if layer n intersects with the bad states, then the property is violated
- ▷ a counterexample can be reconstructed proceeding backwards
- ▷ select any model of $BAD \wedge F[n]$ (we know it is satisfiable), call it $t[n]$
- ▷ compute $Preimage(t[n])$, i.e. the states that can result in $t[n]$ in one step
- ▷ compute $Preimage(t[n]) \wedge F[n-1]$, and select one model $t[n-1]$
- ▷ iterate until the initial states are reached
- ▷ $t[0], t[1], \ldots, t[n]$ is our counterexample

Checking of Invariants: Counterexamples [cont.]



Content

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

Partitioned Transition Relations

- Still, there may be significant efficiency problems:
  - the transition relation may be too large to construct
  - intermediate BDDs may be too large to handle
- IDEA: Partition conjunctively the transition relation:

$$R(V, V') \leftrightarrow \bigwedge_i R_i(V_i, V_i')$$

- Trade one "big" quantification for a sequence of "smaller" quantifications
  - $\exists V_1 \ldots V_n.(R_1(V_1, V_1') \wedge \ldots \wedge R_n(V_n, V_n') \wedge Q(V'))$
    by pushing quantifications inward can be reduced to
  - $\exists V^1.(R_1(V_1, V_1') \wedge \ldots \wedge \exists V^n(R_n(V_n, V_n') \wedge Q(V')))$
    which is typically much smaller

Other Improvements

- ▷ Preliminary step: compute reachable states of the model.
  - limit the transition relation and the sets being manipulated in model checking to the set of reachable states
- ▷ Care Set Simplification
  - reduce BDD size by transformations that preserve meaning within set of interest (care set), e.g. reachable states
- ▷ Cone of influence reduction
  - consider parts of model that are relevant for the property being analyzed
- ▷ Attempt-based BDD primitives (Bwolen Yang)
  - before calling BDD operation, set cut off in result growth
  - heuristic algorithms to clusterize and traverse space

Symbolic Model Checkers

- Most hardware design companies have their own Symbolic Model Checker(s)
  - Intel, IBM, Motorola, Siemens, ST, Cadence, ...
  - very advanced tools
  - proprietary technolgy!
- On the academic side
  - CMU SMV [McMillan]
  - VIS [Berkeley, Colorado]
  - Bwolen Yang's SMV [CMU]
  - NuSMV [CMU, IRST, UNITN, UNIGE]
  - ...