Computer-Aided Verification

CS745/ECE725

Borzoo Bonakdarpour

University of Waterloo

(Fall 2013)

CTL Model Checking

Agenda

- Computation Tree Logic (CTL)
- CTL Model Checking
- Binary Decision Diagrams (BDDs)
- The Model Checker SMV

Agenda

- *Computation Tree Logic (CTL)*
- CTL Model Checking
- Binary Decision Diagrams (BDDs)
- The Model Checker SMV

CTL

- Computation Tree Logic: Intuitions.
- CTL: Syntax and Semantics.
- CTL in Computer Science.
- CTL and Model Checking: Examples.
- CTL Vs. LTL.

Intuition

LTL implicitly quantifies universally over paths:

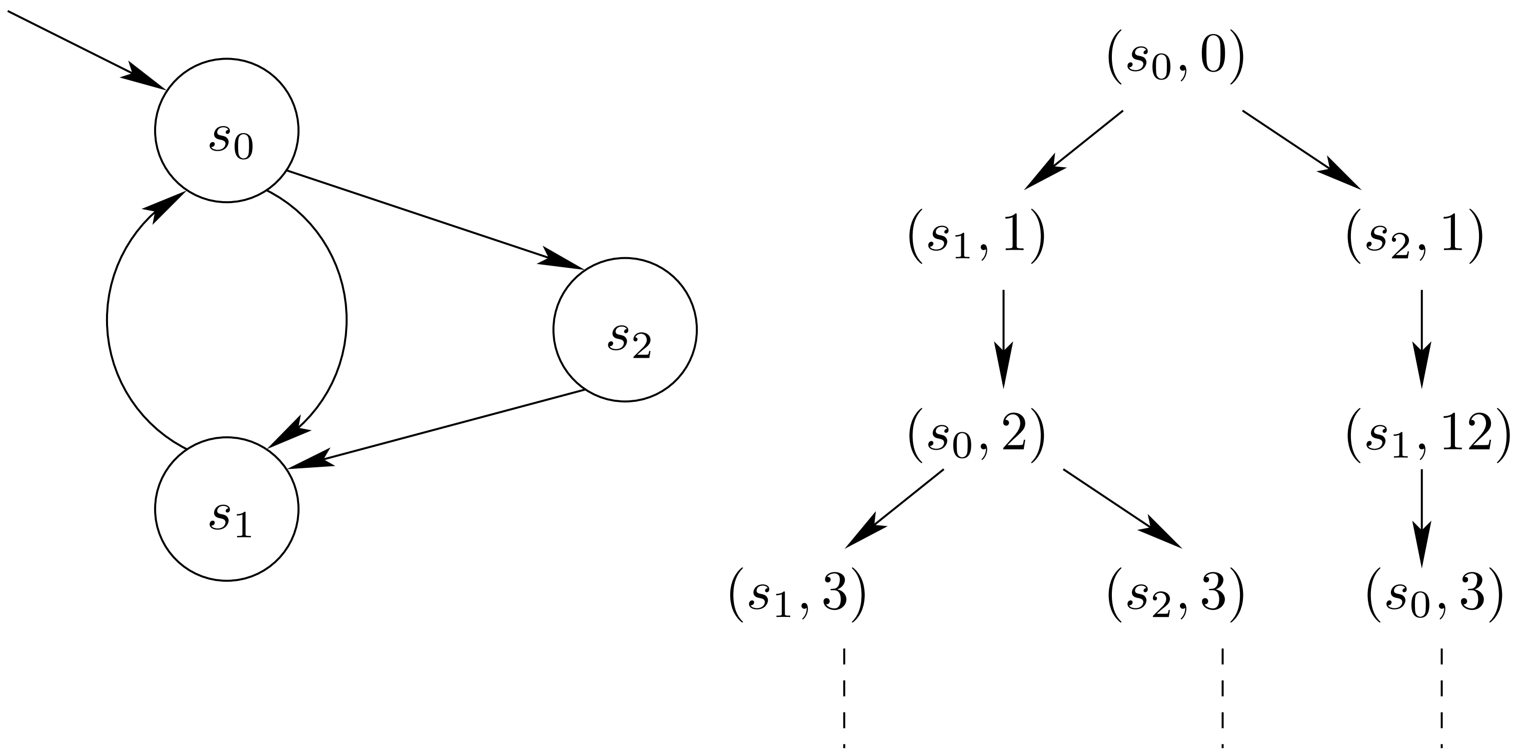
$\langle M, s\rangle \models \phi$ iff for every path π starting at s, $\langle M, \pi\rangle \models \phi$

Properties that assert the *existence* of a path cannot be expressed in LTL. In particular, properties which mix existential and universal path quantifiers cannot be expressed.

