Introduction to Formal Methods
Chapter 05: Symbolic CTL Model Checking

Roberto Sebastiani

DISI, Università di Trento, Italy – roberto.sebastiani@unitn.it
URL: http://disi.unitn.it/rseba/DIDATTICA/fm2020/
Teaching assistant: Enrico Magnago – enrico.magnago@unitn.it

CDLM in Informatica, academic year 2019-2020

last update: Monday 18th May, 2020, 14:48

Outline

Outline

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 $\Longrightarrow$ 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

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 $\Longrightarrow$ 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 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

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

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

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
  - ...

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
  - ...

Outline

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

Canonical representation of Boolean formulas

- "If-then-else" binary direct acyclic graphs (DAGs) with one root and two leaves: 1, 0 (or $\top, \bot$; or T, F)
- Variable ordering $A_1, A_2, ..., A_n$ imposed a priori.
- Paths leading to 1 represent models
  Paths leading to 0 represent counter-models

Note

Some authors call them Reduced Ordered Binary Decision Diagrams (ROBDDs)

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

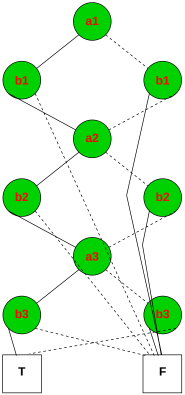
Canonical representation of Boolean formulas

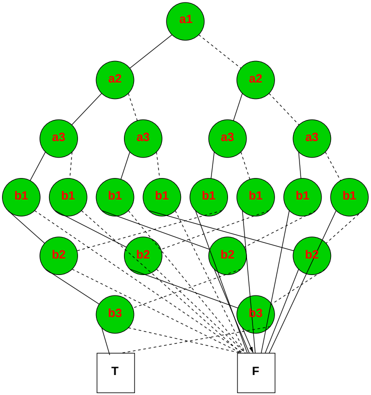
- "If-then-else" binary direct acyclic graphs (DAGs) with one root and two leaves: 1, 0 (or $\top$,$\bot$; or T, F)
- Variable ordering $A_1, A_2, ..., A_n$ imposed a priori.
- Paths leading to 1 represent models
  Paths leading to 0 represent counter-models

Note

Some authors call them Reduced Ordered Binary Decision Diagrams (ROBDDs)

OBDD - Examples

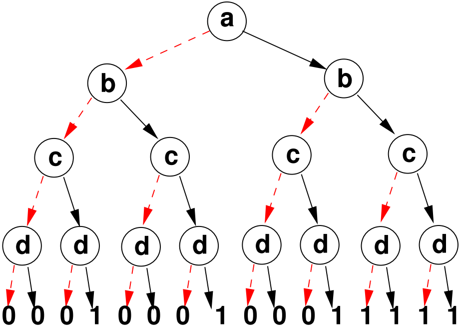




OBDDs of $(a_1 \leftrightarrow b_1) \wedge (a_2 \leftrightarrow b_2) \wedge (a_3 \leftrightarrow b_3)$ with different variable orderings

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 (via hash consing)
  - remove redundancies: nodes with same left and right children can be eliminated ("if A then B else B" $\Longrightarrow$ "B")

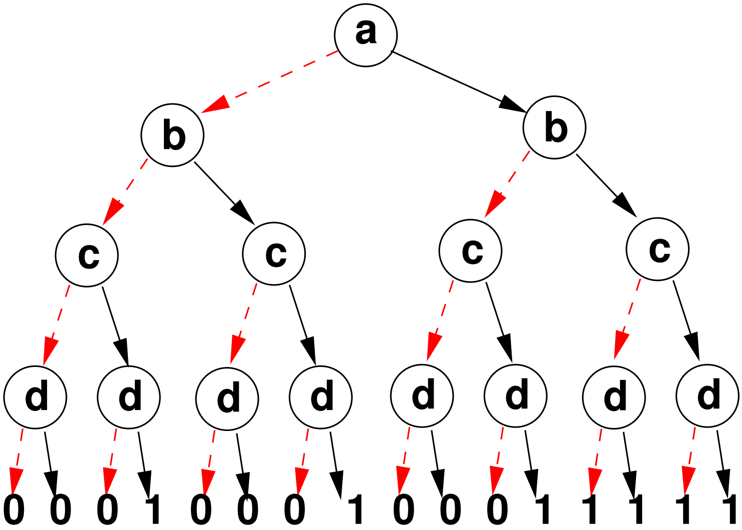
From Ordered Decision Trees to OBDD's: reductions

- Recursive applications of the following reductions:
  - share subnodes: point to the same occurrence of a subtree (via hash consing)
  - remove redundancies: nodes with same left and right children can be eliminated ("if A then B else B" $\Longrightarrow$ "B")

From Ordered Decision Trees to OBDD's: reductions

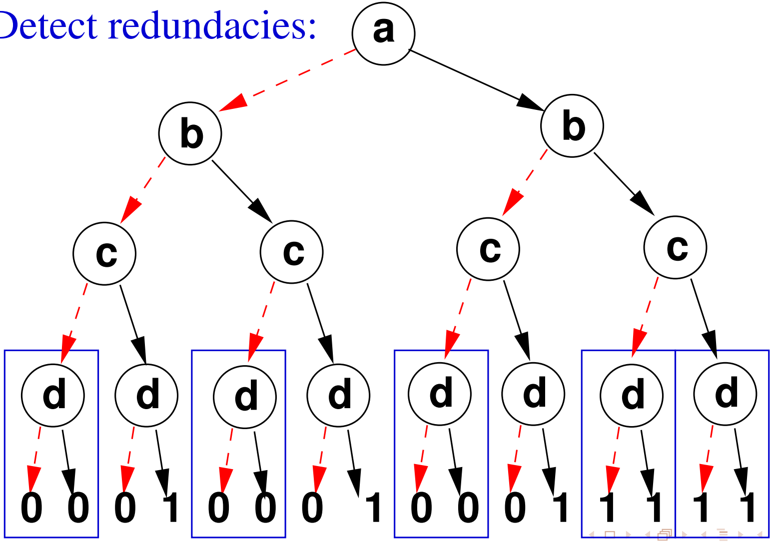
- Recursive applications of the following reductions:
  - share subnodes: point to the same occurrence of a subtree (via hash consing)
  - remove redundancies: nodes with same left and right children can be eliminated ("if A then B else B" $\Longrightarrow$ "B")

Reduction: example



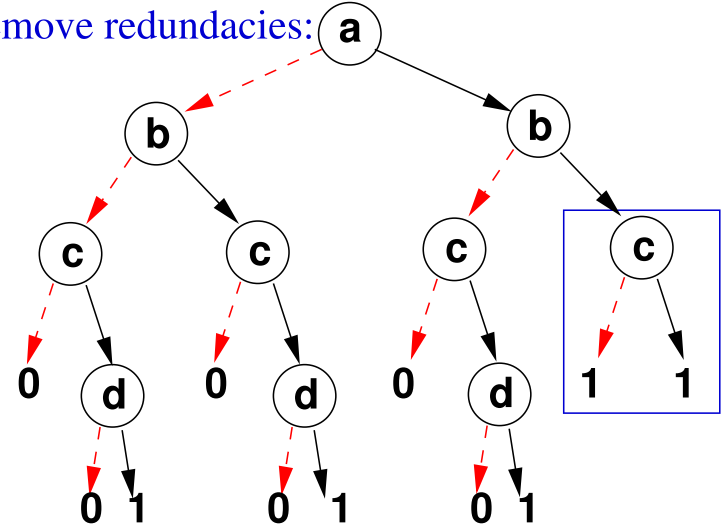
Reduction: example

Detect redundacies:



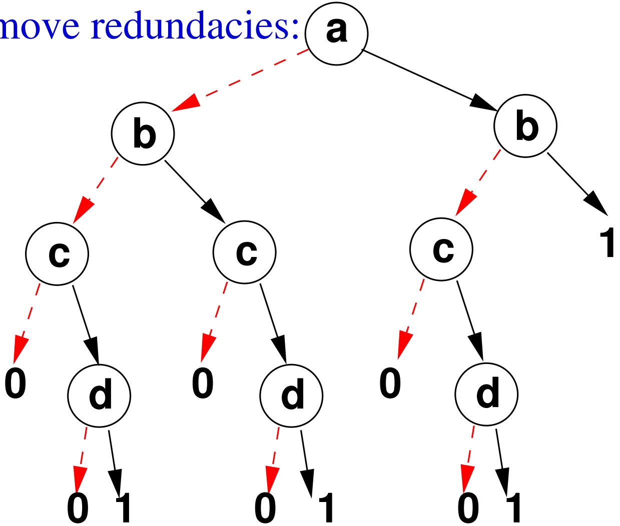
Reduction: example

Remove redundacies:



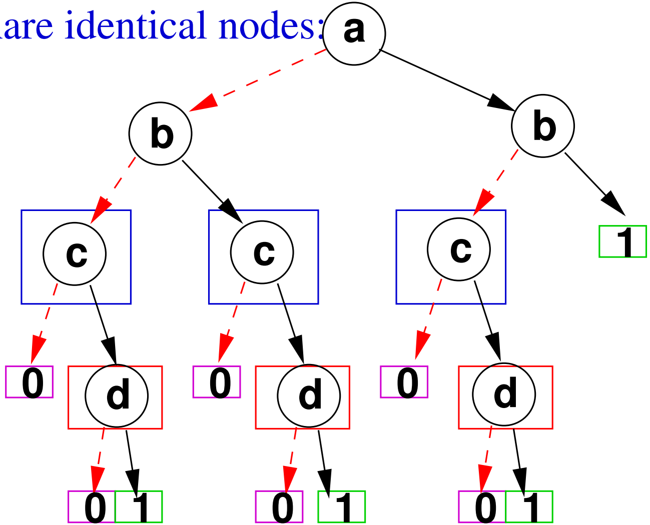
Reduction: example

Remove redundacies:



