Session 10
Algorithms for Model Checking CTL

1 BDDs: binary decision diagrams

Symbolic Model Checking

- The states of a transition system correspond to the possible values of variables (of the modelled program) and their number can be exponential in the number of those variables
- The labelling algorithm uses intensively operations on sets of states and in particular for computing the sets of successors and predecessors of a state.
- The success of model checking only possible because it is possible to manipulate sets in a symbolic manner
- One can represent sets of states in binary encodings that correspond to Boolean functions
- and an efficient way to represent Boolean functions is using OBDDs (*ordered binary decision diagrams*).
- other option is to use SAT solvers

OBDDs:*ordered binary decision diagrams*

- OBDDs represent Boolean functions

$$f : \{0,1\}^n \to \{0,1\}$$

- Boolean Operations:

  conjunction $f \cdot g$ or $f \wedge g$

  disjunction $f + g$ or $f \vee g$

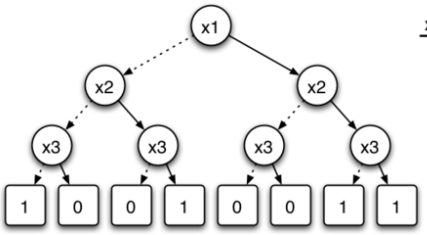
  complement $\bar{f}$ or $\neg f$

  ...

- A Boolean function f with n arguments can be represented by a truth table with 2^n lines.
- It can also be represented by a propositional formula but that is also not efficient

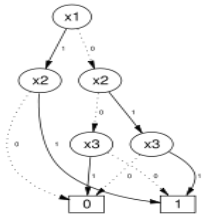
Boolean Functions and BDDs

Boolean function



x1	x2	x3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Decision tree

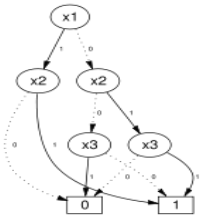


BDD

BDD: Binary decision diagrams

A Boolean function can be represented by a DAG with a root and

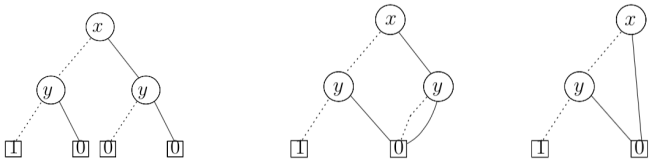
- decision nodes (intern)
- only two terminal (leaves) nodes: 0 and 1.
- each decision node n is labelled with a propositional variable and has two sons: *dashed* arcs (and `solid` arcs correspond to the possible truth values of the variable assignments (0 or 1). Its sons are respectively $lo(n)$ and $hi(n)$



Reduced BDDs

A BDD is reduced if isomorphic subgraphs are identified and do not have nodes whose sons are isomorphic.

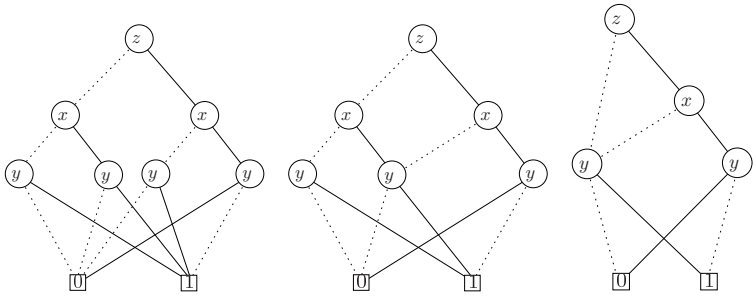
R1 Redundant Tests: both arcs of a node n have the same target node ; n can be eliminate



R2 Redundant decision nodes: if they are roots of (structurally) identical subBDDs; one can be eliminated.

This ensures only two leave nodes

BDDs Reduction

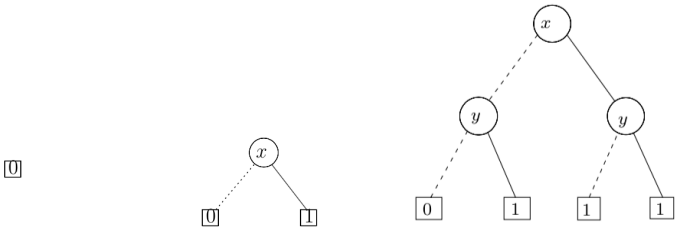


Efficiency of BDDs representation

- Compact representation
- Checking Satisfiability: determine if there is a consistent path from the root that ends in node 1. A path is consistent if for every variable, has only dashed lines or solid lines leaving nodes labelled by that variable
- Checking Validity: no terminal node 0 is reached by consistent paths.
- Conjunction: given B_f and B_g representing two Boolean functions f and g, build a BDD for $f \cdot g$ such that each node 1 of B_f is replaced by B_g.
- Disjunction: as before, but replacing all nodes 0 of B_f by B_g (BDD for $f + g$)
- Complement: $B_{\bar{f}}$ is obtained from B_f, replacing the terminal nodes 0 by 1 and vice-versa.