The Computation Tree Logic, CTL, solves these problems:

- CTL explicitly introduces path quantifiers!
- CTL is the natural temporal logic interpreted over Branching Time Structures.

Intuition



Intuition

CTL is evaluated over branching-time structures (Trees). CTL explicitly introduces *path quantifiers*:

- All Paths: $\mathbf{A}$
- Exists a Path: $\mathbf{E}$

Every temporal operator $(\Box, \Diamond, \bigcirc, \mathrm{U})$ is preceded by a path quantifier ($\mathbf{A}$ or $\mathbf{E}$).

In universal modalities: $(\mathbf{A}\Box, \mathbf{A}\Diamond, \mathbf{A}\bigcirc, \mathbf{A}\mathrm{U})$, the temporal formula is true in *all* the paths starting in the current state.

In existential modalities: $(\mathbf{E}\Box, \mathbf{E}\Diamond, \mathbf{E}\bigcirc, \mathbf{E}\mathrm{U})$, The temporal formula is true in *some* path starting in the current state.

Intuition

Countable set Σ of atomic propositions: $p, q, \cdots$ the set FORM of formulas is:

$$\begin{aligned} \phi, \psi \ \rightarrow p \ | \ \top \ | \ \bot \ | \ \neg\phi \ | \ \phi \wedge \psi \ | \ \phi \vee \psi \ | \\ \mathbf{A}\Box\phi \ | \ \mathbf{A}\Diamond\phi \ | \ \mathbf{A} \bigcirc \phi \ | \ \mathbf{A}\phi\mathsf{U}\psi | \\ \mathbf{E}\Box\phi \ | \ \mathbf{E}\Diamond\phi \ | \ \mathbf{E} \bigcirc \phi \ | \ \mathbf{E}\phi\mathsf{U}\psi | \end{aligned}$$

CTL Semantics

We interpret our CTL temporal formulae over Kripke models linearized as trees.

Let Σ be a set of atomic propositions. We interpret our CTL temporal formulae over Kripke Models:

$$M = \langle S, I, R, \Sigma, L \rangle$$

The semantics of a temporal formula is provided by the satisfaction relation:

$$\models : \langle M \times S \times \mathrm{FORM} \rangle \rightarrow \{true, false\}$$

CTL Semantics

We start by defining when an atomic proposition is true at a state/time "s_i"

$$M, s_i \models p \quad \text{iff} \quad p \in L(s_i) \qquad \text{(for } p \in \Sigma\text{)}$$

The semantics for the classical operators is as expected:

$$M, s_i \models \neg\phi \qquad \text{iff} \quad s_i \not\models \phi$$

$$M, s_i \models \phi \wedge \psi \qquad \text{iff} \quad s_i \models \phi \wedge s_i \models \psi$$

$$M, s_i \models \phi \vee \psi \qquad \text{iff} \quad s_i \models \phi \vee s_i \models \psi$$

$$M, s_i \models \top$$

$$M, s_i \not\models \bot$$

CTL Semantics

We start by defining when an atomic proposition is true at a state/time "s_i"

$$M, s_i \models p \quad \text{iff} \quad p \in L(s_i) \qquad \text{(for } p \in \Sigma\text{)}$$

The semantics for the classical operators is as expected:

$$M, s_i \models \mathbf{A} \bigcirc \phi \qquad \text{iff} \quad \forall \pi = (s_i, s_{i+1}, \cdots) \bullet M, s_{i+1} \models \phi$$

$$M, s_i \models \mathbf{E} \bigcirc \phi \qquad \text{iff} \quad \exists \pi = (s_i, s_{i+1}, \cdots) \bullet M, s_{i+1} \models \phi$$

$$M, s_i \models \mathbf{A} \square \phi \qquad \text{iff} \quad \forall \pi = (s_i, s_{i+1}, \cdots) \bullet \forall j \geq i \bullet M, s_j \models \phi$$

$$M, s_i \models \mathbf{E} \square \phi \qquad \text{iff} \quad \exists \pi = (s_i, s_{i+1}, \cdots) \bullet \forall j \geq i \bullet M, s_j \models \phi$$

