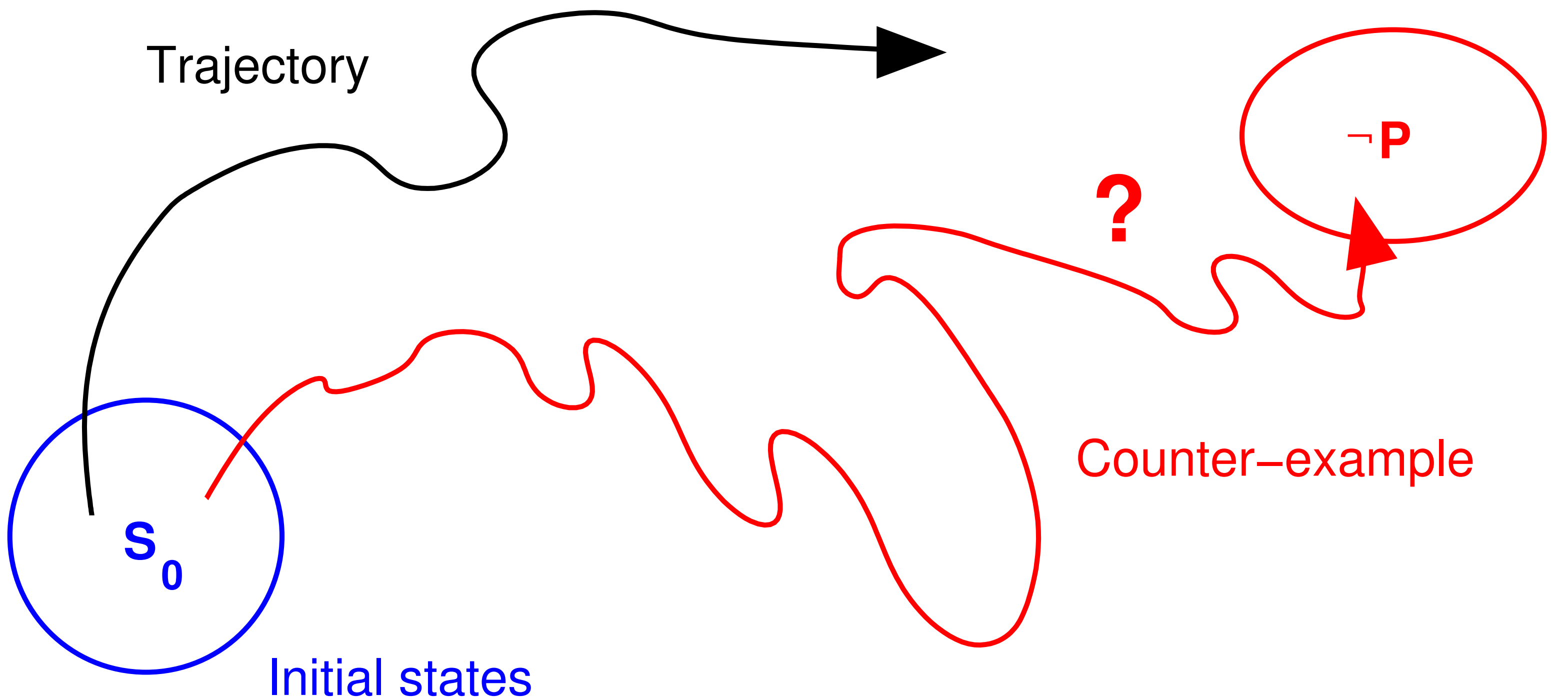
BDDs, Fairness, Validation

Cyrille Artho
KTH Royal Institute of Technology, Stockholm, Sweden
School of Electrical Engineering and Computer Science
Theoretical Computer Science
artho@kth.se

Outline

1. BDDs and symbolic model checking.
2. Bounded model checking.
3. Fairness.
4. Model validation.

Model Checking = state space search



- How to analyze reachable states against the property?

Explicit-state model checking: State space exploration

```
def explore(states: List[State]) {
  // explore each successor of the current state
  // stop search at target
  for (s <- states) {
    if (s.isAccepting) {
      return Found
    }
    if (!visitedStates.contains(s)) {
      currentStates += s
    }
    visitedStates += s
  }
}
```

What if model is too big to fit in memory?

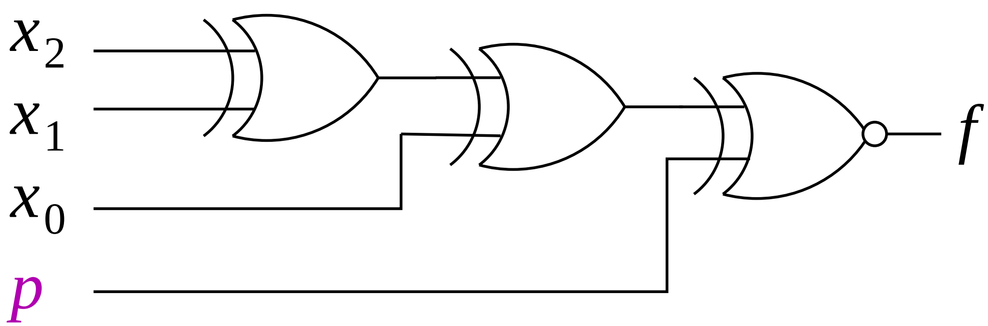
Binary Decision Diagrams (BDDs)

- Published by Randal Bryant in 1986.
- Compressed representation of Boolean functions.
- Eliminates redundancy in sub-expressions.
- For equal variable order, reduction is **canonical** (same result regardless of order in which subexpressions are added).
- Boolean **operations** can be implemented on **compressed** data (ROBDDs).

Example: Parity check for three-bit memory

1. Register $x_2x_1x_0$, Parity p

2. Formula f shows if parity is correct.



Example and illustrations by Armin Biere/JKU Austria.

$x_2 x_1 x_0\ p$	f
0 0 0 0	1
0 0 0 1	0
0 0 1 0	0
0 0 1 1	1
0 1 0 0	0
0 1 0 1	1
0 1 1 0	1
0 1 1 1	0
1 0 0 0	0
1 0 0 1	1
1 0 1 0	1
1 0 1 1	0
1 1 0 0	1
1 1 0 1	0
1 1 1 0	0
1 1 1 1	1

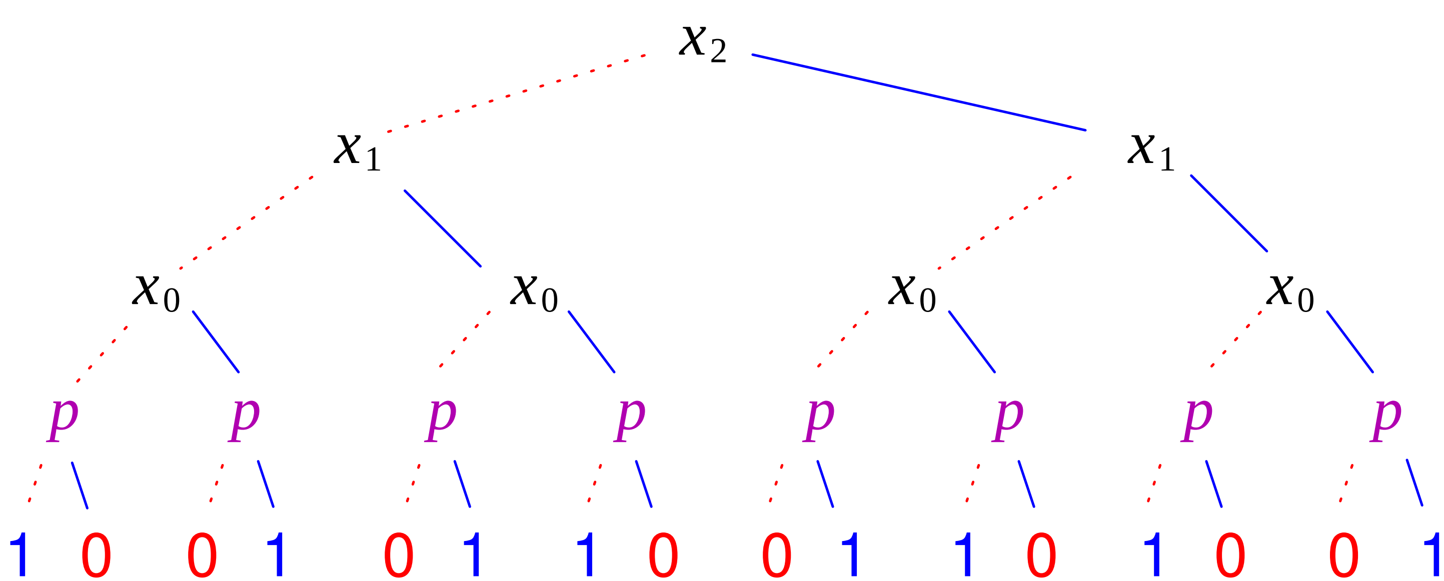
Transposed table

x_2	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
x_1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
x_0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
p	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
f	1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1