B_0, B_1, B_x and complement

B_1 is the same as B_0 but with a 1



Show that the third corresponds to $f_1(x, y) = x \vee y$ and compute BDDs for $f(x, y) = \neg(x \vee y)$ and $f_2(x, y) = x \wedge y$.

1.1 Ordered BDDs

OBDD: Ordered Binary Decision Diagrams

Ordered BDD

If the variables occur always in the same order along any path from the root. This induces an ordering in the set of variables.

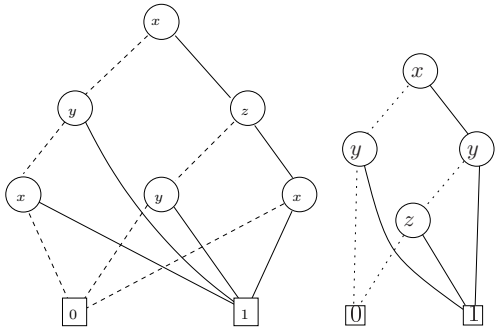
Let $[x_1, \ldots, x_n]$ be a ordered list of variables. A BDD B has order $[x_1, \ldots, x_n]$ if for any occurrence of x_i followed by x_j along a path in B, we have $i < j$.

A BDD is ordered if there is an ordering for the list of its variables. Two orderings of two BDDs B_1 and B_2 are compatibles if do not exist two variables x and y such that $x < y$ in B_1 but $y < x$ in B_2.

Teorema 10.1. *A reduced OBDD representing a given Boolean function f is unique up to isomorphism. That is, two OBDDs B and B' with compatible variable orderings represent the same function if they have the identical structure (*canonical form*). In that case, they are equivalent.*

OBDDs

This two BDDs are equivalent:



But only one is ordered: $[x, y, z]$. For the first it is not possible to find an ordering: an ordering can only exist if there are no multiple occurrences of a same variable along a path

Importance of a Canonical Form

Absence of redundant variables (i.e. from which the function does not depend on)

Test for equivalence Two functions f and g with OBDDs having compatible orderings are equivalent if the reduced OBDDS are isomorphic.

Test for Validity If f is a tautology its reduced OBDD has a unique node with the value 1 (B_1).

Test for Consequence to test if g implies f, compute the reduced OBDD for $f \cdot \bar{g}$ and check if it is B_0.

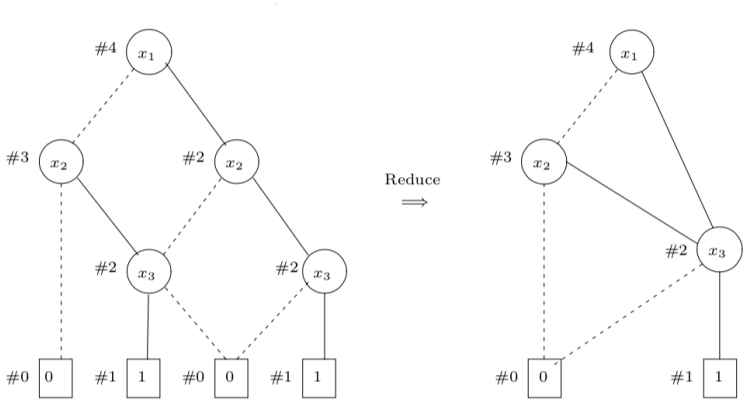
Test for Satisfiability A Boolean function f is satisfiable iff its reduced OBDD is not B_0.

Algorithm for reducing OBDDs: `reduce`

- REDUCE(B_f):
- If B as the order $[x_1, \ldots, x_l]$, B has at most $l + 1$ levels
- Starting bottom-up at terminal nodes,
- Each node n is labelled with $i(n)$ in such a way that two nodes have the same label iff the respective sub-OBDDs represent the same Boolean function
  - Assign the label #0 to all leaves with value 0 and #1 to all leaves with value 1.
  - If $i(lo(n)) = i(hi(n))$, then set $i(n)$ to the same label. i.e. node n can be eliminated as it is redundant.

  - If there exist a node m with the same variable x_i, such that $i(lo(n)) = i(lo(m))$ and $i(hi(n)) = i(hi(m))$, then $i(n) = i(m)$.
  - If none of the cases before apply, assign to n a label with the next unused integer
- in the end remove nodes with the same label, starting bottom up again, redirecting the edges accordingly.