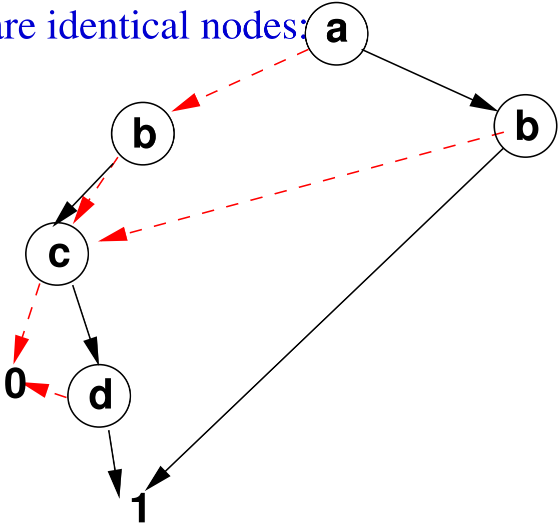
Reduction: example

Share identical nodes:



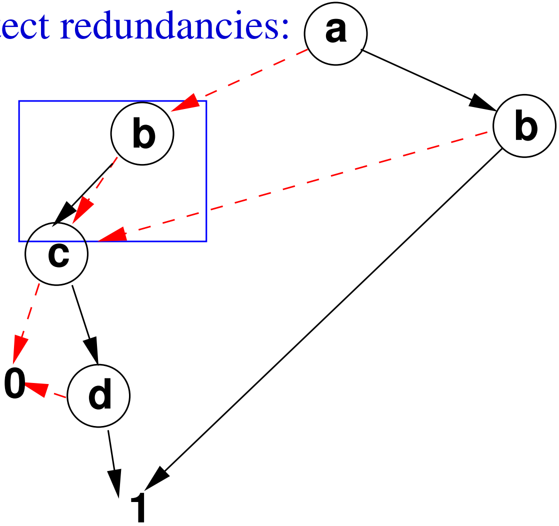
Reduction: example

Share identical nodes:



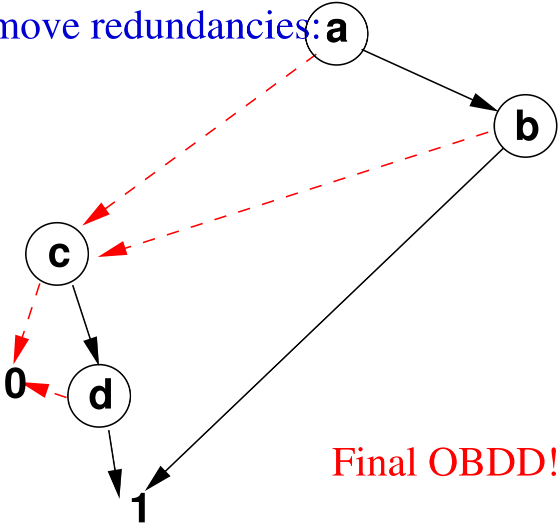
Reduction: example

Detect redundancies:



Reduction: example

Remove redundancies:



Final OBDD!

Recursive structure of an OBDD

Assume the variable ordering $A_1, A_2, ..., A_n$:

$$
\begin{array}{rcl}
OBDD(\top, \{A_1, A_2, ..., A_n\}) & = & 1 \\
OBDD(\bot, \{A_1, A_2, ..., A_n\}) & = & 0 \\
OBDD(\varphi, \{A_1, A_2, ..., A_n\}) & = & \textit{if } A_1 \\
 & & \textit{then } OBDD(\varphi[A_1|\top], \{A_2, ..., A_n\}) \\
 & & \textit{else } OBDD(\varphi[A_1|\bot], \{A_2, ..., A_n\})
\end{array}
$$

Incrementally building an OBDD

- $obdd\_build(\top, \{...\}) := 1$,
- $obdd\_build(\bot, \{...\}) := 0$,
- $obdd\_build(A_i, \{...\}) := ite(A_i, 1, 0)$,
- $obdd\_build((\neg\varphi), \{A_1, ..., A_n\}) :=$
  $apply(\neg, obdd\_build(\varphi, \{A_1, ..., A_n\}))$
- $obdd\_build((\varphi_1\ op\ \varphi_2), \{A_1, ..., A_n\}) :=$
  $reduce($
  $apply(\quad op,$
  $\quad obdd\_build(\varphi_1, \{A_1, ..., A_n\}), \quad op \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$
  $\quad obdd\_build(\varphi_2, \{A_1, ..., A_n\})$
  $))$

"$ite(A_i, \varphi_i^\top, \varphi_i^\bot)$" is "If A_i Then $\varphi_i^\top$ Else $\varphi_i^\bot$"

Incrementally building an OBDD

- $obdd\_build(\top, \{...\}) := 1$,
- $obdd\_build(\bot, \{...\}) := 0$,
- $obdd\_build(A_i, \{...\}) := ite(A_i, 1, 0)$,
- $obdd\_build((\neg\varphi), \{A_1, ..., A_n\}) :=$
  $apply(\neg, obdd\_build(\varphi, \{A_1, ..., A_n\}))$
- $obdd\_build((\varphi_1\ op\ \varphi_2), \{A_1, ..., A_n\}) :=$
  $reduce($
  $apply(\quad op,$
  $obdd\_build(\varphi_1, \{A_1, ..., A_n\}), \quad op \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$
  $obdd\_build(\varphi_2, \{A_1, ..., A_n\})$
  $))$

"$ite(A_i, \varphi_i^\top, \varphi_i^\bot)$" is "If A_i Then $\varphi_i^\top$ Else $\varphi_i^\bot$"

Incrementally building an OBDD

- $obdd\_build(\top, \{...\}) := 1$,
- $obdd\_build(\bot, \{...\}) := 0$,
- $obdd\_build(A_i, \{...\}) := ite(A_i, 1, 0)$,
- $obdd\_build((\neg\varphi), \{A_1, ..., A_n\}) :=$
  $apply(\neg, obdd\_build(\varphi, \{A_1, ..., A_n\}))$
- $obdd\_build((\varphi_1\ op\ \varphi_2), \{A_1, ..., A_n\}) :=$
  $reduce($
  $apply(\quad op,$
  $obdd\_build(\varphi_1, \{A_1, ..., A_n\}), \quad op \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$
  $obdd\_build(\varphi_2, \{A_1, ..., A_n\})$
  $))$

"$ite(A_i, \varphi_i^\top, \varphi_i^\bot)$" is "If A_i Then $\varphi_i^\top$ Else $\varphi_i^\bot$"

Incrementally building an OBDD

- $obdd\_build(\top, \{...\}) := 1$,
- $obdd\_build(\bot, \{...\}) := 0$,
- $obdd\_build(A_i, \{...\}) := ite(A_i, 1, 0)$,
- $obdd\_build((\neg\varphi), \{A_1, ..., A_n\}) :=$
  $apply(\neg, obdd\_build(\varphi, \{A_1, ..., A_n\}))$
- $obdd\_build((\varphi_1 \; op \; \varphi_2), \{A_1, ..., A_n\}) :=$
  $reduce($
  $apply(\quad op,$
  $obdd\_build(\varphi_1, \{A_1, ..., A_n\}), \quad op \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$
  $obdd\_build(\varphi_2, \{A_1, ..., A_n\})$
  $))$

"$ite(A_i, \varphi_i^\top, \varphi_i^\bot)$" is "If A_i Then $\varphi_i^\top$ Else $\varphi_i^\bot$"

Incrementally building an OBDD

- $obdd\_build(\top, \{...\}) := 1,$
- $obdd\_build(\bot, \{...\}) := 0,$
- $obdd\_build(A_i, \{...\}) := ite(A_i, 1, 0),$
- $obdd\_build((\neg\varphi), \{A_1, ..., A_n\}) :=$
  $apply(\neg, obdd\_build(\varphi, \{A_1, ..., A_n\}))$
- $obdd\_build((\varphi_1 \; op \; \varphi_2), \{A_1, ..., A_n\}) :=$
  $reduce($
  $\quad apply(\quad op,$
  $\quad\quad obdd\_build(\varphi_1, \{A_1, ..., A_n\}),$ $\quad op \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$
  $\quad\quad obdd\_build(\varphi_2, \{A_1, ..., A_n\})$
  $))$

"$ite(A_i, \varphi_i^{\top}, \varphi_i^{\bot})$" is "If A_i Then $\varphi_i^{\top}$ Else $\varphi_i^{\bot}$"

Incrementally building an OBDD (cont.)