CTL Semantics

The semantics for the classical operators is as expected:

$$M, s_i \models \mathbf{A}\Diamond\phi \quad \text{iff} \quad \forall\pi = (s_i, s_{i+1}, \cdots) \bullet \exists j \geq i \bullet M, s_j \models \phi$$

$$M, s_i \models \mathbf{E}\Diamond\phi \quad \text{iff} \quad \exists\pi = (s_i, s_{i+1}, \cdots) \bullet \exists j \geq i \bullet M, s_j \models \phi$$

$$M, s_i \models \mathbf{A}\phi\mathsf{U}\psi \quad \text{iff} \quad \forall\pi = (s_i, s_{i+1}, \cdots) \bullet \exists j \geq i \bullet M, s_j \models \psi \wedge \forall i \leq k \leq j \bullet M, s_k \models \phi$$

$$M, s_i \models \mathbf{E}\phi\mathsf{U}\psi \quad \text{iff} \quad \exists\pi = (s_i, s_{i+1}, \cdots) \bullet \exists j \geq i \bullet M, s_j \models \psi \wedge \forall i \leq k \leq j \bullet M, s_k \models \phi$$

CTL Semantics

CTL is given by the standard Boolean logic enhanced with temporal operators.

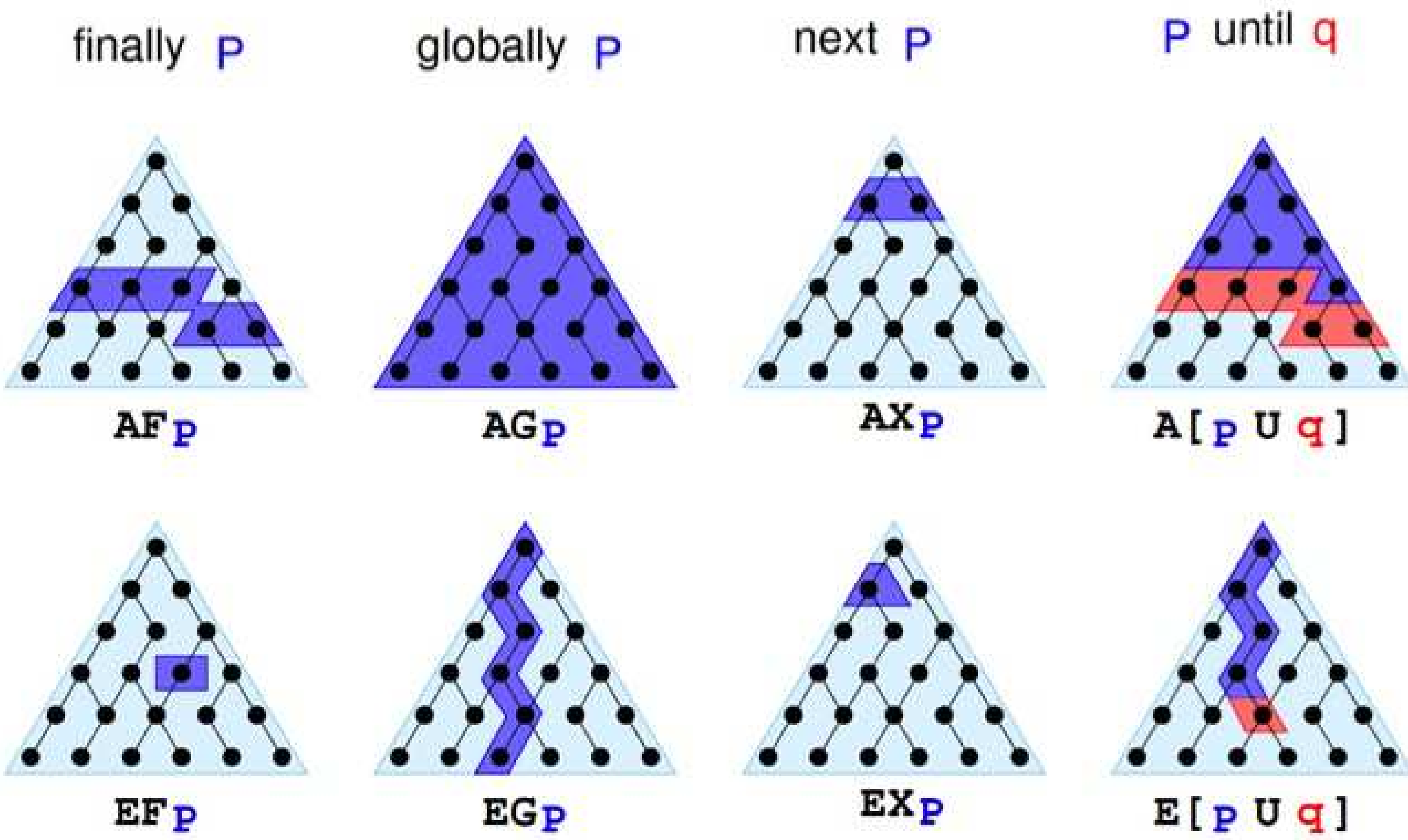
Necessarily Next. $\mathbf{A} \bigcirc \phi$ is true in s_t iff ϕ is true in every successor state s_{t+1}.