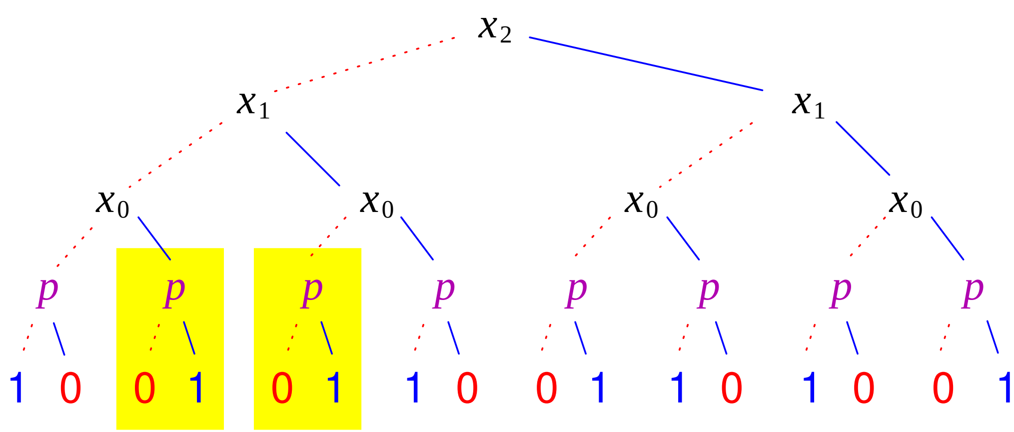
$\bar{x}_2$								x_2							
$\bar{x}_1$				x_1				$\bar{x}_1$				x_1			
$\bar{x}_0$		x_0		$\bar{x}_0$		x_0		$\bar{x}_0$		x_0		$\bar{x}_0$		x_0	
$\bar{p}$	p	$\bar{p}$	p	$\bar{p}$	p	$\bar{p}$	p	$\bar{p}$	p	$\bar{p}$	p	$\bar{p}$	p	$\bar{p}$	p
1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1

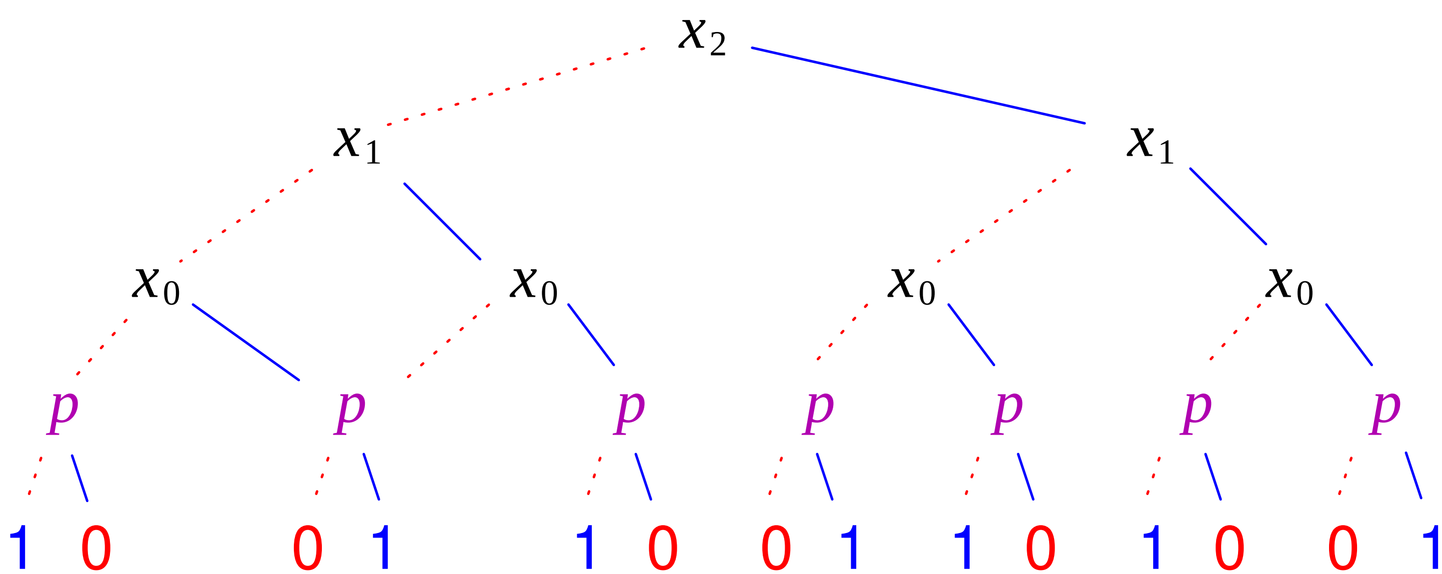
Graphical representation

$\bar{x}_2$								x_2							
$\bar{x}_1$				x_1				$\bar{x}_1$				x_1			
$\bar{x}_0$		x_0		$\bar{x}_0$		x_0		$\bar{x}_0$		x_0		$\bar{x}_0$		x_0	
$\bar{p}$	p	$\bar{p}$	p	$\bar{p}$	p	$\bar{p}$	p	$\bar{p}$	p	$\bar{p}$	p	$\bar{p}$	p	$\bar{p}$	p
1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1