Algorithm for reducing OBDDs: `reduce`



OBDD

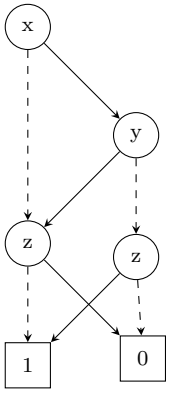
For each of the following Boolean functions, determine reduced OBDD's for each of the orders $[x, y, z]$ and $[z, y, x]$

First compute the binary decision tree and then apply the REDUCE algorithm.

a)

x	y	z	$f(x,y,z)$
1	1	1	0
1	1	0	1
1	0	1	1
1	0	0	0
0	1	1	0
0	1	0	1
0	0	1	0
0	0	0	1

b) $f(x, y, z) = x \cdot (y + \overline{z})$.



x	y	z	$f(x,y,z)$
1	1	1	0
1	1	0	1
1	0	1	1
1	0	0	0
0	1	1	0
0	1	0	1
0	0	1	0
0	0	0	1

Shannon Expansion

Definição 10.1. *Let f be a Boolean formula and x a variable.*

1. *$f[0/x]$ obtained by replacing all occurrences of x in f by* 0
2. *$f[1/x]$obtained by replacing all occurrences of x in f by* 1

Lema 10.1 (Shannon Expansion)**.**

$$f \equiv \bar{x} \cdot f[0/x] + x \cdot f[1/x]$$

or, equivalenly, we write

$$f = x \rightarrow f[1/x], f[0/x],$$

that corresponds to the statement IF...THEN...ELSE *and can be also denoted* ITE$(x, f[0/x], f[1/x])$*.*

Shannon Expansion

For any Boolean binary operator *op*

$$f \ op \ g \quad = \quad \bar{x_i} \cdot (f[0/x_i] \ op \ g[0/x_i]) + x_i \cdot (f[1/x_i] \ op \ g[1/x_i])$$

or, equivalently,

$$x_i \rightarrow f[1/x_i] \ op \ g[1/x_i], f[0/x_i] \ op \ g[0/x_i],$$

All Boolean functions can be written with this operator $\rightarrow$, considering that $x_i \rightarrow f, f \equiv f$. From a formula using only this operator (and the Boolean constants) a BDD (decision tree) is easily constructed.

Algorithm of application for OBDDs

APPLY(op,B_f,B_g): where OP is a binary Boolean operation.

Idea:

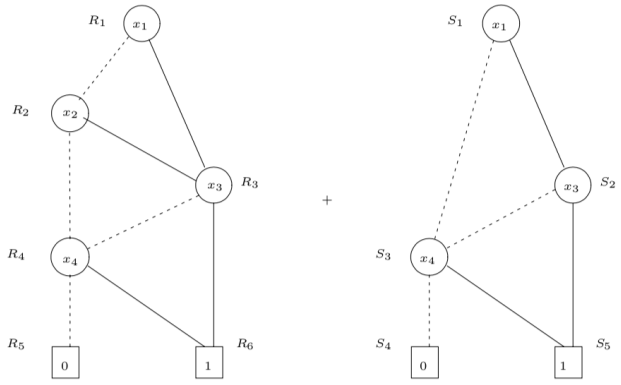
- let v be variable with the highest order in B_f or B_g
- split the problem into two subproblems for v being 0 and v being 1 (Shannon expansion for the sub-nodes lo and hi)
- at the leaves apply the Boolean operation op

apply

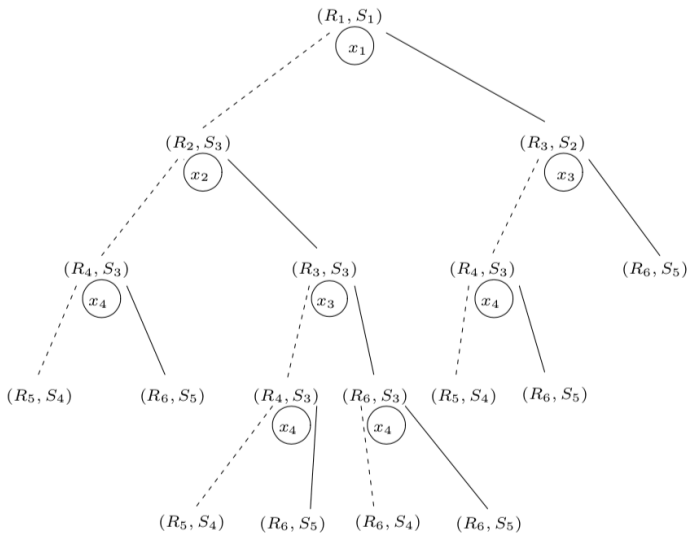
APPLY(op,B_f,B_g): Let r_f and r_g be the roots of B_f and B_g, respectively. To compute the OBDD $B_{f\ op\ g}$ (usually not reduced) apply the following steps:

- If both are leaves with values l_f and l_g, respectively, (in $\{0,1\}$), then $B_{f\ op\ g} = B_0$ if $l_f\ op\ l_g = 0$ e $B_{f\ op\ g} = B_1$, otherwise.
- In the remaining cases, at least one of the root nodes is not a leave. If both roots are x_i-nodes, create a x_i-node with a dashed arc for the OBDD APPLY(op,$lo(r_f)$,$lo(r_g)$) and a solid arc for the OBDD APPLY(op,$hi(r_f)$,$hi(r_g)$). This corresponds to Shannon expansion.
- If r_f is x_i-node and r_g a leave or a x_j-node with $j > i$ (ordering $[x_1, \ldots, x_n]$), then create a x_i-node with a dashed arc for the APPLY(op,$lo(r_f)$,r_g) and a solid arc for the OBDD APPLY(op,$hi(r_f)$,r_g). In this case g does not depend on x_i and $g \equiv g[0/x_i] \equiv g[1/x_i]$.
- simmetricaly for r_g

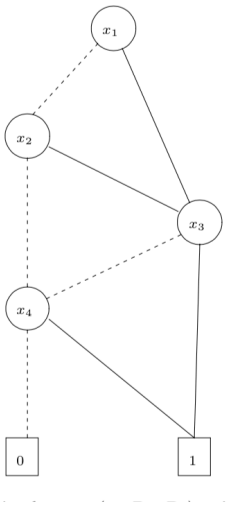
Algoritmos para OBDDs: apply$(+, B_f, B_g)$



Algoritmos para OBDDs: apply



Algoritmos para OBDDs: apply



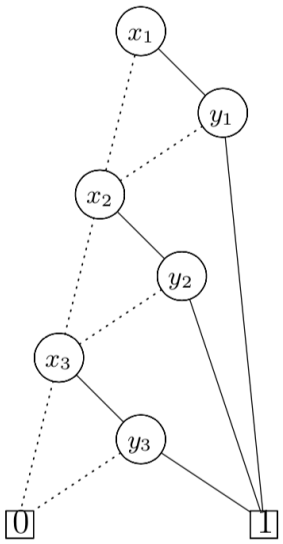
Algorithm of restriction for OBDDs: $f[val/x]$

RESTRICT(val,x,B_f):

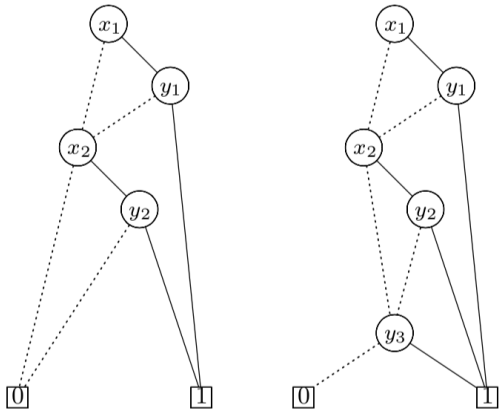
- To compute the OBDD $B_{f[0/x]}$ redirect the arcs that point to a x-node n for the node $lo(n)$ and remove the node n. Reduce.

- To compute the OBDD $B_{f[1/x]}$ redirect the arcs that point to a x-node n for the node $hi(n)$ and remove the node n. Reduce.

$f = x_1 \cdot y_1 + x_2 \cdot y_2 + x_3 \cdot y_3$



restrict$(0, x_3, B_f)$ **and restrict**$(1, x_3, B_f)$



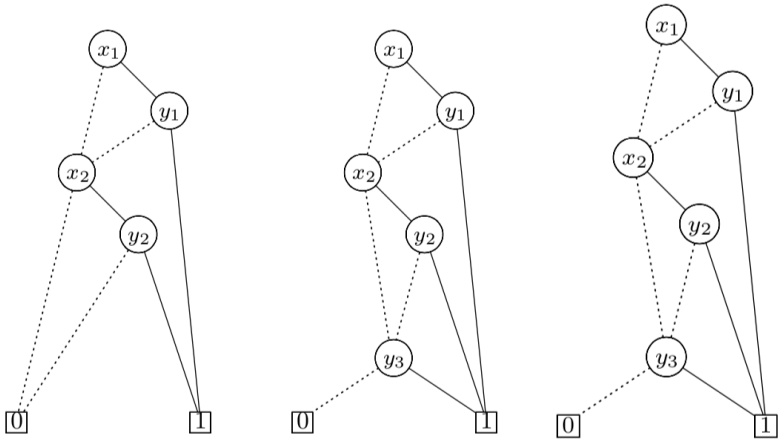
Algorithm exists for OBDDs

$\exists x.f$ is true if f is could be made true putting x to 0 or 1, i.e. if there exists a value for x that makes f true. Dually $\forall x.f$ is defined