- $apply\ (op,\ O_i, O_j) := (O_i\ op\ O_j)$ **if** $(O_i, O_j \in \{1, 0\})$
- $apply\ (\neg,\ ite(A_i, \varphi_i^\top, \varphi_i^\perp)) :=$
  $ite(A_i, apply(\neg, \varphi_i^\top), apply(\neg, \varphi_i^\perp))$
- $apply\ (op,\ ite(A_i, \varphi_i^\top, \varphi_i^\perp),\ ite(A_j, \varphi_j^\top, \varphi_j^\perp)) :=$
  if $(A_i = A_j)$ **then** $ite(A_i,\ apply\ (op, \varphi_i^\top, \varphi_j^\top),\ apply\ (op, \varphi_i^\perp, \varphi_j^\perp))$
  if $(A_i < A_j)$ **then** $ite(A_i,\ apply\ (op, \varphi_i^\top, ite(A_j, \varphi_j^\top, \varphi_j^\perp)),\ apply\ (op, \varphi_i^\perp, ite(A_j, \varphi_j^\top, \varphi_j^\perp)))$
  if $(A_i > A_j)$ **then** $ite(A_j,\ apply\ (op, ite(A_i, \varphi_i^\top, \varphi_i^\perp), \varphi_j^\top),\ apply\ (op, ite(A_i, \varphi_i^\top, \varphi_i^\perp), \varphi_j^\perp))$

$op \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$

Incrementally building an OBDD (cont.)

- $apply\ (op,\ O_i, O_j) := (O_i\ op\ O_j)$ **if** $(O_i, O_j \in \{1, 0\})$
- $apply\ (\neg,\ ite(A_i, \varphi_i^\top, \varphi_i^\bot)) :=$
  $ite(A_i, apply(\neg, \varphi_i^\top), apply(\neg, \varphi_i^\bot))$
- $apply\ (op,\ ite(A_i, \varphi_i^\top, \varphi_i^\bot),\ ite(A_j, \varphi_j^\top, \varphi_j^\bot)) :=$
  if $(A_i = A_j)$ **then** $ite(A_i,\ apply\ (op, \varphi_i^\top, \varphi_j^\top),\ apply\ (op, \varphi_i^\bot, \varphi_j^\bot))$
  if $(A_i < A_j)$ **then** $ite(A_i,\ apply\ (op, \varphi_i^\top, ite(A_j, \varphi_j^\top, \varphi_j^\bot)),\ apply\ (op, \varphi_i^\bot, ite(A_j, \varphi_j^\top, \varphi_j^\bot)))$
  if $(A_i > A_j)$ **then** $ite(A_j,\ apply\ (op, ite(A_i, \varphi_i^\top, \varphi_i^\bot), \varphi_j^\top),\ apply\ (op, ite(A_i, \varphi_i^\top, \varphi_i^\bot), \varphi_j^\bot))$

$op \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$

Incrementally building an OBDD (cont.)

- $apply\ (op,\ O_i, O_j) := (O_i\ op\ O_j)$ **if** $(O_i, O_j \in \{1, 0\})$
- $apply\ (\neg,\ ite(A_i, \varphi_i^\top, \varphi_i^\bot)) :=$
  $ite(A_i, apply(\neg, \varphi_i^\top), apply(\neg, \varphi_i^\bot))$
- $apply\ (op,\ ite(A_i, \varphi_i^\top, \varphi_i^\bot),\ ite(A_j, \varphi_j^\top, \varphi_j^\bot)) :=$

  if $(A_i = A_j)$ **then** $ite(A_i,\ apply\ (op, \varphi_i^\top, \varphi_j^\top),\ apply\ (op, \varphi_i^\bot, \varphi_j^\bot))$

  if $(A_i < A_j)$ **then** $ite(A_i,\ apply\ (op, \varphi_i^\top, ite(A_j, \varphi_j^\top, \varphi_j^\bot)),\ apply\ (op, \varphi_i^\bot, ite(A_j, \varphi_j^\top, \varphi_j^\bot)))$

  if $(A_i > A_j)$ **then** $ite(A_j,\ apply\ (op, ite(A_i, \varphi_i^\top, \varphi_i^\bot), \varphi_j^\top),\ apply\ (op, ite(A_i, \varphi_i^\top, \varphi_i^\bot), \varphi_j^\bot))$

$op \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$

Incrementally building an OBDD (cont.)

- Ex: build the obdd for $A_1 \vee A_2$ from those of A_1, A_2 (order: A_1, A_2):

$$
\begin{aligned}
& apply(\vee, \overbrace{ite(A_1, \top, \bot)}^{A_1}, \overbrace{ite(A_2, \top, \bot)}^{A_2}) \\
= \;& ite(A_1,\ apply(\vee, \top, ite(A_1, \top, \bot)),\ apply(\vee, \bot, ite(A_2, \top, \bot))\ ) \\
= \;& ite(A_1,\ \top,\ ite(A_2, \top, \bot)\ )
\end{aligned}
$$

- Ex: build the obdd for $(A_1 \vee A_2) \wedge (A_1 \vee \neg A_2)$ from those of $(A_1 \vee A_2)$, $(A_1 \vee \neg A_2)$ (order: A_1, A_2):

$$
\begin{aligned}
& apply(\wedge, \overbrace{ite(A_1, \top, ite(A_2, \top, \bot))}^{(A_1 \vee A_2)}, \overbrace{ite(A_1, \top, ite(A_2, \bot, \top))}^{(A_1 \vee \neg \mathbf{A}_2)}, \\
= \;& ite(A_1,\ apply(\wedge, \top, \top),\ apply(\wedge,\ ite(A_2, \top, \bot),\ ite(A_2, \bot, \top)) \\
= \;& ite(A_1,\ \top,\ ite(A_2,\ apply(\wedge, \top, \bot),\ apply(\wedge, \bot, \top))) \\
= \;& ite(A_1,\ \top,\ ite(A_2,\ \bot,\ \bot)) \\
= \;& ite(A_1,\ \top,\ \bot)
\end{aligned}
$$

Incrementally building an OBDD (cont.)

- Ex: build the obdd for $A_1 \vee A_2$ from those of A_1, A_2 (order: A_1, A_2):

$$
\begin{array}{rl}
 & apply(\vee, \overbrace{ite(A_1, \top, \bot)}^{A_1}, \overbrace{ite(A_2, \top, \bot)}^{A_2}) \\
= & ite(A_1,\ apply(\vee, \top, ite(A_1, \top, \bot)),\ apply(\vee, \bot, ite(A_2, \top, \bot))\ ) \\
= & ite(A_1,\ \top,\ ite(A_2, \top, \bot)\ )
\end{array}
$$

- Ex: build the obdd for $(A_1 \vee A_2) \wedge (A_1 \vee \neg A_2)$ from those of $(A_1 \vee A_2)$, $(A_1 \vee \neg A_2)$ (order: A_1, A_2):

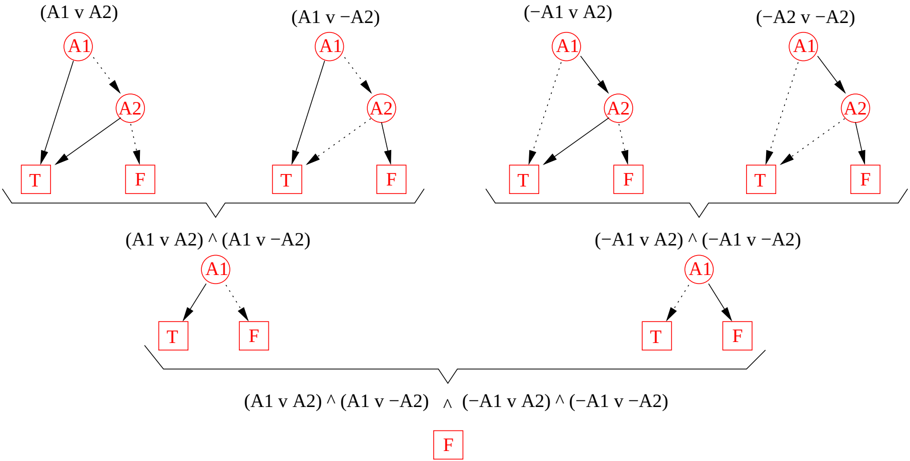
$$
\begin{array}{rl}
 & apply(\wedge, \overbrace{ite(A_1, \top, ite(A_2, \top, \bot))}^{(A_1 \vee A_2)}, \overbrace{ite(A_1, \top, ite(A_2, \bot, \top))}^{(A_1 \vee \neg \mathbf{A}_2)}, \\
= & ite(A_1,\ apply(\wedge, \top, \top),\ apply(\wedge,\ ite(A_2, \top, \bot),\ ite(A_2, \bot, \top)) \\
= & ite(A_1,\ \top,\ ite(A_2,\ apply(\wedge, \top, \bot),\ apply(\wedge, \bot, \top))) \\
= & ite(A_1,\ \top,\ ite(A_2,\ \bot,\ \bot)) \\
= & ite(A_1,\ \top,\ \bot)
\end{array}
$$

OBBD incremental building – example

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

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

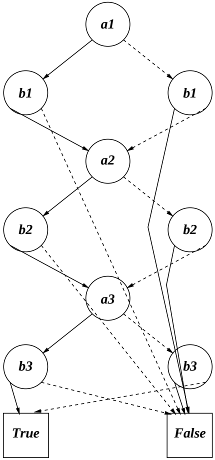
$$(a_1 \leftrightarrow b_1) \wedge (a_2 \leftrightarrow b_2) \wedge (a_3 \leftrightarrow b_3)$$

Linear size

Exponential size

Critical choice of variable Orderings in OBDD's

$$(a_1 \leftrightarrow b_1) \wedge (a_2 \leftrightarrow b_2) \wedge (a_3 \leftrightarrow b_3)$$