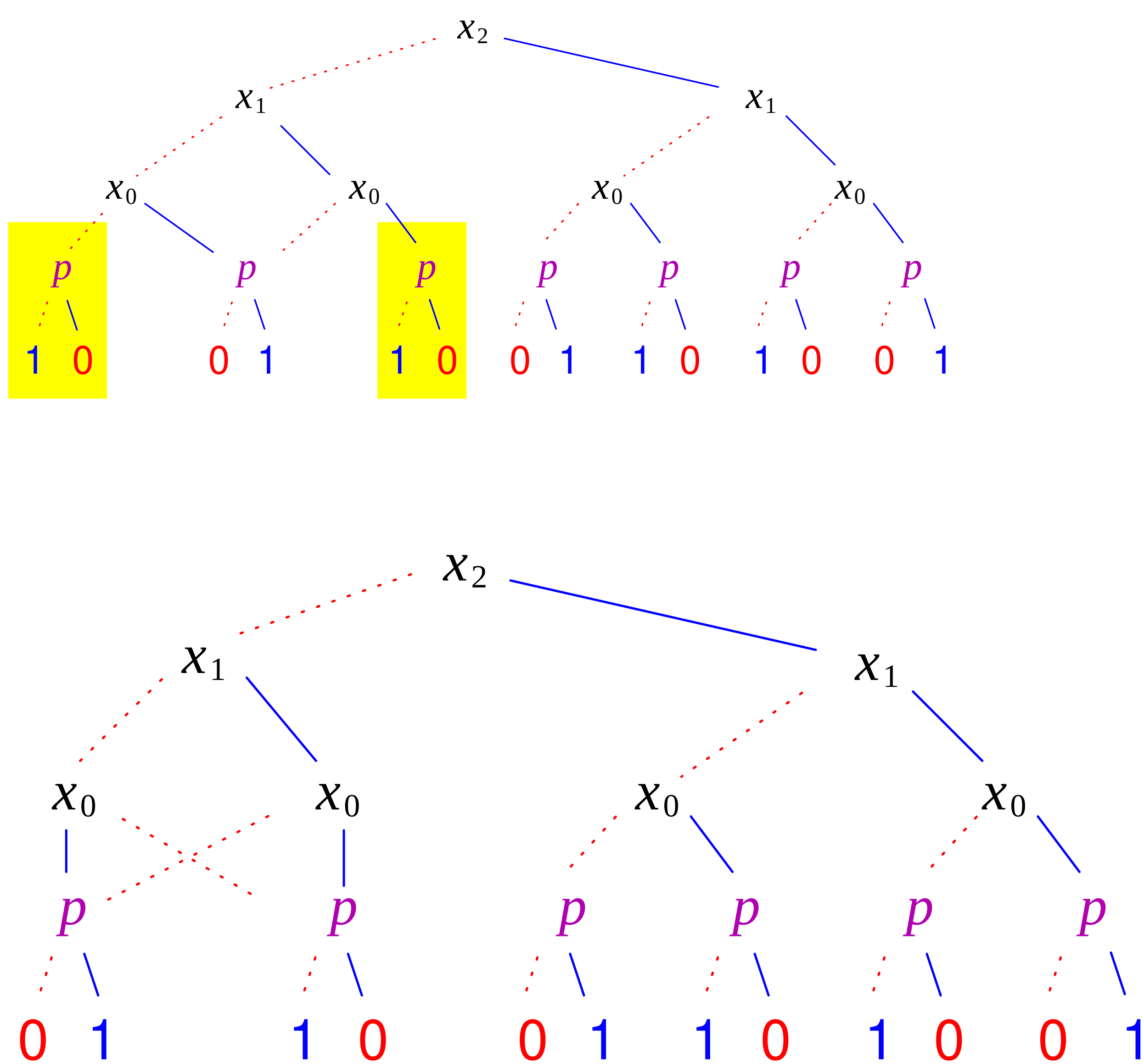


Combining identical subtrees

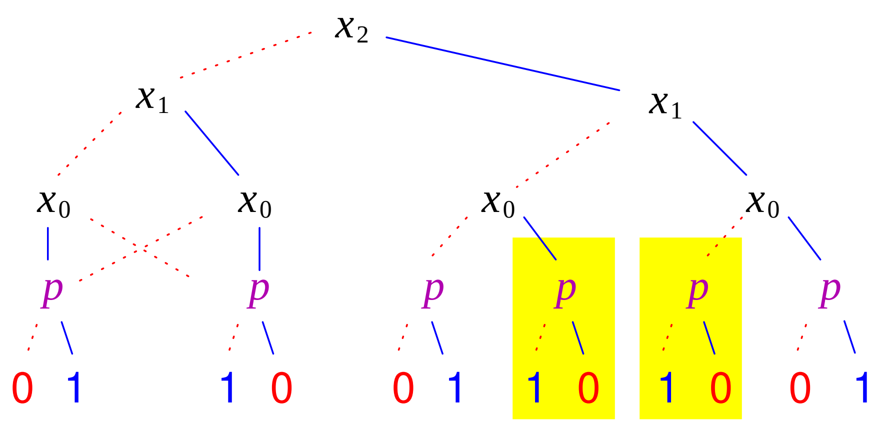


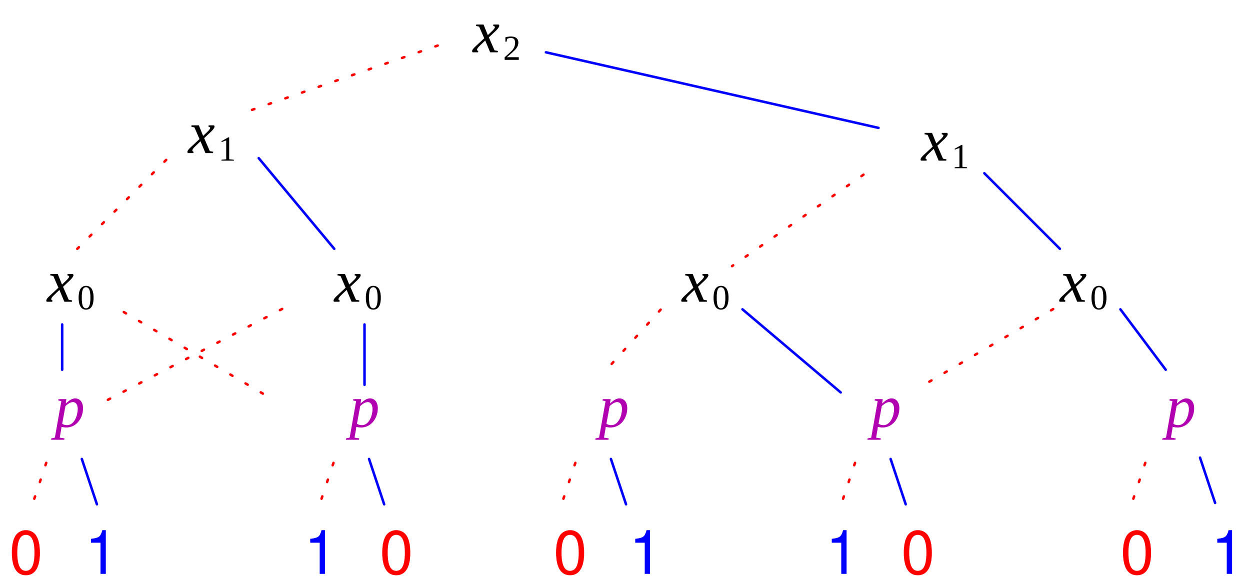


Combining identical subtrees — 2

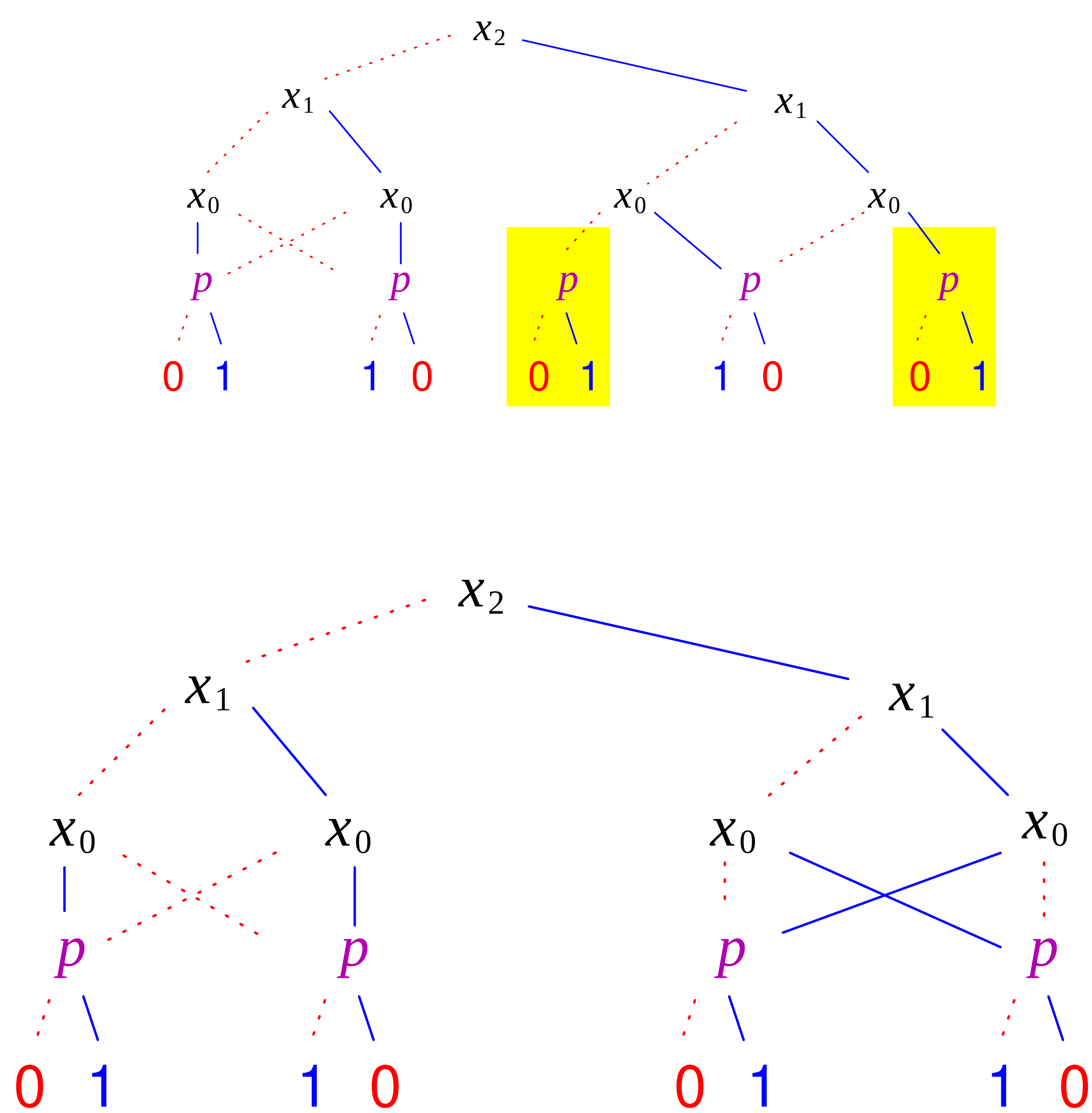


Combining identical subtrees — 3

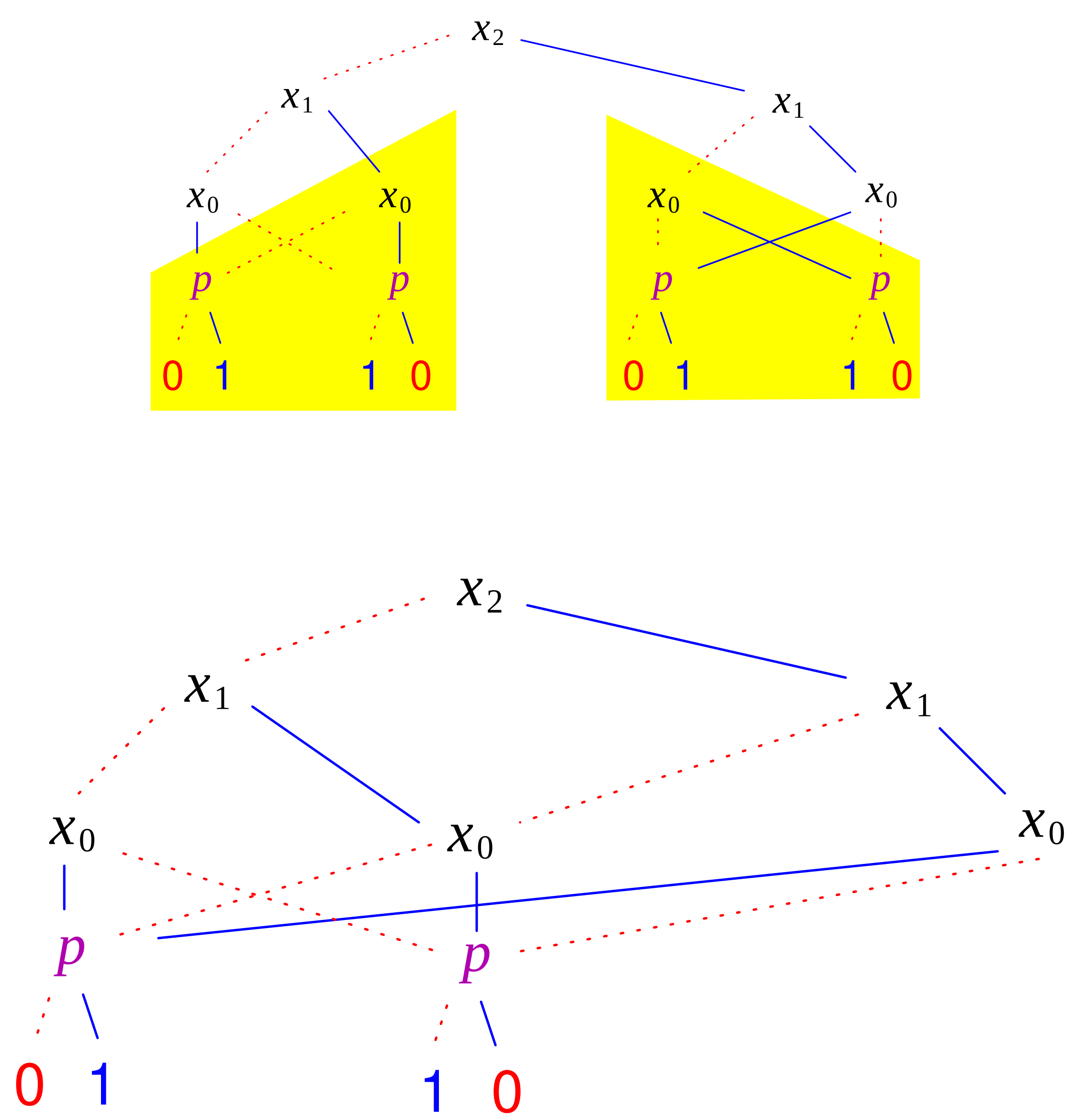




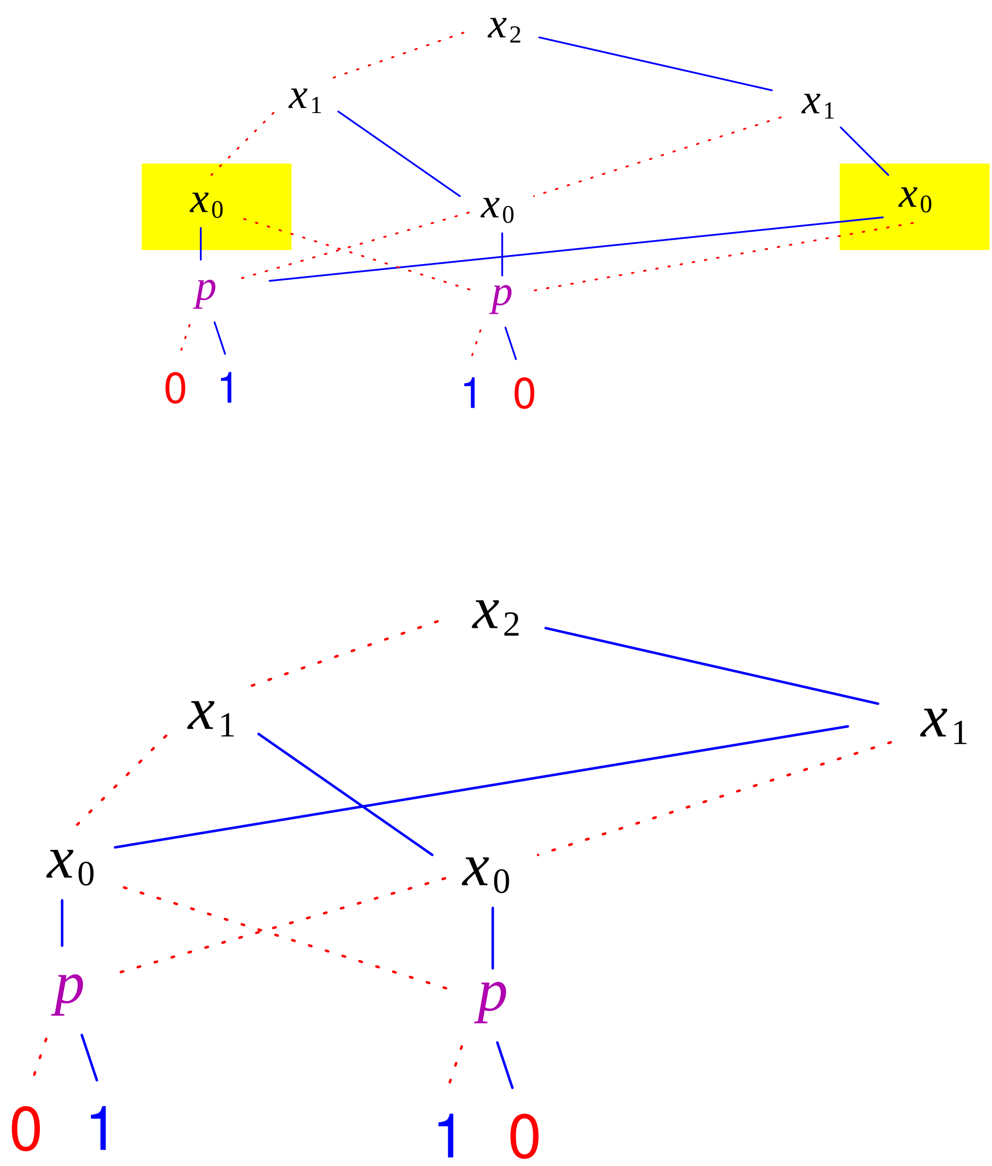
Combining identical subtrees — 4



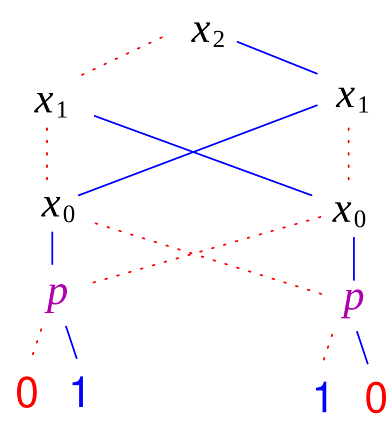
Combining identical subtrees — 5

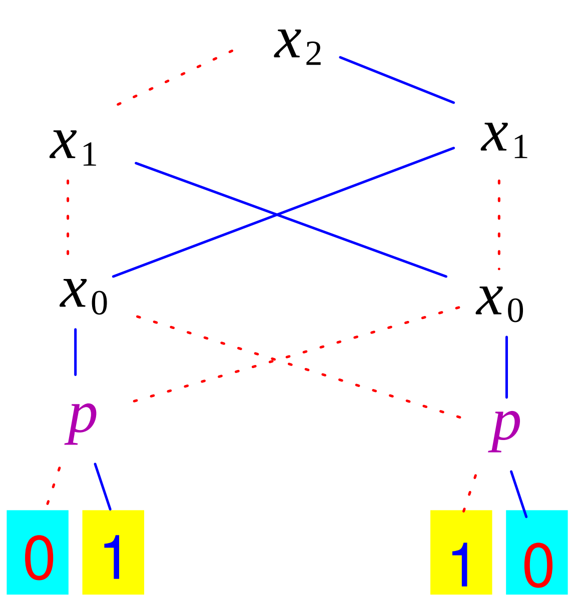


Combining identical subtrees — 6

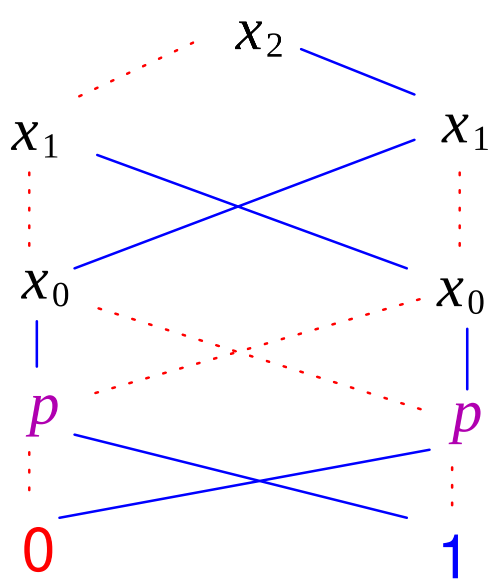


Combining identical subtrees — 7





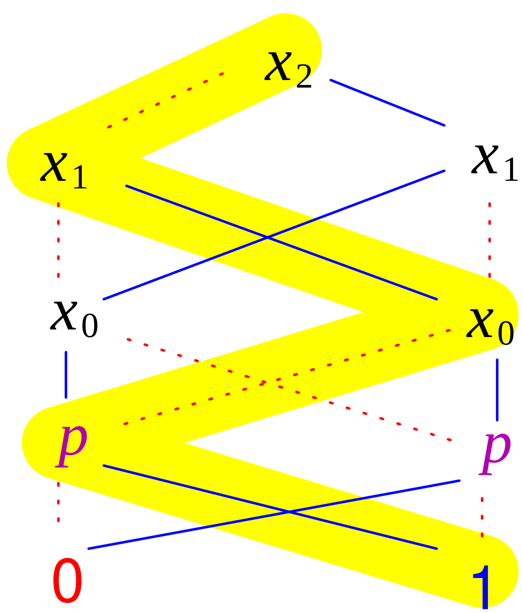
Final result: A Reduced, Ordered Binary Decision Diagram (ROBDD)



- All redundant subtrees are shared.