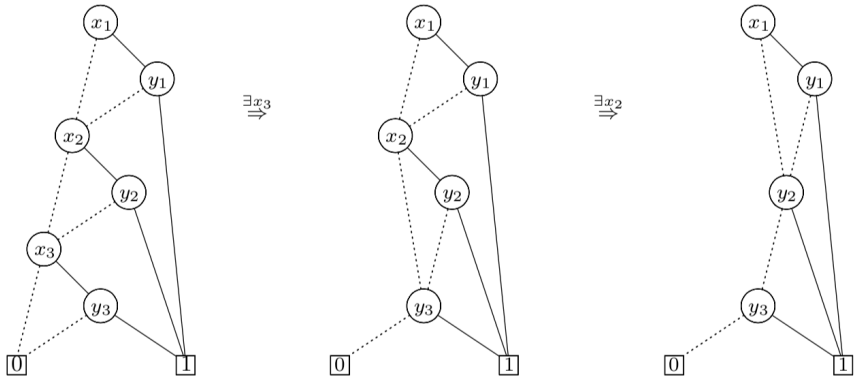
$$
\begin{aligned}
\exists x.f &\equiv f[0/x] + f[1/x] \\
\textsc{exists}(x, B_f) &= \textsc{apply}(+, B_{f[0/x]}, B_{f[1/x]}) \\
\forall x.f &\equiv f[0/x] \cdot f[1/x] \\
\textsc{forall}(x, B_f) &= \textsc{apply}(\cdot, B_{f[0/x]}, B_{f[1/x]})
\end{aligned}
$$

Note: OBDD for $\textsc{exists}(x, B_f)$ can be obtained from B_f replacing each the node x (n) by the OBDD $\textsc{apply}(+, lo(n), hi(n))$. It als o generalizes to $\exists x_1, \ldots \exists x_n.f$.

$\mathbf{exists}(x_3, B_f) = \mathbf{apply}(+, restrict(0, x_3, B_f), \mathbf{restrict}(1, x_3, B_f))$



$\mathbf{exists}(x_2, \mathbf{exists}(x_3, B_f))$



Boolean Formulae and OBDDs

Boolean formula f	OBDD B_f that represents
0	B_0
1	B_1
x	B_x
$\bar{f}$	swap 0 and 1 inB_f
$f + g$	APPLY $(+, B_f, B_g)$
$f \cdot g$	APPLY $(\cdot, B_f, B_g)$
$f \oplus g$	APPLY $(\oplus, B_f, B_g)$
$f[1/x]$	RESTRICT $(1, x, B_f)$
$f[0/x]$	RESTRICT $(0, x, B_f)$
$\exists x.f$	APPLY $(+, B_{f[0/x]}, B_{f[1/x]})$
$\forall x.f$	APPLY $(\cdot, B_{f[0/x]}, B_{f[1/x]})$

Computational Complexity for OBDDs

Algorithm	Input OBDD(s)	Output OBDD	Time complexity
REDUCE	B	reduced B	$O(\|B\| \cdot \log \|B\|)$
APPLY	B_f, B_g (reduced)	$B_{f \text{ op } g}$ (reduced)	$O(\|B_f\| \cdot \|B_g\|)$
RESTRICT	B_f (reduced)	$B_{f[0/x]}$ or $B_{f[1/x]}$ (reduced)	$O(\|B_f\| \cdot \log \|B_f\|)$
$\exists$	B_f (reduced)	$B_{\exists x_1.\exists x_2....\exists x_n.f}$ (reduced)	NP-complete

Exercise

Consider the functions $f(x,y) = x + y$, $g(x,y) = \overline{x} \cdot \overline{y}$ e $h(x,y,z) = x \cdot y + \overline{z} \cdot \overline{x}$.

- Determine reduced OBDD's B_f, B_g and B_h with the order $[x, y, z]$.
- Determine $B_{\overline{f}}$.
- Determine B_{f+g} applying for that the algorithm `apply` a B_f e B_g e reduzindo em seguida.
- Determine $B_{\exists yh}$ and $B_{\forall yh}$.

Transition systems using OBDDs

Let $T = (S, \longrightarrow, I, AP, L)$ be a model with $|AP| = n$.

- Each state $s \in S$ can be represented by the set of propositional variables $L(s)$, i.e. by a Boolean tuple

  $$(v_1, \ldots, v_n)$$

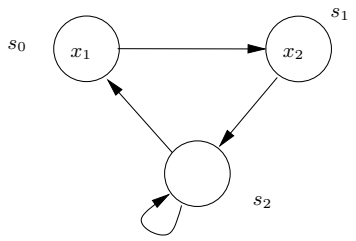
  such that $v_i = 1$ if $x_i \in L(s)$ and $v_i = 0$, otherwise.
- This implies that L has to be injective, which is easily achieved introducing new propositional variables.
- Each state s can be associated to a OBDD of the Boolean function

  $$f_s(x_1, \ldots x_n) = l_1 \cdot l_2 \cdots l_n$$

  where $l_i = x_i$ if $x_i \in L(s)$ and $\bar{x_i}$, otherwise (i.e., a valuation for x_i, $i = 1..n$).
- The set of states $\{s_1, \ldots, s_m\}$ can be represented by the OBDD for the Boolean function:

  $$l_{11} \cdot l_{12} \cdot \cdots \cdot l_{1n} + \cdots + l_{m1} \cdot l_{m2} \cdot \cdots \cdot l_{mn}$$