Possibly Next. $\mathbf{E} \bigcirc \phi$ is true in s_t iff ϕ is true in one successor state s_{t+1}.

Necessarily in the future (or "Inevitably"). $\mathbf{A}\Diamond\phi$ is true in s_t Iff ϕ is inevitably true in some $s_{t'}$ with $t' \geq t$.

Possibly in the future (or "Possibly"). $\mathbf{E}\Diamond\phi$ is true in s_t iff ϕ may be true in some $s_{t'}$ with $t' \geq t$.

CTL Semantics



Safety Properties

Safety:

"something bad will not happen"

Typical examples:

$$\mathbf{A}\Box\neg(\text{reactor\_temp} > 1000)$$

Safety properties are usually of the form:

$$\mathbf{A}\Box\neg\cdots$$

Liveness Properties

Liveness:

"something good will happen"

Typical examples:

- $\mathbf{A}\Diamond \mathrm{rich}$
- $\mathbf{A}\Diamond(x > 5)$
- $\mathbf{A}\Box(\mathrm{start} \Rightarrow \mathbf{A}\Diamond \mathrm{terminate})$
  Leads-to, unbounded response

and so on.....

Liveness properties are usually of the form:

$$\mathbf{A}\Diamond\neg\cdots$$

In-class Exercise

Write a CTL formula that is equal to the following LTL formula:

$$\Diamond T \;\Rightarrow\; \Diamond C$$

In-class Exercise

Write a CTL formula that is equal to the following LTL formula:

$$\Diamond T \;\Rightarrow\; \Diamond C$$

What about:

$$\mathbf{A}\Diamond T \;\Rightarrow\; \mathbf{A}\Diamond C$$

LTL vs. CTL

Many CTL formulae cannot be expressed in LTL (e.g., those containing paths quantified existentially)

E.g., $\mathbf{E}\phi$

Many LTL formulae cannot be expressed in CTL

E.g., $\Diamond T \ \Rightarrow \ \Diamond C$ (*Strong Fairness* in LTL)

i.e, formulae that select a *range of paths* with a property

Some formulae can be expressed both in LTL and in CTL (typically LTL formulae with operators of nesting depth 1)

Agenda

- Computation Tree Logic (CTL)
- *CTL Model Checking*
- Binary Decision Diagrams (BDDs)
- The Model Checker SMV

Problem Statement and Assumptions

Problem. Given a model $\mathcal{M}$ and a CTL formula ϕ, determine whether or not $\mathcal{M} \models \phi$.

Assumptions:

- $\mathcal{M}$ is a finite model: finite number of states with variables of finite domain.
- ϕ is a finite length CTL formula.

Solution

1. Transform ϕ into a formula in terms of: $\mathbf{A}\Diamond, \mathbf{E}\mathsf{U}, \mathbf{E}\bigcirc, \wedge, \vee, \perp$.
2. For each subformula φ of ϕ, label states of $\mathcal{M}$, say s, such that $s \models \varphi$.
3. If the initial state s_0 satisfies a subformula φ, then $\mathcal{M} \models \varphi$ as well.