- Memory-efficient data structure.
- Usually „BDD” is used to denote ROBDD.

How to read a BDD

x_2	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
x_1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
x_0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
p	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1



Binary Decision Diagrams: Definition

- Given: Set of variables $V = \{x_0, x_1, x_2, \ldots\}$.
- BDD on V is a directed, acyclic graph (DAG)

$$G = (N, \rightarrow, \ell)$$

- DAG contains set of nodes N,
  one root $r \in N$ (only nodes without predecessor),
  one or two sinks in N (only nodes without successor), written $0, 1$;
- edges $\rightarrow \subseteq N \times N$, $\rightarrow^+$ is acyclic;
  exactly two successors n_0 and n_1 for all $n \in N \setminus \{0, 1\}$;
- labels $\ell : N \setminus \{0, 1\} \rightarrow V$ of nodes with variables ($\ell() = 0, \ell(1) = 1$).

Semantics of BDDs

$$\boxed{[\![\cdot]\!] : N \to \mathrm{BooleExpr}}$$

Projection $[\![\cdot]\!]$ of nodes N on Boolean expressions/functions over V.

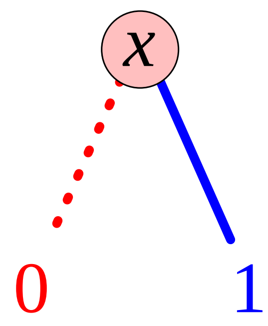
$$[\![n]\!] = \bar{x}_i \wedge [\![n_0]\!] \vee x_i \wedge [\![n_1]\!]$$

$$[\![0]\!] = 0$$

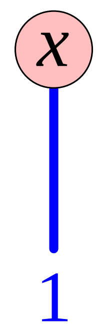
$$[\![1]\!] = 1$$

for $n = \mathrm{node}(\mathrm{x_i}, \mathrm{n_1}, \mathrm{n_0})$

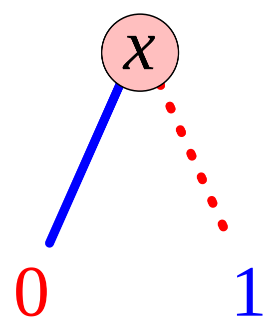
Reading BDDs



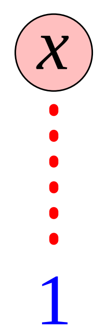
or



x



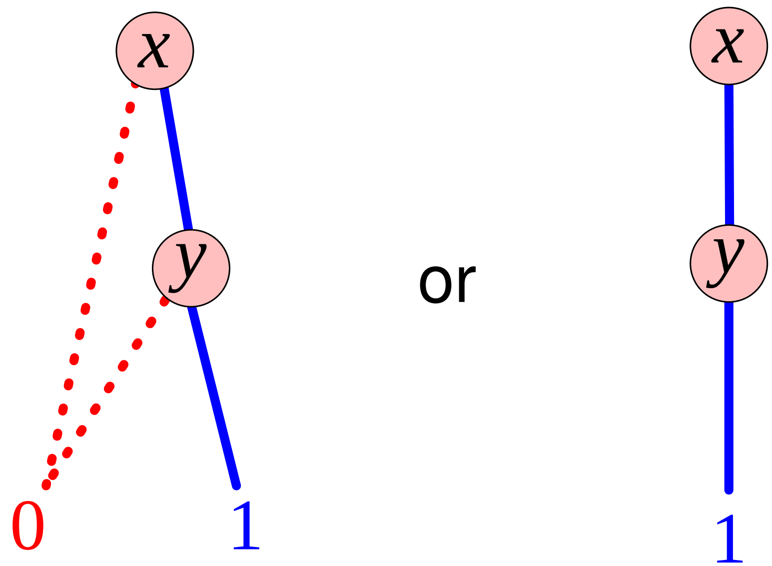
or



$\bar{x}$

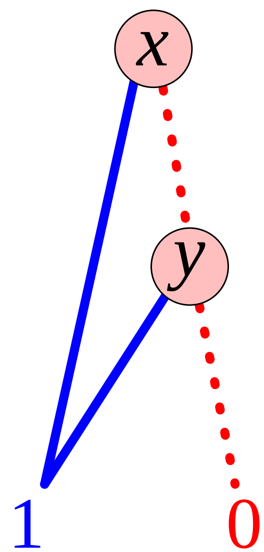
Examples and illustrations by Armin Biere/JKU Austria.