Example I



sets of states	representation by Boolean values	representation by Boolean function
$\emptyset$		0
$\{s_0\}$	$(1,0)$	$x_1 \cdot \bar{x_2}$
$\{s_1\}$	$(0,1)$	$\bar{x_1} \cdot x_2$
$\{s_2\}$	$(0,0)$	$\bar{x_1} \cdot \bar{x_2}$
$\{s_0, s_1\}$	$(1,0)$, $(0,1)$	$x_1 \cdot \bar{x_2} + \bar{x_1} \cdot x_2$
$\{s_0, s_2\}$	$(1,0)$, $(0,0)$	$x_1 \cdot \bar{x_2} + \bar{x_1} \cdot \bar{x_2}$
$\{s_1, s_2\}$	$(0,1)$, $(0,0)$	$\bar{x_1} \cdot x_2 + \bar{x_1} \cdot \bar{x_2}$
$S = \{s_0, s_1, s_2\}$	$(1,0)$, $(0,1)$, $(0,0)$	$x_1 \cdot \bar{x_2} + \bar{x_1} \cdot x_2 + \bar{x_1} \cdot \bar{x_2}$

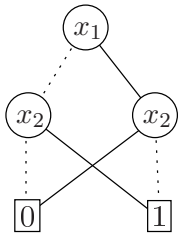
Compact Representation using OBDDs

The Boolean functions representing sets of states can be compactly represented by OBDDs.

For instance the state $\{s_0, s_1\}$, corresponds to

$$x_1 \cdot \bar{x_2} + \bar{x_1} \cdot x_2$$

pode ser representado por:



Set Operations using OBDDs

For implementing the labelling algorithm using OBDDs we need to express set operations:

- To test if $s \in S'$ this means that the valuation represented by s satisfies $B_{S'}$, i.e. we need to test that $B_s \cdot B_{S'}$ is satisfiable (i.e the reduced OBDD is not B_0).
- Intersection, union and complementation are implemented as the Boolean operations $\cdot$, $+$, and $^-$ respectively and
- with OBDDs using the function `apply`.

Representing the Transition Relation using OBDDs

- The transition relation $\longrightarrow$ is a subset of $S \times S$.
- Then a transition $s \longrightarrow s'$ can be represented by a pair of Boolean vectors

$$((v_1, \ldots, v_n), (v'_1, \ldots, v'_n)),$$

- to represent elements of the second component we use a copy of the atomic propositions $x'_1, \ldots, x'_n$
- the Boolean function is

$$(l_1 \cdot l_2 \cdots l_n).(l'_1 \cdot l'_2 \cdots l'_n)$$

  such that l_i is defined as above and $l'_i = x'_i$ se $x_i \in L(s')$, $l'_i = \bar{x'_i}$, otherwise.
- The transition relation is represented by the disjunction of these formulae
- from that one can build a OBDD for $\longrightarrow$.

$[x_1, x_2, x'_1, x'_2]$

x_1	x_2	x'_1	x'_2	$\longrightarrow$
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

$[x_1, x'_1, x_2, x'_2]$

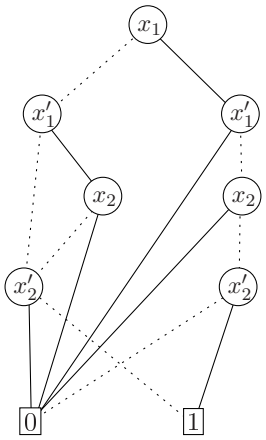
x_1	x'_1	x_2	x'_2	$\rightarrow$
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

$$f_{\overrightarrow{left}} = \bar{x_1} \cdot \bar{x_2} \cdot \bar{x'_1} \cdot \bar{x'_2} + \bar{x_1} \cdot \bar{x_2} \cdot x'_1 \cdot \bar{x'_2} + x_1 \cdot \bar{x_2} \cdot \bar{x'_1} \cdot x'_2 + \bar{x_1} \cdot x_2 \cdot \bar{x'_1} \cdot \bar{x'_2}$$

$$f_{\overrightarrow{right}} = \bar{x_1} \cdot \bar{x'_1} \cdot \bar{x_2} \cdot \bar{x'_2} + \bar{x_1} \cdot \bar{x'_1} \cdot x_2 \cdot \bar{x'_2} + \bar{x_1} \cdot x'_1 \cdot \bar{x_2} \cdot \bar{x'_2} + x_1 \cdot \bar{x'_1} \cdot \bar{x_2} \cdot \bar{x'_2}$$

Representing the Transition Relation using OBDDs

Using the truth table on the right (where the unprimed and primed variables are interleaved) we obtain the following OBDD for $\longrightarrow$ of Example I:



Representation of $pre_\exists$ and $pre_\forall$ in OBDDs

- finally we need OBDDs for the functions

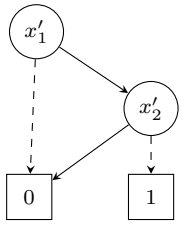
$$\begin{aligned} pre_\exists(X) &= \{s \in S \mid \exists s'\, s \longrightarrow s' \wedge s' \in X\} \\ pre_\forall(X) &= \{s \in S \mid \forall s'(s \longrightarrow s') \Rightarrow s' \in X)\} \end{aligned}$$

- which can be obtained using B_X and $B_{\longrightarrow}$.
- As $pre_\forall(X) = S \setminus pre_\exists(S \setminus X)$, it is enough to for consider $pre_\exists(X)$:
  - consider $B_{X'}$
  - determine `exists(`$\vec{x}'$`,apply(`$\cdot$`,`$B_{\longrightarrow}$`,`$B_{X'}$`))`.
- where in general $\exists \vec{x} f$ represents $\exists x_1 \ldots \exists x_n f$, for $\vec{x} = (x_1, \ldots, x_n)$, i.e. there exists a valuation for $\vec{x}$ that turns f true
- in this case, such valuation corresponds to a state s' $(l_1' \cdot l_2' \cdots l_n')$ such that $B_{\longrightarrow} \cdot B_{X'}$ represents the set of states, s $(l_1 \cdot l_2 \cdots l_n)$, such that $(l_1 \cdot l_2 \cdots l_n) \cdot (l_1' \cdot l_2' \cdots l_n')$ is true.

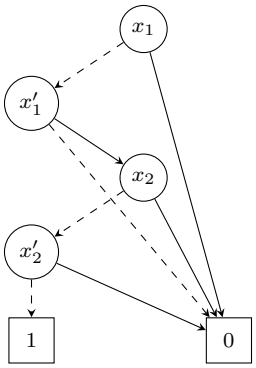
Compute $Sat(\mathbf{EX}x_1)$ using OBDDs

- We have that $Sat(x_1) = \{s_0\}$
- and we need to compute $pre_\exists(Sat(x_1))$.

- which corresponds to the OBDD for $\text{EXISTS}(x'_1, \text{EXISTS}(x'_2, \text{APPLY}(\cdot, B \longrightarrow, B_{s'_0}))$.
- For $B_{s'_0}$ we have

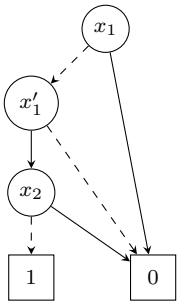


$\mathbf{apply}(\cdot, B \longrightarrow, B_{s'_0})$



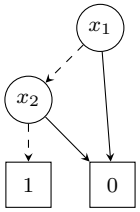
$\mathbf{exists}(x'_2, \mathbf{apply}(\cdot, B \longrightarrow, B_{s'_0}))$

We substitute the node x'_2 by $\text{APPLY}(+, B_0, B_1) = B_1$



$Sat(\mathbf{EX}x_1)$

Finally we have the OBDD for $\text{EXISTS}(x'_1, \text{EXISTS}(x'_2, \text{APPLY}(\cdot, B \longrightarrow, B_{s'_0}))$.



which is exactly the OBDD for the set of states $\{s_2\}$ showing that $Sat(\text{EX}x_1) = \{s_2\}$

Example II

Consider the model $\mathcal{M} = (S = \{s_0, s_1, s_2, s_3\}, \{s_0 \to s_2, s_0 \to s_1, s_1 \to s_1, s_1 \to s_2, s_1 \to s_3, s_2 \to s_0, s_2 \to s_1, s_2 \to s_2, s_3 \to s_0, s_3 \to s_3\}, L(s_0) = \{x_1, x_2\}, L(s_1) = \{x_1\}, L(s_2) = \{\}, L(s_3) = \{x_2\}$.

1. Using the order $[x_1, x_2]$, determine (reduced) OBDD's for representing the sets of states $\{s_0, s_1\}$ e $\{s_0, s_2\}$.
2. Determine the truth table for the transition function using the order $[x_1, x_1', x_2, x_2']$.
3. Draw a (reduced) OBDD for the transition function.
4. Apply the the labelling algorithm (adapted to the OBDD's representation and using the order $[x_1, x_2]$) to the model $\mathcal{M}$, to determine the sets of states where the following formulae are true.
   - EX x_2;
   - AG $(x_1 \vee x_2)$;
   - E $(x_2$ U $x_1)$.