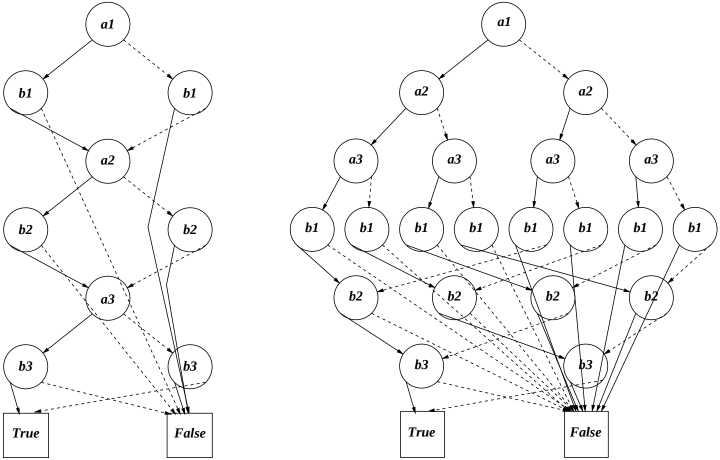


Linear size

Exponential size

Critical choice of variable Orderings in OBDD's

$$(a_1 \leftrightarrow b_1) \wedge (a_2 \leftrightarrow b_2) \wedge (a_3 \leftrightarrow b_3)$$



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

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

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

Note

The size of intermediate OBDD's may be bigger than that of the final one (e.g., inconsistent 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

Note

The size of intermediate OBDD's may be bigger than that of the final one (e.g., inconsistent 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

Note

The size of intermediate OBDD's may be bigger than that of the final one (e.g., inconsistent 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

Note

The size of intermediate OBDD's may be bigger than that of the final one (e.g., inconsistent formula)

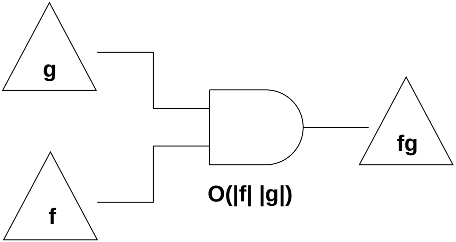
Useful Operations over OBDDs

- 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).

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

Shannon's expansion:

- If v is a Boolean variable and f is a Boolean formula, then
  $$\exists v.f \quad := \quad f|_{v=0} \vee f|_{v=1}$$
  $$\forall v.f \quad := \quad f|_{v=0} \wedge f|_{v=1}$$
- 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: $\exists b, c \,.\, ((a \wedge b) \vee (c \wedge d)) \ = \ a \vee d$

Note

Naive expansion of quantifiers to propositional logic may cause a blow-up in size of the formulae

Boolean quantification

Shannon's expansion:

- If v is a Boolean variable and f is a Boolean formula, 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 $\forall v.f$!!
- 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: $\exists b, c\ .\ ((a \wedge b) \vee (c \wedge d)) \;=\; a \vee d$

Note

Naive expansion of quantifiers to propositional logic may cause a blow-up in size of the formulae

Boolean quantification

Shannon's expansion:

- If v is a Boolean variable and f is a Boolean formula, then

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

- 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: $\exists b, c \,.\, ((a \wedge b) \vee (c \wedge d)) \;=\; a \vee d$

Note

Naive expansion of quantifiers to propositional logic may cause a blow-up in size of the formulae

Boolean quantification

Shannon's expansion:

- If v is a Boolean variable and f is a Boolean formula, then
  $$\begin{array}{rcl} \exists v.f & := & f|_{v=0} \vee f|_{v=1} \\ \forall v.f & := & f|_{v=0} \wedge f|_{v=1} \end{array}$$
- 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: $\exists b, c \,.\, ((a \wedge b) \vee (c \wedge d)) \ = \ a \vee d$

Note

Naive expansion of quantifiers to propositional logic may cause a blow-up in size of the formulae

Boolean quantification

Shannon's expansion:

- If v is a Boolean variable and f is a Boolean formula, then
  $$\begin{array}{lcl} \exists v.f & := & f|_{v=0} \vee f|_{v=1} \\ \forall v.f & := & f|_{v=0} \wedge f|_{v=1} \end{array}$$
- 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: $\exists b, c \,.\, ((a \wedge b) \vee (c \wedge d)) \ = \ a \vee d$

Note

Naive expansion of quantifiers to propositional logic may cause a blow-up in size of the formulae

Boolean quantification

Shannon's expansion:

- If v is a Boolean variable and f is a Boolean formula, then
$$\begin{array}{lcl} \exists v.f & := & f|_{v=0} \vee f|_{v=1} \\ \forall v.f & := & f|_{v=0} \wedge f|_{v=1} \end{array}$$
- v does no more occur in $\exists v.f$ and $\forall v.f$!!
- Multi-variable quantification: $\exists(w_1, \ldots, w_n).f \;:=\; \exists w_1 \ldots \exists w_n.f$

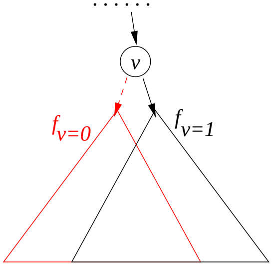
- Intuition:
  - $\mu \models \exists v.f$ iff exists *tvalue* $\in \{\top, \bot\}$ s.t. $\mu \cup \{v := tvalue\} \models f$
  - $\mu \models \forall v.f$ iff forall *tvalue* $\in \{\top, \bot\}$, $\mu \cup \{v := tvalue\} \models f$
- Example: $\exists b, c\,.\,((a \wedge b) \vee (c \wedge d)) \;=\; a \vee d$

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



$\Longrightarrow$ 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).

Outline

Symbolic Representation of Kripke Structures

- Symbolic representation:
  - sets of states as their characteristic function (Boolean formula)
  - 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": $\neg x_1$

Symbolic Representation of Kripke Structures

- Symbolic representation:
  - sets of states as their characteristic function (Boolean formula)
  - 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": $\neg x_1$

Symbolic Representation of Kripke Structures

- Symbolic representation:
  - sets of states as their characteristic function (Boolean formula)
  - 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 a 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 represented by (any formula which is logically equivalent to) the formula $\xi(Q)$:

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

  (Intuition: $\xi(Q)$ holds iff the system is in one of the states $s \in Q$)
- Bijection between models of $\xi(Q)$ and states in Q

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 a 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 represented by (any formula which is logically equivalent to) the formula $\xi(Q)$:

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

(Intuition: $\xi(Q)$ holds iff the system is in one of the states $s \in Q$)
- Bijection between models of $\xi(Q)$ and states in Q

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 a 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 represented by (any formula which is logically equivalent to) the formula $\xi(Q)$:

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

(Intuition: $\xi(Q)$ holds iff the system is in one of the states $s \in Q$)

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

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 a 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 represented by (any formula which is logically equivalent to) the formula $\xi(Q)$:

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

  (Intuition: $\xi(Q)$ holds iff the system is in one of the states $s \in Q$)
- Bijection between models of $\xi(Q)$ and states in Q

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 a 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 represented by (any formula which is logically equivalent to) the formula $\xi(Q)$:

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

(Intuition: $\xi(Q)$ holds iff the system is in one of the states $s \in Q$)
- Bijection between models of $\xi(Q)$ and states in Q

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 a 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 represented by (any formula which is logically equivalent to) the formula $\xi(Q)$:

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

(Intuition: $\xi(Q)$ holds iff the system is in one of the states $s \in Q$)

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

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 a 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 represented by (any formula which is logically equivalent to) the formula $\xi(Q)$:

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

  (Intuition: $\xi(Q)$ holds iff the system is in one of the states $s \in Q$)
- 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
  $\Longrightarrow$ 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": $\neg x_1$

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

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
  $\Longrightarrow$ 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": $\neg x_1$

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

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
  $\Longrightarrow$ 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”: $\neg x_1$

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

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: $\xi(P \cup Q) := \xi(P) \vee \xi(Q)$
- 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 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:
  $\xi(P \cup Q) := \xi(P) \vee \xi(Q)$
- 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 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: $\xi(P \cup Q) := \xi(P) \vee \xi(Q)$
- 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 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: $\xi(P \cup Q) := \xi(P) \vee \xi(Q)$
- 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 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:
  $\xi(P \cup Q) := \xi(P) \vee \xi(Q)$
- 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 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:
  $\xi(P \cup Q) := \xi(P) \vee \xi(Q)$
- 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 $\xi(s) \wedge \xi(s')$ (Intuition: $\xi(s, s')$ holds iff the system is in the state s and moves to state s' in next step)
- 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 $\bigvee_{(s,s')\in R} \xi(s) \wedge \xi(s')$!

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 $\xi(s) \land \xi(s')$ (Intuition: $\xi(s, s')$ holds iff the system is in the state s and moves to state s' in next step)
- 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
$\Longrightarrow$ Typically R can be encoded by a much smaller formula than $\bigvee_{(s,s')\in R} \xi(s) \land \xi(s')$!

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 $\xi(s) \wedge \xi(s')$ (Intuition: $\xi(s, s')$ holds iff the system is in the state s and moves to state s' in next step)
- 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 $\bigvee_{(s,s')\in R} \xi(s) \wedge \xi(s')$!

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 $\xi(s) \land \xi(s')$ (Intuition: $\xi(s, s')$ holds iff the system is in the state s and moves to state s' in next step)
- 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
$\Longrightarrow$ Typically R can be encoded by a much smaller formula than $\bigvee_{(s,s')\in R} \xi(s) \land \xi(s')$!

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 $\xi(s) \wedge \xi(s')$ (Intuition: $\xi(s, s')$ holds iff the system is in the state s and moves to state s' in next step)
- 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 $\bigvee_{(s,s')\in R} \xi(s) \wedge \xi(s')$!