More examples

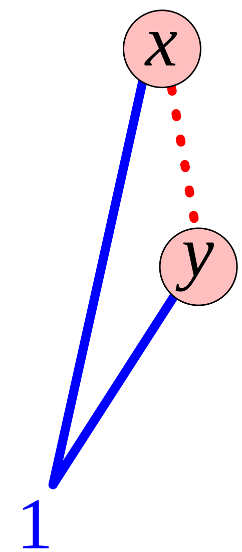


What does this BDD express?

More examples — 2

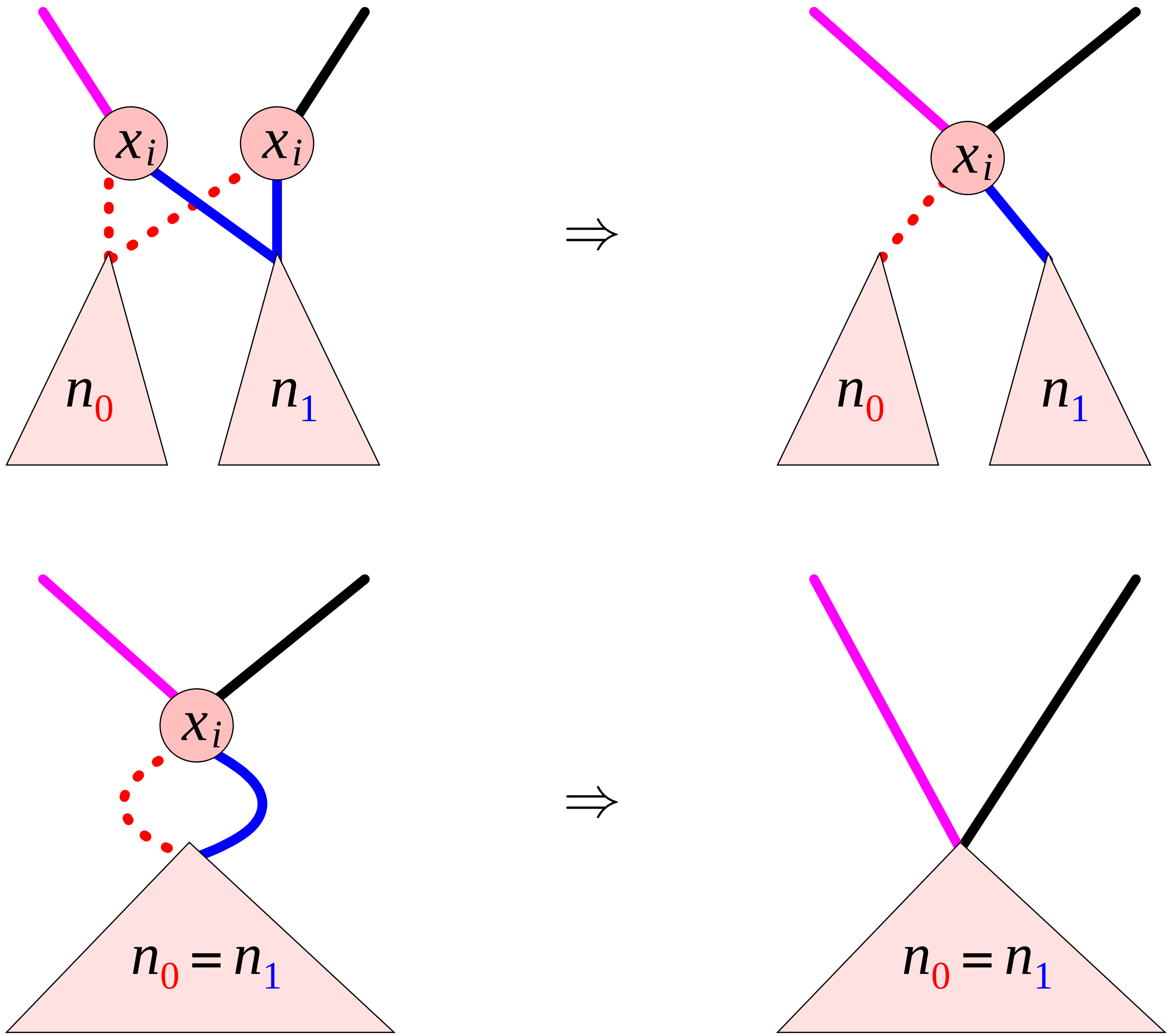


or



What does this BDD express?

Reduction



Reduction algorithm

1. Traverse BDD from bottom to top

2. For each layer of variables,

   (a) eliminate nodes with two equal successors,
   (b) sort nodes lexicographically,
   (c) unite nodes with pairwise equal successors.

   Alternative to (b): hashing.

Hashing: nodes are unified as they are created!

ROBDDs [Bryant86]

Assumption 1: Variables $V \cup \{0,1\}$ are (linearly) ordered:

$$0 < 1 < x_0 < x_1 < x_2 < \dots$$

Induced order on nodes N:

$$n < n' \text{ iff (if and only if) } \ell(n) < \ell(n')$$

Assumption 2: Edge relation is compatible with order:

$$n > n_0 \text{ and } n > n_1$$

Each function has exactly one BDD!

(for a given variable order)

Usage of BDDs in model checking

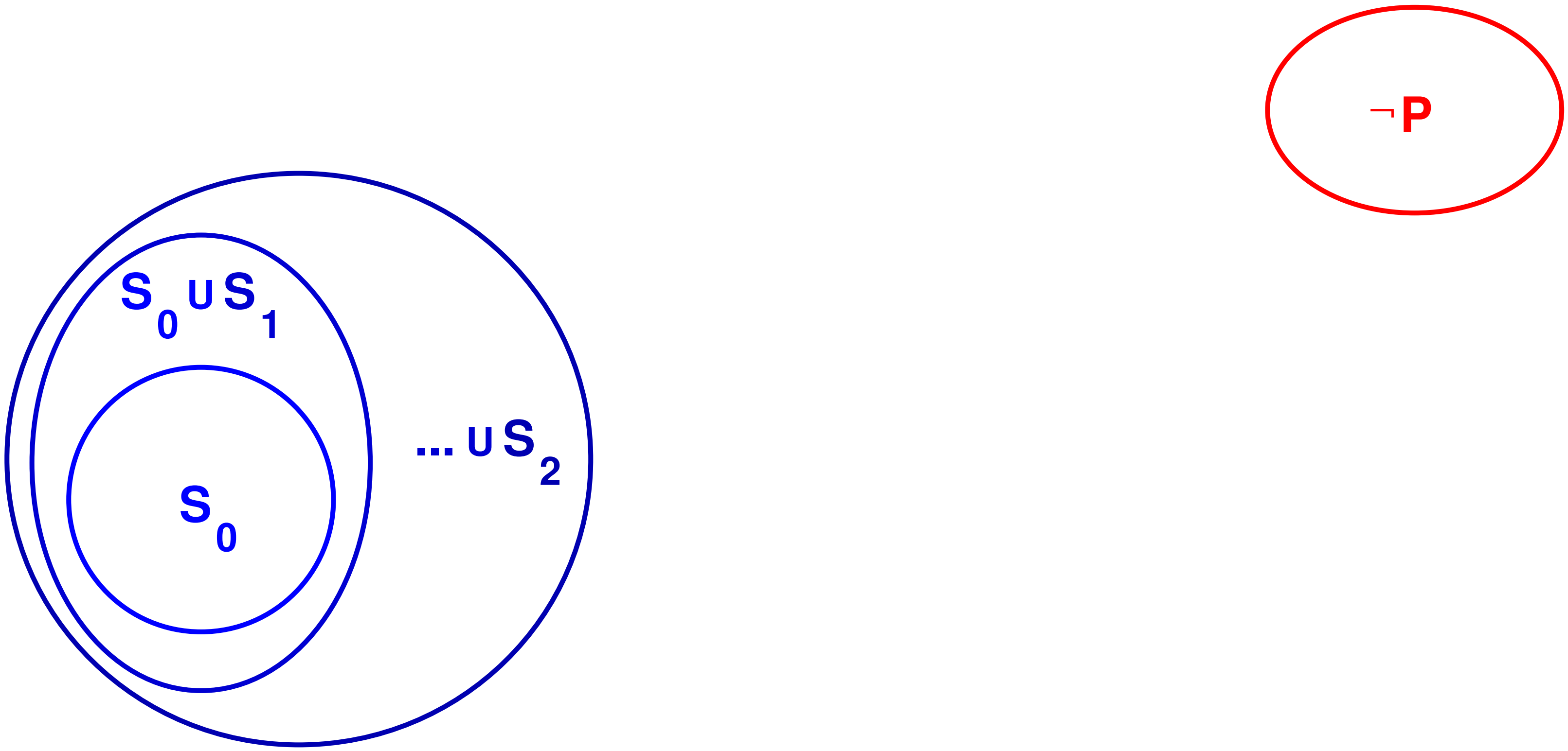
- Equivalence checking.
- Efficient representation of formulas, sets.
- „Good” variable order crucial for small size and performance.