Labelling Algorithm

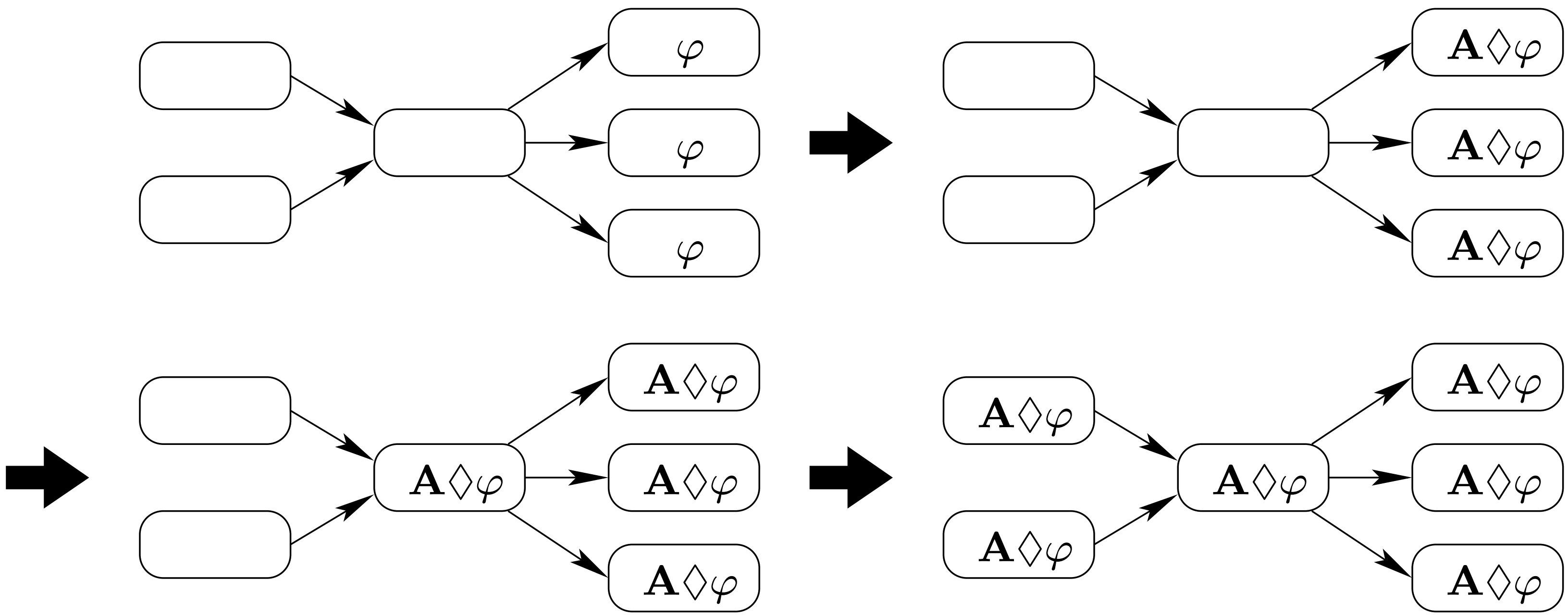
Let φ be a subformula of ϕ and states satisfying all the immediate subformulas of φ have already been labelled. We want to determine which states to label with φ. If φ is:

- $\bot$: then no states are labelled with $\bot$.
- p (atomic proposition): label s with p if $p \in L(s)$.
- $\varphi_1 \wedge \varphi_2$: label s with $\varphi_1 \wedge \varphi_2$ if s is already labelled both with φ_1 and with φ_2:
- $\neg\varphi$: label s with $\neg\varphi$ if s is not already labelled with φ.
- $\mathbf{E} \bigcirc \varphi$: label any state with $\mathbf{E} \bigcirc \varphi$ if one of its successors is labelled with φ.

Labelling Algorithm $\mathbf{A}\Diamond\varphi$

1- If any state s is labelled with φ, label it with $\mathbf{A}\Diamond\varphi$.