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 $\xi(s) \wedge \xi(s')$ (Intuition: $\xi(s, s')$ holds iff the system is in the state s and moves to state s' in next step)
- The transition relation R can be (naively) represented by

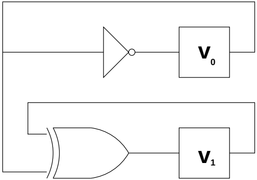
$$\bigvee_{(s,s')\in R} \xi(s, s') = \bigvee_{(s,s')\in R} (\xi(s) \wedge \xi(s'))$$

Note

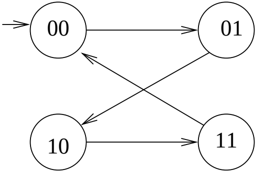
Each formula equivalent to $\xi(R)$ is a representation of R
$\Longrightarrow$ Typically R can be encoded by a much smaller formula than $\bigvee_{(s,s')\in R} \xi(s) \wedge \xi(s')$!

Example: a simple counter

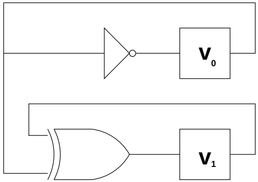
```
MODULE main
 VAR
   v0     : boolean;
   v1     : boolean;
   out    : 0..3;

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

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

   out := toint(v0) + 2*toint(v1);
```



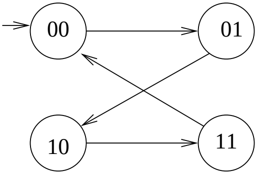
v_1	v_0	v_1'	v_0'
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0



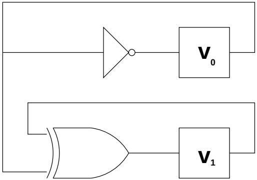
Example: a simple counter [cont.]



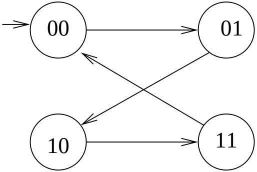
v_1	v_0	v_1'	v_0'
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0



Example: a simple counter [cont.]

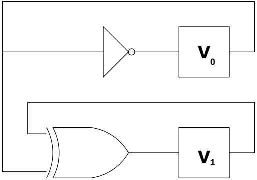


v_1	v_0	v'_1	v'_0
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

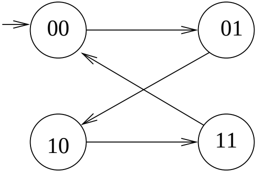


$$\xi(R) \quad = \quad (v'_0 \leftrightarrow \neg v_0) \wedge (v'_1 \leftrightarrow v_0 \bigoplus v_1)$$

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

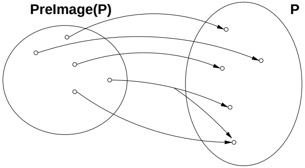


$$\xi(R) = (v_0' \leftrightarrow \neg v_0) \wedge (v_1' \leftrightarrow v_0 \bigoplus v_1)$$

$$\begin{aligned} \bigvee_{(s,s')\in R} \xi(s) \wedge \xi(s') = \; & (\neg v_1 \wedge \neg v_0 \wedge \neg v_1' \wedge v_0') \vee \\ & (\neg v_1 \wedge v_0 \wedge v_1' \wedge \neg v_0') \vee \\ & (v_1 \wedge \neg v_0 \wedge v_1' \wedge v_0') \vee \\ & (v_1 \wedge v_0 \wedge \neg v_1' \wedge \neg v_0') \end{aligned}$$

Pre-Image

- (Backward) pre-image of a set:

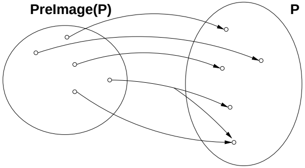


Evaluate one-shot all transitions ending in the states of the set

- Set theoretic view:
  $PreImage(P, R) := \{s \mid \text{for some } s' \in P, (s, s') \in R\}$
- 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: $\mu \Longleftrightarrow s$, $\mu' \Longleftrightarrow s'$, $\mu \cup \mu' \Longleftrightarrow \langle s, s' \rangle$

Pre-Image

- (Backward) pre-image of a set:

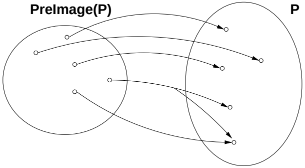


Evaluate one-shot all transitions ending in the states of the set

- Set theoretic view:
  $PreImage(P, R) := \{s \mid \text{for some } s' \in P, (s, s') \in R\}$
- 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: $\mu \Longleftrightarrow s$, $\mu' \Longleftrightarrow s'$, $\mu \cup \mu' \Longleftrightarrow \langle s, s' \rangle$

Pre-Image

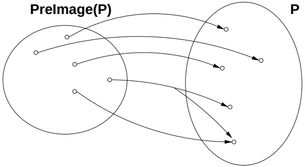
- (Backward) pre-image of a set:



  Evaluate one-shot all transitions ending in the states of the set
- Set theoretic view:
  $PreImage(P, R) := \{s \mid \text{for some } s' \in P, (s, s') \in R\}$
- 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: $\mu \Longleftrightarrow s$, $\mu' \Longleftrightarrow s'$, $\mu \cup \mu' \Longleftrightarrow \langle s, s' \rangle$

Pre-Image

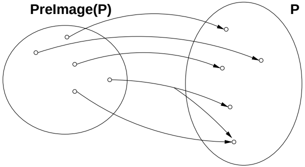
- (Backward) pre-image of a set:



  Evaluate one-shot all transitions ending in the states of the set
- Set theoretic view:
  $PreImage(P, R) := \{s \mid \text{for some } s' \in P, (s, s') \in R\}$
- 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: $\mu \Longleftrightarrow s$, $\mu' \Longleftrightarrow s'$, $\mu \cup \mu' \Longleftrightarrow \langle s, s' \rangle$

Pre-Image

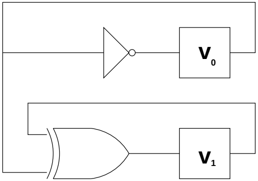
- (Backward) pre-image of a set:



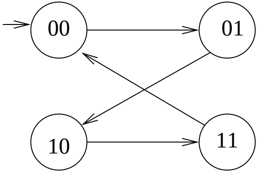
  Evaluate one-shot all transitions ending in the states of the set

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



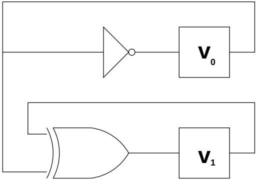
v_1	v_0	v_1'	v_0'
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0



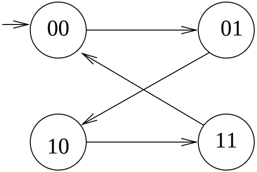
$\xi(R) = (v_0' \leftrightarrow \neg v_0) \wedge (v_1' \leftrightarrow v_0 \bigoplus v_1)$
$\xi(P) := (v_0 \leftrightarrow v_1)$ (i.e., $P = \{00, 11\}$)

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

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

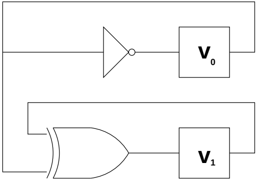


$\xi(R) = (v_0' \leftrightarrow \neg v_0) \wedge (v_1' \leftrightarrow v_0 \bigoplus v_1)$

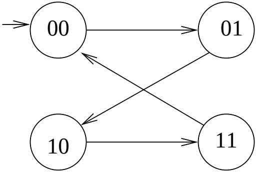
$\xi(P) := (v_0 \leftrightarrow v_1)$ (i.e., $P = \{00, 11\}$)

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

Example: simple counter



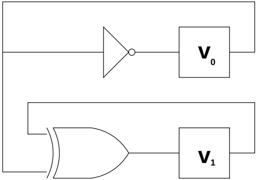
v_1	v_0	v_1'	v_0'
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0



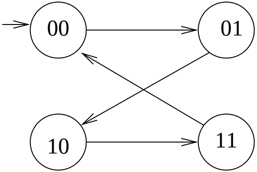
$\xi(R) = (v_0' \leftrightarrow \neg v_0) \wedge (v_1' \leftrightarrow v_0 \bigoplus v_1)$
$\xi(P) := (v_0 \leftrightarrow v_1)$ (i.e., $P = \{00, 11\}$)

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

Example: simple counter



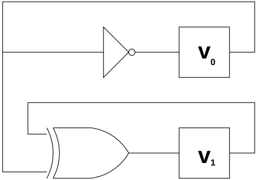
v_1	v_0	v_1'	v_0'
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0



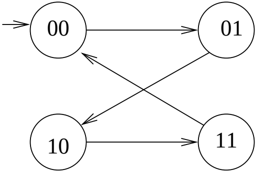
$\xi(R) = (v_0' \leftrightarrow \neg v_0) \land (v_1' \leftrightarrow v_0 \bigoplus v_1)$
$\xi(P) := (v_0 \leftrightarrow v_1)$ (i.e., $P = \{00, 11\}$)

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

Example: simple counter



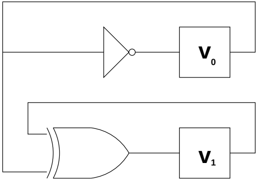
v_1	v_0	v'_1	v'_0
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0