$\rightarrow$ Heuristics determine variable order.
$\rightarrow$ Computations on BDDs use hashing, caching to avoid redundant computations.
$\rightarrow$ Some BDDs (e. g., multiplication) always exponentially large.

- Free efficient implementations (e. g.„ CUDD).

Key data structure for symbolic model checking.

Symbolic model checking



$$New := \mathrm{Image}(Old) \cup Old$$

Basic algorithm

S = states; R = transition relation; I= initial states

$\mathrm{CalculateReachable}(S,R,I)$

$New := I$

do

$Old := New$

$New := \mathrm{Image}(S,R,Old) \cup Old$

while $Old \neq New$

return New

$\mathrm{Image}(S,R,Old)$

return $\{t \in S | \exists s \in Old \text{ with } R(s,t)\}$

Representing sets and formulas as binary vectors (BDDs)

1. Encode systems as Booleans:
   encode S in 2^n, interpret states $s \in S$ as n-ary Boolean vectors.

2. Interpret initial states I as characteristic function $I : 2^n \to \{0, 1\}$.

3. Interpret transitions R as characteristic function $R : 2^n \times 2^n \to \{0, 1\}$.

4. Represent Boolean functions as BDDs.

5. Use BDD operations instead of set operations.

Inventors: McMillan and Coudert/Madre.

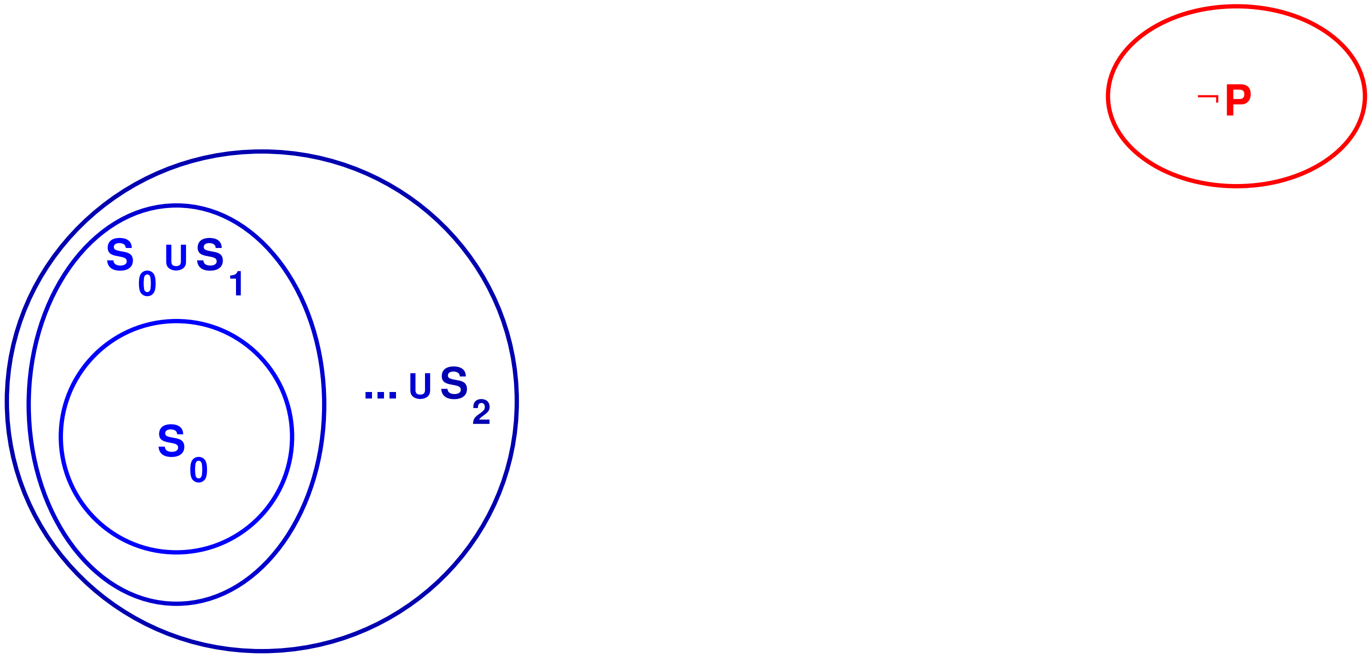
Quizzes

1. BDDs: Link to interactive web page where you can practice.
2. Temporal logics: Three examples for practice.

- Quizzes are not graded but highly recommended as exam preparation.

Bounded model checking

- Symbolic model checking is efficient if formula can be compressed.
- Good variable order is not always possible.
- Another approach: Expand the state transition system for k steps:



Use solver to check system states after k steps.

SAT solving

- Satisfiability (SAT) problem is NP-complete.
- SAT solvers are efficient solvers for logical formulas.
- Intelligent pre-processing and state space search take advantage of structure in formula.
- Does not work against worst case, but real formulas have internal structure.
- Expanded state transition system can be expressed in propositional logic and solved by SAT solver.

Bounded model checking in NuSMV

```
MODULE main -- from the NuSMV tutorial
VAR
  y : 0..15;
ASSIGN
  init(y) := 0;
TRANS
case
  y=7: next(y)=0;
  TRUE : next(y) = ((y + 1) mod 16);
esac
LTLSPEC G (y=4 -> X y=6)
```

Running NuSMV in BMC mode

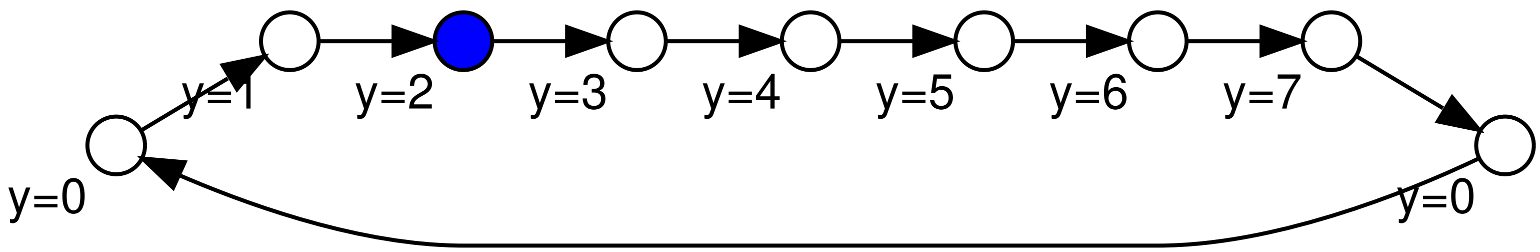
```
NuSMV -bmc bmc_tutorial.smv
-- no counterexample found with bound 0
-- no counterexample found with bound 1
-- no counterexample found with bound 2
-- no counterexample found with bound 3
-- no counterexample found with bound 4
-- specification  G (y = 4 ->  X y = 6) is false
-- as demonstrated by the following execution sequence
Trace Description: BMC Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    y = 0
  -> State: 1.2 <-
    y = 1
...
  -> State: 1.6 <-
    y = 5
```