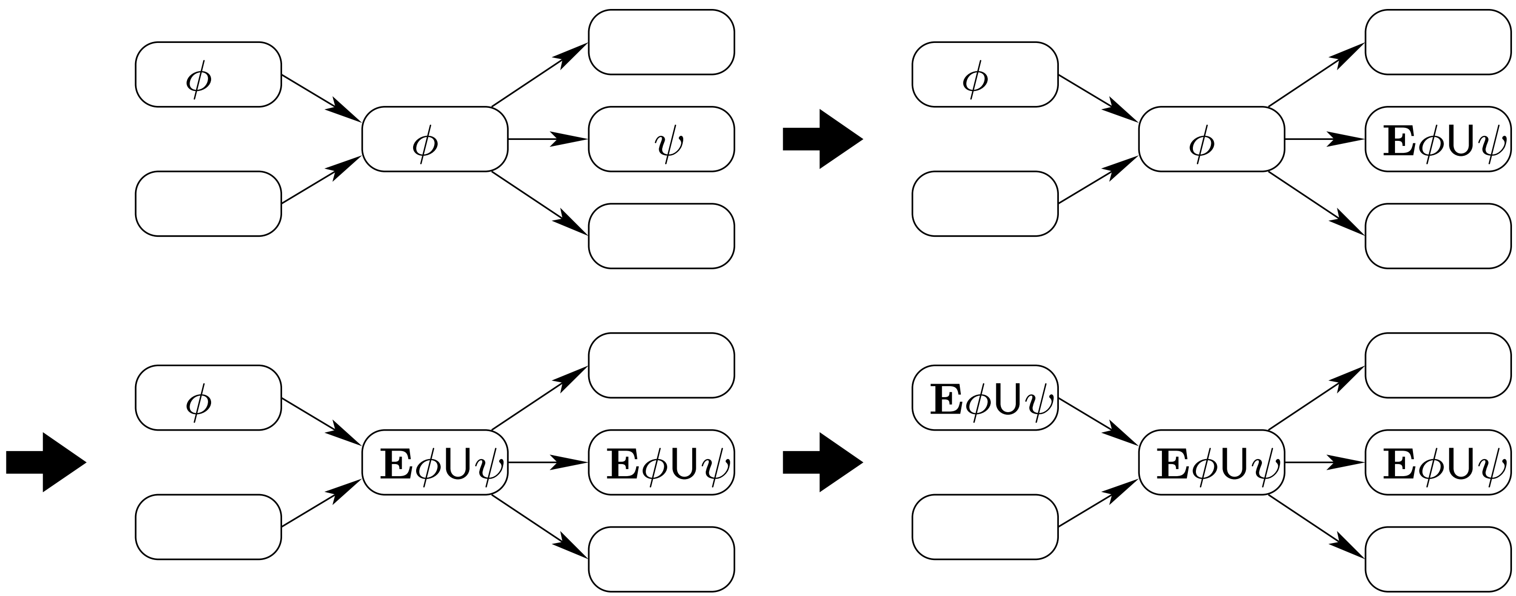
2- **Repeat:** label any state with $\mathbf{A}\Diamond\varphi$, if all successor states are labelled with $\mathbf{A}\Diamond\varphi$, until there is no change.



Labelling Algorithm: $\mathbf{E}\phi\mathsf{U}\psi$

1- If any state s is labelled with ψ, label it with $\mathbf{E}\phi\mathsf{U}\psi$.

2- **Repeat:** label any state with $\mathbf{E}\phi\mathsf{U}\psi$, if it is labelled with ϕ and at least one of its successors is labelled with $\mathbf{E}\phi\mathsf{U}\psi$, until there is no change.



Complexity: $O(S^2)$, where S is the set of reachable states.

Labelling Algorithm

Handling $\mathbf{E}\Box\varphi$ Directly

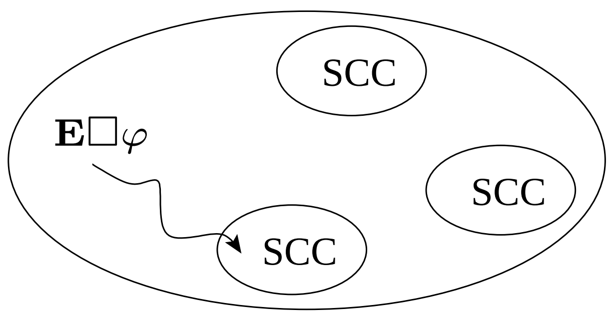
1- Label all the states that are already labelled φ, by $\mathbf{E}\Box\varphi$.

2- **Repeat:** Delete the label $\mathbf{E}\Box\varphi$ from any state if none of its successors is labelled with $\mathbf{E}\Box\varphi$; until there is no change.