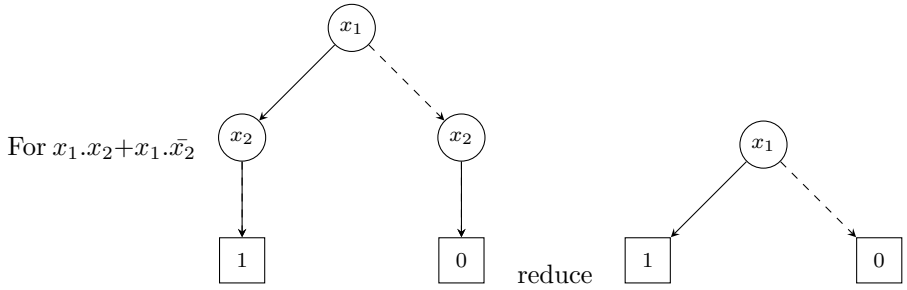
Set Representations

Consider the model $\mathcal{M} = (S = \{s_0, s_1, s_2, s_3\}, \{s_0 \to s_2, s_0 \to s_1, s_1 \to s_1, s_1 \to s_2, s_1 \to s_3, s_2 \to s_0, s_2 \to s_1, s_2 \to s_2, s_3 \to s_0, s_3 \to s_3\}, L(s_0) = \{x_1, x_2\}, L(s_1) = \{x_1\}, L(s_2) = \{\}, L(s_3) = \{x_2\}$.

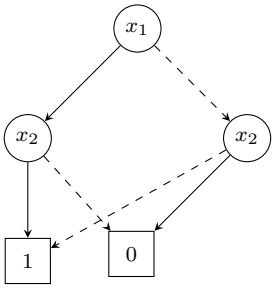
Using the order $[x_1, x_2]$, determine (reduced) OBDD's for representing the sets of states $\{s_0, s_1\}$ e $\{s_0, s_2\}$

$$\begin{array}{rcl} \{s_0\} & | & x_1.x_2 \\ \{s_1\} & | & x_1.\bar{x_2} \\ \{s_2\} & | & \bar{x_1}.\bar{x_2} \\ \{s_0, s_1\} & | & x_1.x_2 + x_1.\bar{x_2} \\ \{s_0, s_2\} & | & x_1.x_2 + \bar{x_1}.\bar{x_2} \end{array}$$

For $x_1.x_2 + x_1.\bar{x_2}$



For $x_1.x_2 + \bar{x_1}.\bar{x_2}$ is already reduced



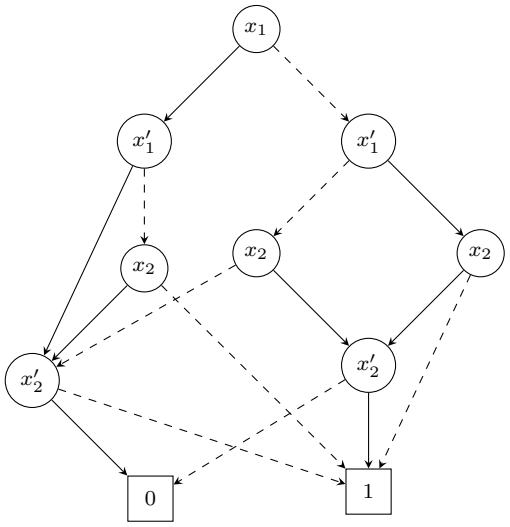
Determine the truth table for the transition function using the order $[x_1, x_1', x_2, x_2']$. We have for $\{s_0\}$, $x_1.x_2$, $\{s_1\}$, $x_1.\bar{x_2}$, $\{s_2\}$, $\bar{x_1}.\bar{x_2}$ and $\{s_3\}$, $\bar{x_1}.x_2$.

$s_0 \to s_2$	$x_1.x_2.\bar{x_1'}.\bar{x_2'}$
$s_0 \to s_1$	$x_1.x_2.x_1'.\bar{x_2'}$
$s_1 \to s_1$	$x_1.\bar{x_2}.x_1'.\bar{x_2'}$
$s_1 \to s_2$	$x_1.\bar{x_2}.\bar{x_1'}.\bar{x_2'}$
$s_1 \to s_3$	$x_1.\bar{x_2}.\bar{x_1'}.x_2'$
$s_2 \to s_0$	$\bar{x_1}.\bar{x_2}.x_1'.x_2'$
$s_2 \to s_1$	$\bar{x_1}.\bar{x_2}.x_1'.\bar{x_2'}$
$s_2 \to s_2$	$\bar{x_1}.\bar{x_2}.\bar{x_1'}.\bar{x_2'}$
$s_3 \to s_0$	$\bar{x_1}.x_2.x_1'.x_2'$
$s_3 \to s_3$	$\bar{x_1}.x_2.\bar{x_1'}.x_2'$

The truth table is

x_1	x_1'	x_2	x_2'	$\rightarrow$
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Draw a (reduced) OBDD for the transition function.



BDD software

- There are many libraries that implement OBDD
- `https://github.com/johnyf/tool_lists/blob/master/bdd.md`
- Most popular:
- CUDD (exists for several programming languages)
- you can also implement your own...
- A web interface: `http://formal.cs.utah.edu:8080/pbl/BDD.php`

References

[HR04] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and reasoning about systems*. CUP, 2004.