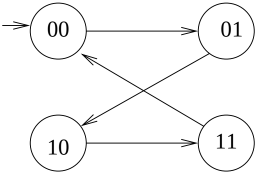
$\xi(R) = (v'_0 \leftrightarrow \neg v_0) \wedge (v'_1 \leftrightarrow v_0 \bigoplus v_1)$
$\xi(P) := (v_0 \leftrightarrow v_1)$ (i.e., $P = \{00, 11\}$)

$$
\begin{array}{ll}
\xi(PreImage(P, R)) & = \\
\exists V'.(\xi(P)[V'] \wedge \xi(R)[V, V']) & = \\
\exists v'_0 v'_1.((v'_0 \leftrightarrow v'_1) \wedge (v'_0 \leftrightarrow \neg v_0) \wedge (v'_1 \leftrightarrow v_0 \bigoplus v_1)) & = \\
\underbrace{(\neg v_0 \wedge v_0 \bigoplus v_1)}_{v'_0=\top, v'_1=\top} \vee \underbrace{\bot}_{v'_0=\top, v'_1=\bot} \vee \underbrace{\bot}_{v'_0=\bot, v'_1=\top} \vee \underbrace{(v_0 \wedge \neg(v_0 \bigoplus v_1))}_{v'_0=\bot, v'_1=\bot} & = \\
v_1 \quad (i.e., \{10, 11\}) &
\end{array}
$$

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

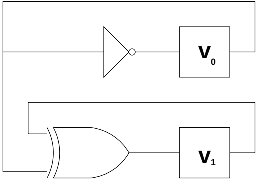


$\xi(R) = (v_0' \leftrightarrow \neg v_0) \wedge (v_1' \leftrightarrow v_0 \bigoplus v_1)$

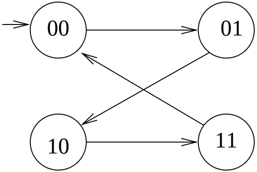
$\xi(P) := (v_0 \leftrightarrow v_1)$ (i.e., $P = \{00, 11\}$)

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

Example: simple counter



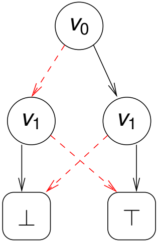
v_1	v_0	v_1'	v_0'
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0



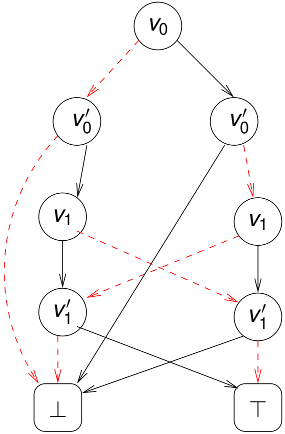
$\xi(R) = (v_0' \leftrightarrow \neg v_0) \wedge (v_1' \leftrightarrow v_0 \bigoplus v_1)$
$\xi(P) := (v_0 \leftrightarrow v_1)$ (i.e., $P = \{00, 11\}$)

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

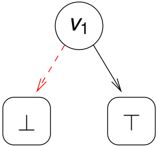
Pre-Image [cont.]



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



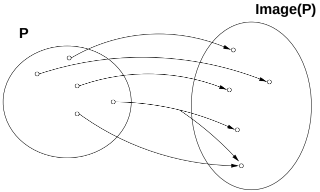
$\xi(R) = (v'_0 \leftrightarrow \neg v_0) \wedge (v'_1 \leftrightarrow v_0 \bigoplus v_1)$



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

Forward Image

- Forward image of a set:



Evaluate one-shot all transitions from the states of the set

- Set theoretic view:

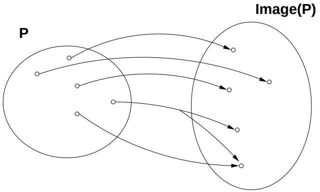
$$Image(P, R) := \{s' \mid \text{for some } s \in P, (s, s') \in R\}$$

- Logical Characterization:

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

Forward Image

- Forward image of a set:



Evaluate one-shot all transitions from the states of the set

- Set theoretic view:

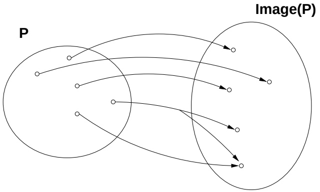
$$Image(P, R) := \{s' \mid \text{for some } s \in P, (s, s') \in R\}$$

- Logical Characterization:

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

Forward Image

- Forward image of a set:



Evaluate one-shot all transitions from the states of the set

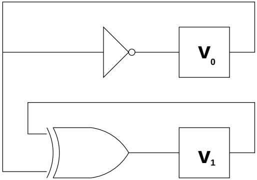
- Set theoretic view:

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

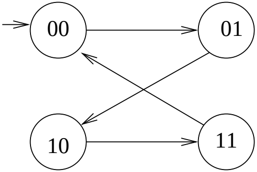
- Logical Characterization:

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

Example: simple counter

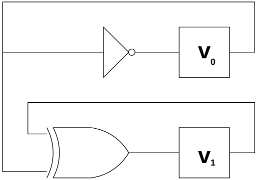


v_1	v_0	v_1'	v_0'
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

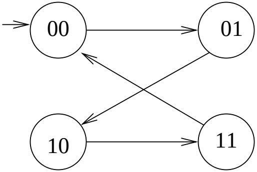


$\xi(R) = (v_0' \leftrightarrow \neg v_0) \wedge (v_1' \leftrightarrow v_0 \bigoplus v_1)$
$\xi(P) := (v_0 \leftrightarrow v_1)$ (i.e., $P = \{00, 11\}$)

Example: simple counter



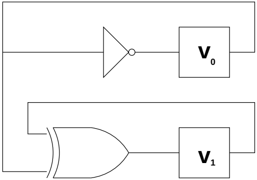
v_1	v_0	v_1'	v_0'
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0



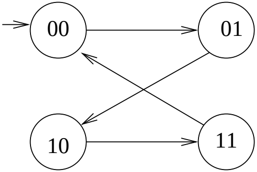
$\xi(R) = (v_0' \leftrightarrow \neg v_0) \wedge (v_1' \leftrightarrow v_0 \bigoplus v_1)$

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

Example: simple counter

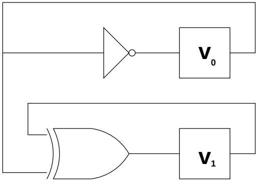


v_1	v_0	v_1'	v_0'
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

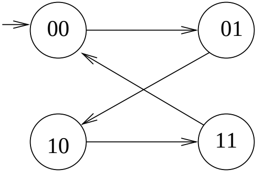


$\xi(R) = (v_0' \leftrightarrow \neg v_0) \wedge (v_1' \leftrightarrow v_0 \bigoplus v_1)$
$\xi(P) := (v_0 \leftrightarrow v_1)$ (i.e., $P = \{00, 11\}$)

Example: simple counter



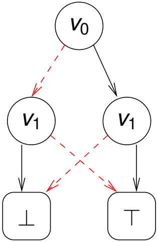
v_1	v_0	v_1'	v_0'
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0



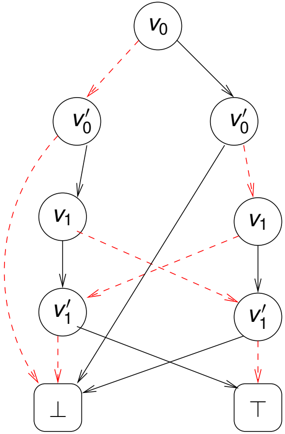
$\xi(R) = (v_0' \leftrightarrow \neg v_0) \wedge (v_1' \leftrightarrow v_0 \bigoplus v_1)$
$\xi(P) := (v_0 \leftrightarrow v_1)$ (i.e., $P = \{00, 11\}$)

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

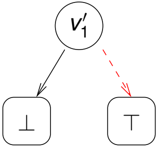
Forward Image [cont.]



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



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



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

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

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

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

Outline

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

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

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 $\neg$, $\wedge$, **EX**, **EU**, **EG**

- A general M.C. algorithm (fix-point):
  - (i) represent I and R as Boolean formulas $\xi(I), \xi(R)$
  - (ii) for every $\varphi_i \in Sub(\varphi)$, find $\xi([\varphi_i])$
  - (iii) 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 **EX**: handled by symbolic PreImage computation
  - other temporal operators **EG**, **EU**: handled by fix-point symbolic computation

General Schema

Assume φ written in terms of $\neg$, $\wedge$, **EX**, **EU**, **EG**

- A general M.C. algorithm (fix-point):
  (i) represent I and R as Boolean formulas $\xi(I), \xi(R)$
  (ii) for every $\varphi_i \in Sub(\varphi)$, find $\xi([\varphi_i])$
  (iii) 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 **EX**: handled by symbolic PreImage computation
  - other temporal operators **EG**, **EU**: handled by fix-point symbolic computation

General Schema

Assume φ written in terms of $\neg$, $\wedge$, **EX**, **EU**, **EG**

- A general M.C. algorithm (fix-point):
  - (i) represent I and R as Boolean formulas $\xi(I), \xi(R)$
  - (ii) for every $\varphi_i \in Sub(\varphi)$, find $\xi([\varphi_i])$
  - (iii) 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 **EX**: handled by symbolic PreImage computation
  - other temporal operators **EG**, **EU**: handled by fix-point symbolic computation

General Schema

Assume φ written in terms of $\neg$, $\wedge$, **EX**, **EU**, **EG**

- A general M.C. algorithm (fix-point):
  - (i) represent I and R as Boolean formulas $\xi(I), \xi(R)$
  - (ii) for every $\varphi_i \in Sub(\varphi)$, find $\xi([\varphi_i])$
  - (iii) 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 **EX**: handled by symbolic PreImage computation
  - other temporal operators **EG**, **EU**: handled by fix-point symbolic computation