Liveness properties with BMC

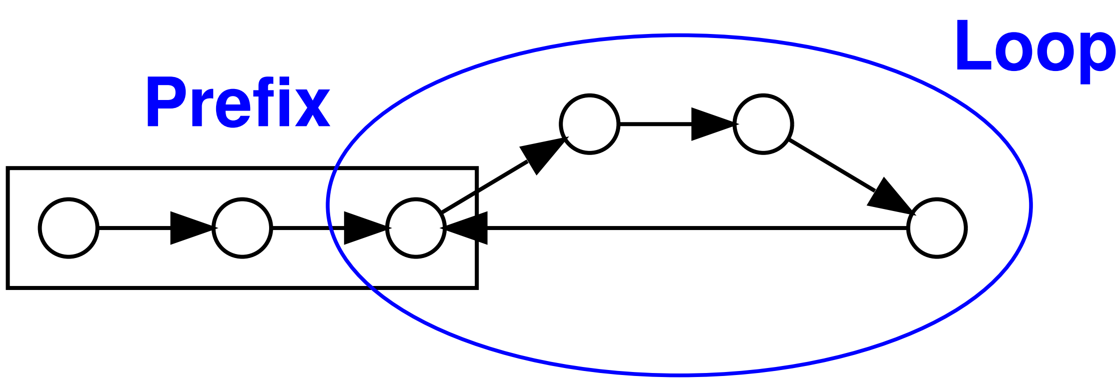
```
LTLSPEC !G F(y = 2)
NuSMV -bmc bmc_tutorial.smv
-- no counterexample found with bound 0
...
-- no counterexample found with bound 7
-- specification !( G ( F y = 2)) is false
-- as demonstrated by the following execution sequence
Trace Description: BMC Counterexample
Trace Type: Counterexample
-- Loop starts here
-> State: 1.1 <-
y = 0
-> State: 1.2 <-
y = 1
...
-> State: 1.9 <-
y = 0
```

Counterexamples for liveness properties

Previous example:



General case:



How large does the bound k have to be for BMC?

Fairness

Fairness constraints have to hold infinitely often during a valid trace:

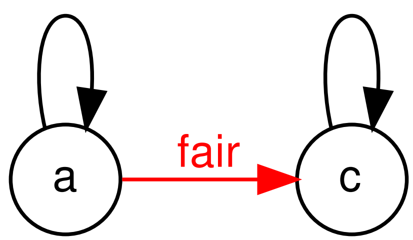
```
FAIRNESS button_pressed;
```

Straightforward for booleans, but not for temporal formulas.

Weak (Büchi) fairness: $\Box\Diamond(\mathit{fairness} \rightarrow \mathit{property})$

A set of transitions cannot be enabled **forever** without being taken.

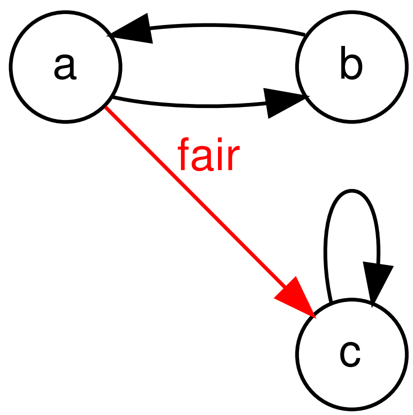
`JUSTICE` or **`FAIRNESS`** in NuSMV (same functionality).



Strong (Streett) fairness: $(\Box\Diamond fairness) \rightarrow (\Box\Diamond property)$

Set of transitions cannot be enabled **infinitely often** without being taken.

`COMPASSION(p, q)` in NuSMV.



Fairness?

NuSMV: „Fair states set is empty”

```
********* WARNING *********
Fair states set of the finite state machine is empty.
This might make results of model checking not trustable.
********* END WARNING *********
```

Explanation: You have a fairness expression that is never true.

- The desired fair state(s) are never reached!
- Usually the result of a deadlock in your FSM.
- You cannot trust the result until you fix this warning.

Fairness properties

- Elevator will eventually reach 12^{th} floor.
- Button will eventually be pressed.

What is the consequence of each property when used as a fairness property?

Fairness

You may have to specify both the *presence* as well as the *absence* of an action as fairness properties:

```
FAIRNESS(request)
FAIRNESS(!request)
```

What does that mean?

- The same as `(request && !request)`?
- Both cases appear equally often?

Wrong use of fairness

With great power comes great responsibility.

Do not use fairness conditions that

- relate to internal states or outputs;
- encode a safety property.

If you ignore all possibilities of a controller doing the wrong thing, the model checker will find correct solutions, but. . .

Wrong use of fairness

Photo by MichaelMaggs

With great power...

Fairness condition ensures occurrence of any event(s)

Possibility of using desired **end state in fairness property**

Traces that violate liveness property are ignored!

Model validation

Model | **Real system**

Is the model adequate?

Correct but flawed models

Vacuity: $a \to b$ holds, but a is never true.
Entire property holds for the wrong reason („antecedent failure”)!

Solution: Check that a holds in at least some states.

Zeno-timelocks: Model executes infinitely fast.
No way for real system to fulfill property!

Solution: Use simulation mode to get example traces, study how model reacts.

Model validation

1. Does each formula (or subformula) have a temporal operator?

   Without a temporal operator, a formula only applies to the initial state,

   or the current state in a compound formula where you reason about a subtrace.

   This may not be the intended meaning.

2. If you have a negation, be very careful about where you place it.
   !G is not the same as G!, so the negation has to be placed on the correct side of any temporal operator. (A formula may not hold globally, i.e., never, or globally, the negation of the formula may hold.)
   Both formulas can make sense, but they mean different things.
   Same for all other temporal operators or path quantifiers.

Model validation – 2

Other techniques to validate your model:

1. Check the reachability of desired states.
   Use simulation mode or define additional liveness properties.

2. Negate the specification.
   Now there should be paths that violate the desired property.
   Study the result and see if it makes sense.

3. Add additional simple properties as sanity checks.

From: Artho, Hayamizu, Ramler, Yamagata: With an Open Mind: How to Write Good Models.

Use fairness safely!

Fairness should relate to the
behavior of agents using the system, so there is

- no denial of service,
- infinite idling, or
- abuse of the system.

Examples:

- A button is eventually pressed.
- A system mode is eventually changed.
- A process eventually leaves a critical section.

Simulating models with NuSMV

Key simulation commands (when using **`NuSMV -int`**):

`NuSMV > go`	Load and prepare model
`NuSMV > pick_state -i`	Pick initial state interactively
`NuSMV > simulate -p -i`	Simulate (interactively), print progress; control-C exits
`NuSMV > show_trace -v`	Show trace
`NuSMV > quit`	Exit

Back to vending machine

```
-> State: 1.1 <-
  choice = TRUE
  payment = TRUE
  acc_payment = FALSE
  state = ready
  release_item = FALSE
-> State: 1.2 <-
  state = expect_payment
-> State: 1.3 <-
  acc_payment = TRUE
-> State: 1.4 <-
  state = dispense_item
  release_item = TRUE
-> State: 1.5 <-
  acc_payment = FALSE
  state = ready
  release_item = FALSE
-> State: 1.6 <-
  state = expect_payment
-> State: 1.7 <-
  acc_payment = TRUE
-> State: 1.8 <-
  state = dispense_item
  release_item = TRUE
```

Summary

Model checking

Symbolic checking.

- Everything is a bit vector.
- Efficient representation: BDDs.

Bounded model checking.

- Expand k trans. $\rightarrow$ 0-order formula.
- Efficient solution: SAT solver.

Model validation

- Negate properties.
- Check if antecedent (a in $a \rightarrow b$) is ever true.
- Ensure „interesting" states are actually reachable.