General Schema

Assume φ written in terms of $\neg$, $\wedge$, **EX**, **EU**, **EG**

- A general M.C. algorithm (fix-point):

  (i) represent I and R as Boolean formulas $\xi(I), \xi(R)$

  (ii) for every $\varphi_i \in Sub(\varphi)$, find $\xi([\varphi_i])$

  (iii) 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 **EX**: handled by symbolic PreImage computation
  - other temporal operators **EG**, **EU**: handled by fix-point symbolic computation

General Schema

Assume φ written in terms of $\neg$, $\wedge$, **EX**, **EU**, **EG**

- A general M.C. algorithm (fix-point):
  (i) represent I and R as Boolean formulas $\xi(I), \xi(R)$
  (ii) for every $\varphi_i \in Sub(\varphi)$, find $\xi([\varphi_i])$
  (iii) 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 **EX**: handled by symbolic PreImage computation
  - other temporal operators **EG**, **EU**: handled by fix-point symbolic computation

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

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

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

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(apply(op,X_1,X_2))"

General M.C. Procedure

OBDD Check(CTL_formula β) {
  if (*In_OBDD_Hash*(β))
    return *OBDD_Get_From_Hash*(β);
  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));
  EXβ_1: **return** PreImage(Check(β_1));
  EGβ_1: **return** Check_EG(Check(β_1));
  E(β_1**U**β_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) {

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

$\quad$**repeat**

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

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

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

return $Y;$

}

Check_EU

```
OBDD Check_EU(OBDD X_1,X_2) {
    Y' := X_2; j := 1;
    repeat
        Y := Y'; j := j + 1;
        Y' := Y ∨ (X_1 ∧ PreImage(Y));
    until (Y' ↔ 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
  $\Longrightarrow$ problems of up to 10^{120} states handled!!

Outline

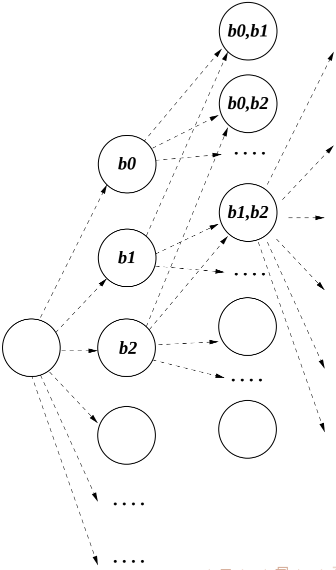
A simple example

```
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.]

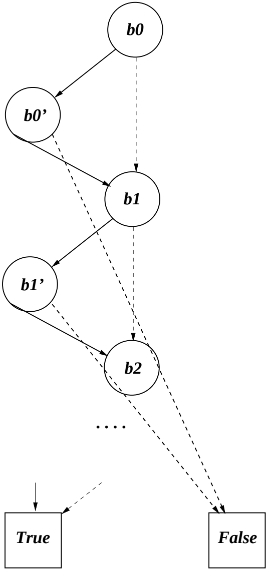
- N Boolean variables $b0, b1, \ldots$
- Initially, all variables set to 0
- Each variable can pass from 0 to 1, but not vice-versa
- 2^N states, all reachable
- (Simplified) model of a student career behaviour.

A simple example: FSM



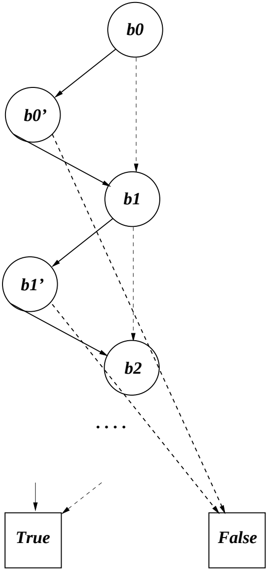
(transitive trans. omitted)
2^N STATES
$O(2^N)$ TRANSITIONS

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



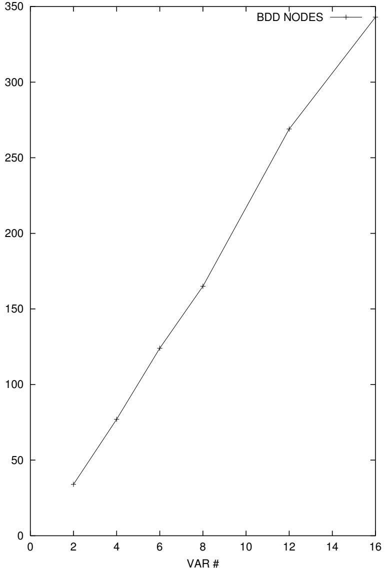
$2N + 2$ NODES

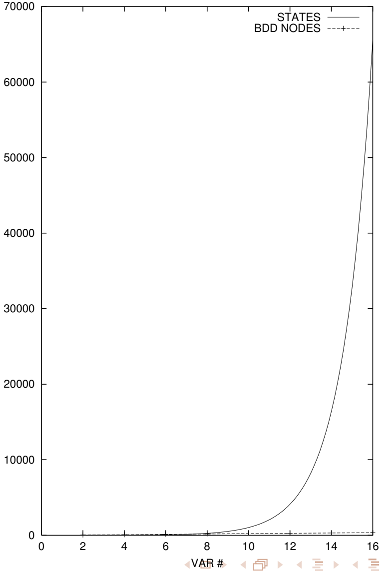
A simple example: $OBDD(\xi(R))$



$2N + 2$ NODES

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

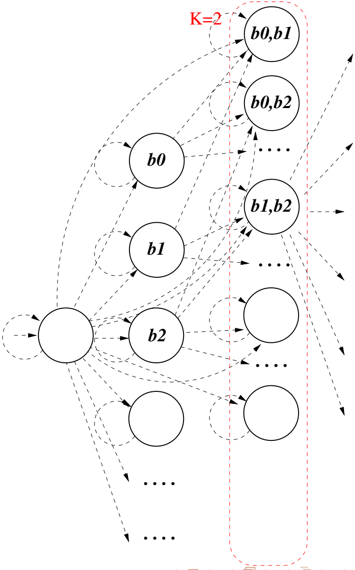




A simple example: reaching K bits true

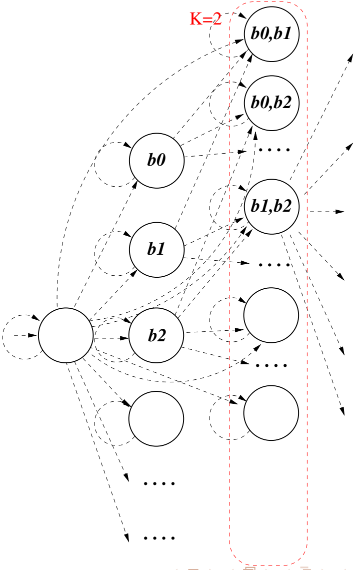
- Property $\mathbf{EF}(b0 + b1 + ... + b(N-1) \geq K)$ ($K \leq N$)
  (it may be reached a state in which K bits are true)
- E.g.: “it is reachable a state where K exams are passed”

A simple example: FSM



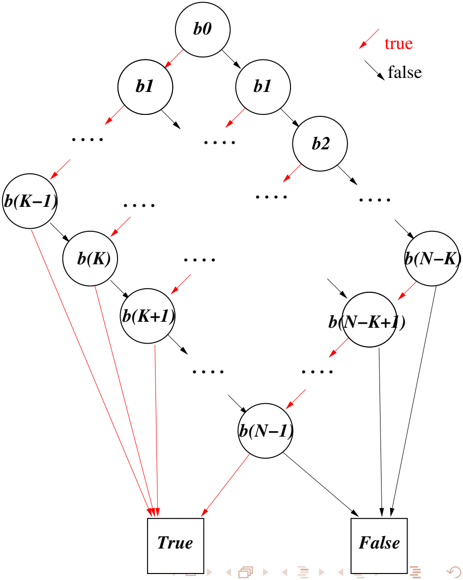
$\binom{N}{K} + \binom{N}{K+1} + \ldots + \binom{N}{N}$

A simple example: FSM



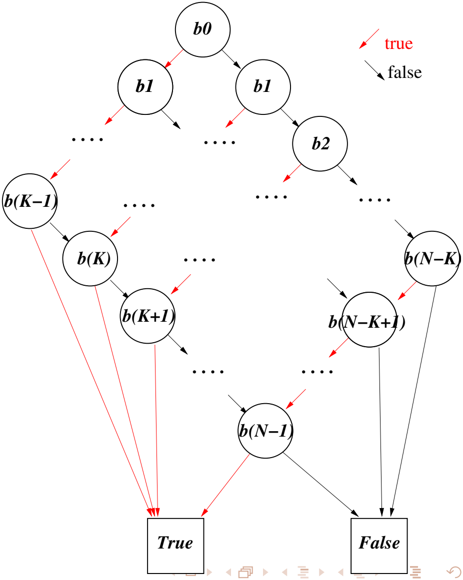
$$\binom{N}{K} + \binom{N}{K+1} + \ldots + \binom{N}{N}$$

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



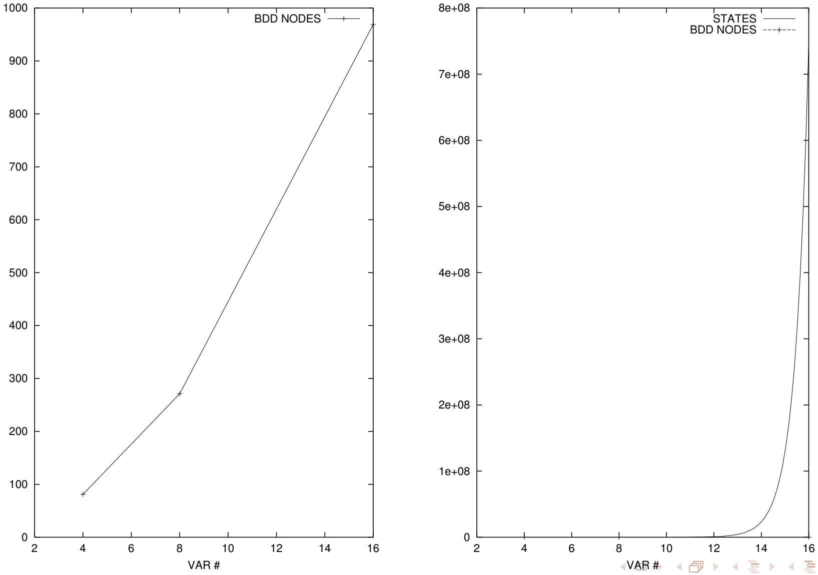
$(N-K+1)\cdot K+2$ NODES

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



$(N-K+1)\cdot K+2$ NODES

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



Outline

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

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]
  - ...

Outline

Ex: OBDDs

Let $\varphi \stackrel{\text{def}}{=} (A \wedge (B \vee C))$ and $\varphi' \stackrel{\text{def}}{=} \exists A.\forall B.\varphi$. Using the variable ordering "A, B, C", draw the OBDD corresponding to the formulas φ and φ'.

$\varphi \stackrel{\text{def}}{=} (A \wedge (B \vee C))$

Ex: OBDDs

Let $\varphi \stackrel{\text{def}}{=} (A \wedge (B \vee C))$ and $\varphi' \stackrel{\text{def}}{=} \exists A.\forall B.\varphi$. Using the variable ordering "A, B, C", draw the OBDD corresponding to the formulas φ and φ'.

$\varphi \stackrel{\text{def}}{=} (A \wedge (B \vee C))$

[Solution:

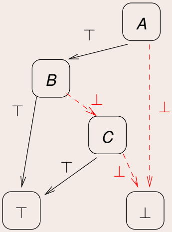
]

Ex: OBDDs

Let $\varphi \stackrel{\text{def}}{=} (A \wedge (B \vee C))$ and $\varphi' \stackrel{\text{def}}{=} \exists A.\forall B.\varphi$. Using the variable ordering "A, B, C", draw the OBDD corresponding to the formulas φ and φ'.

$\varphi \stackrel{\text{def}}{=} (A \wedge (B \vee C))$

[Solution:



]

Ex: OBDDs (cont.)

$\varphi' \stackrel{\text{def}}{=} \exists A.\forall B.(A \wedge (B \vee C))$

Ex: OBDDs (cont.)

$\varphi' \stackrel{\text{def}}{=} \exists A.\forall B.(A \wedge (B \vee C))$

[Solution:

]

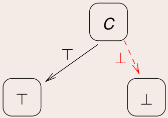
Ex: OBDDs (cont.)

$\varphi' \stackrel{\text{def}}{=} \exists A.\forall B.(A \wedge (B \vee C))$

[Solution:

$$
\begin{array}{lllllll}
\varphi' & \stackrel{\text{def}}{=} & \exists A.\forall B.\varphi & & & & \\
 & = & \forall B.(A \wedge (B \vee C)))[A := \top] & & & \vee & (\forall B.(A \wedge (B \vee C)))[A := \bot] \\
 & = & \forall B.(B \vee C)) & & & \vee & \forall B.\bot \\
 & = & ((B \vee C)[B := \top] & \wedge & (B \vee C)[B := \bot]) & \vee & \bot \\
 & = & (\top & \wedge & C) & & \\
 & = & C & & & &
\end{array}
$$

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 $P \stackrel{\text{def}}{=} (v_1 \wedge v_2)$. Write:

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 $P \stackrel{\text{def}}{=} (v_1 \wedge v_2)$. 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.

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 $P \stackrel{\text{def}}{=} (v_1 \wedge v_2)$. 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 $(\neg v_1 \wedge \neg v_2)$, $T(v_1, v_2, v_1', v_2')$ is $(v_1' \leftrightarrow \neg v_1) \wedge (v_2' \leftrightarrow (v_1 \leftrightarrow v_2))$]

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 $P \stackrel{\text{def}}{=} (v_1 \wedge v_2)$. 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 $(\neg v_1 \wedge \neg v_2)$, $T(v_1, v_2, v_1', v_2')$ is $(v_1' \leftrightarrow \neg v_1) \wedge (v_2' \leftrightarrow (v_1 \leftrightarrow v_2))$]

- the graph representing the FSM. (Assume the notation "$v_1 v_2$" for labeling the states: e.g. "10" means "$v_1 = 1, v_2 = 0$".)

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 $P \stackrel{\text{def}}{=} (v_1 \wedge v_2)$. 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 $(\neg v_1 \wedge \neg v_2)$, $T(v_1, v_2, v_1', v_2')$ is $(v_1' \leftrightarrow \neg v_1) \wedge (v_2' \leftrightarrow (v_1 \leftrightarrow v_2))$]

- 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:

  ]

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))
```

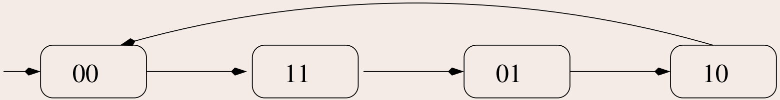
and consider the property $P \stackrel{\text{def}}{=} (v_1 \wedge v_2)$. Write:

- 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 $(\neg v_1 \wedge \neg v_2)$, $T(v_1, v_2, v_1', v_2')$ is $(v_1' \leftrightarrow \neg v_1) \wedge (v_2' \leftrightarrow (v_1 \leftrightarrow v_2))$]

- 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:



]

Ex: Symbolic CTL Model Checking (cont.)

- the Boolean formula representing symbolically $\mathbf{EX}P$. [The formula must be computed symbolically, not simply inferred from the graph of the previous question!]

Ex: Symbolic CTL Model Checking (cont.)

- the Boolean formula representing symbolically **EX**P. [The formula must be computed symbolically, not simply inferred from the graph of the previous question!]

  [Solution:

$$
\begin{array}{rcl}
\mathbf{EX}(P) & = & \exists v_1', v_2'.(T(v_1, v_2, v_1', v_2') \wedge P(v_1', v_2')) \\
 & = & \exists v_1', v_2'.((v_1' \leftrightarrow \neg v_1) \wedge (v_2' \leftrightarrow (v_1 \leftrightarrow v_2)) \wedge \underbrace{(v_1' \wedge v_2')}_{\Longrightarrow v_1'=\top, v_2'=\top}) \\
 & = & \overbrace{(\neg v_1 \wedge \neg v_2)}^{v_1'=\top, v_2'=\top} \vee \bot \vee \bot \vee \bot \\
 & = & (\neg v_1 \wedge \neg v_2)
\end{array}
$$

  .]

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

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 $(v_1 \leftrightarrow v_2') \wedge (v_2 \leftrightarrow v_1')$]

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

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 $(v_1 \leftrightarrow v_2') \wedge (v_2 \leftrightarrow v_1')$]
- the graph representing the FSM. (Assume the notation "$v_1 v_2$" for labeling the states. E.g., "10" means "$v_1 = 1, v_2 = 0$".)

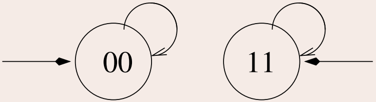
  [Solution:]

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 $(v_1 \leftrightarrow v_2') \wedge (v_2 \leftrightarrow v_1')$]
- 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: 

  ]

Ex: Symbolic CTL Model Checking (cont.)

- 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).

Ex: Symbolic CTL Model Checking (cont.)

- 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:

$$\begin{array}{rcl}
R^1(v_1', v_2') & = & \exists v_1, v_2.(I(v_1, v_2) \wedge T(v_1, v_2, v_1', v_2')) \\
& = & \exists v_1, v_2.((v_1 \leftrightarrow v_2) \wedge (v_1 \leftrightarrow v_2') \wedge (v_2 \leftrightarrow v_1')) \\
& = & ((v_1 \leftrightarrow v_2) \wedge (v_1 \leftrightarrow v_2') \wedge (v_2 \leftrightarrow v_1'))[v_1 = \bot, v_2 = \bot] \vee \\
& & ((v_1 \leftrightarrow v_2) \wedge (v_1 \leftrightarrow v_2') \wedge (v_2 \leftrightarrow v_1'))[v_1 = \bot, v_2 = \top] \vee \\
& & ((v_1 \leftrightarrow v_2) \wedge (v_1 \leftrightarrow v_2') \wedge (v_2 \leftrightarrow v_1'))[v_1 = \top, v_2 = \bot] \vee \\
& & ((v_1 \leftrightarrow v_2) \wedge (v_1 \leftrightarrow v_2') \wedge (v_2 \leftrightarrow v_1'))[v_1 = \top, v_2 = \top] \\
& = & (\neg v_1' \wedge \neg v_2') \vee \bot \vee \bot \vee (v_1' \wedge v_2') \\
& = & (\neg v_1' \wedge \neg v_2') \vee (v_1' \wedge v_2') \\
& = & (v_1' \leftrightarrow v_2')
\end{array}$$

  .]