Labelling Algorithm

There is even a more efficient way to handle $\mathbf{E}\Box\varphi$:

1. restrict the graph to states satisfying φ, i.e., delete all other states and their transitions;
2. find the maximal strongly connected components (SCCs); these are maximal regions of the reachable states in which every state is reachable from every other one in that region.
3. use breadth-first searching on the restricted graph to find any state that can reach an SCC.



Complexity: $O(S)$, where S is the set of reachable states.

State Space Explosion

Notice that in worst case, one has to explore the set of all states to label them:

- *Forward reachablity:* computing successor states until a fixpoint is reached
- *Backward reachability:* computing predecessor states until a fixpoint is reached

Question. Is it possible to make this computation more efficient?

Agenda

- Computation Tree Logic (CTL)
- CTL Model Checking
- *Binary Decision Diagrams (BDDs)*
- The Model Checker SMV

State Space Explosion

Exhaustive analysis may require to store all the states of the Kripke structure, and to explore them one-by-one.

The state space may be exponential in the number of components and variables (E.g., 300 Boolean vars $\Rightarrow$ up to 2^{300} 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 of *set of states* by *formulae* in propositional logic:

- manipulation of *sets of states*, rather than single states;
- manipulation of *sets of transitions*, rather than single transitions.

OBDDs

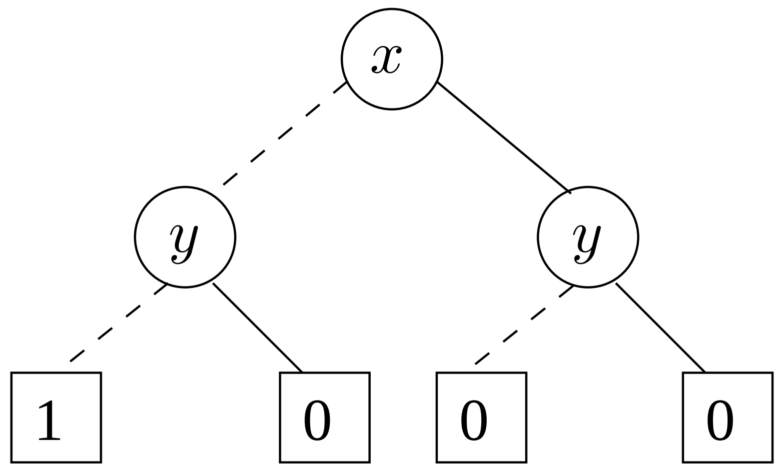
Ordered Binary Decision Diagrams (OBDD) are used to represent formulae in propositional logic.

A simple version: *Binary Decision Trees*:

- Non-Terminal nodes labelled with Boolean variables/propositions;
- Leaves (terminal nodes) are labelled with either 0 or 1;
- Two kinds of lines: dashed and solid;
- Paths leading to 1 represent models, while paths leading to 0 represent counter-models.

Binary Decision Trees

BDT representing the formula: $\phi = \neg x \wedge \neg y$:



The assignment, $x = 0$ and $y = 0$ makes true the formula.

Binary Decision Trees

Let T be a BDT, then T determines a unique Boolean formula in the following way:

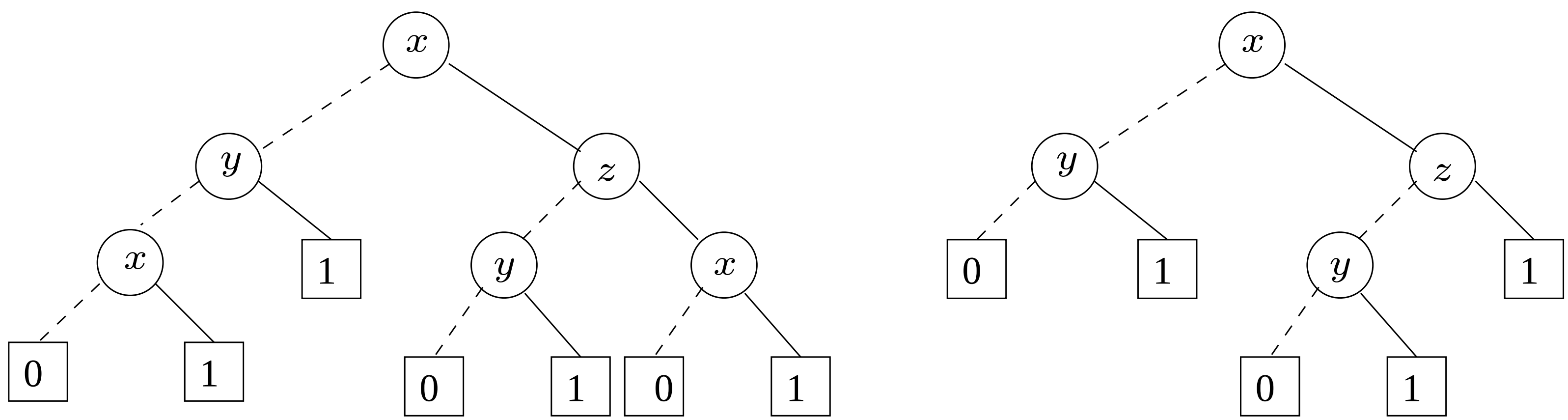
Fixed an assignment for the variables in T we start at the root and:

- If the value of the variable in the current node is 1 we follow the solid line;
- Otherwise, we follow the dashed line;
- The truth value of the formula is given by the value of the leaf we reach.

Binary Decision Trees

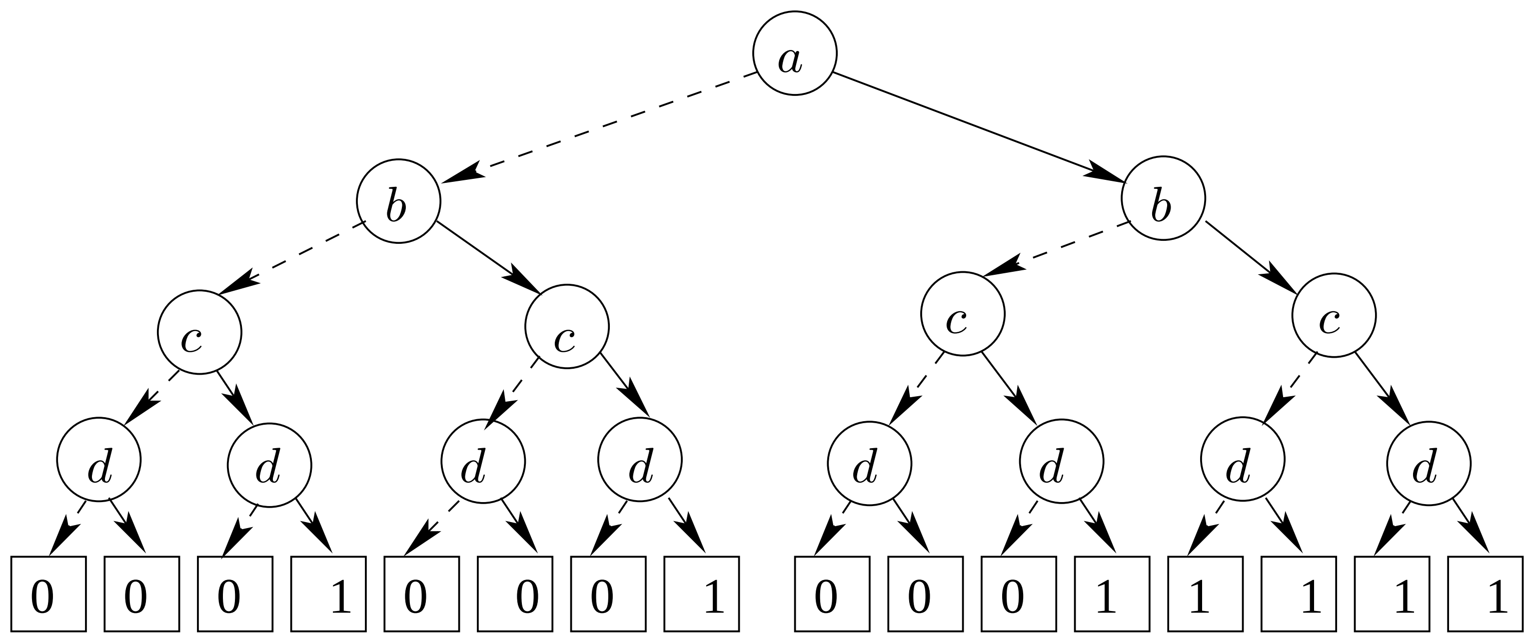
BDT's with multiple occurrences of a variable along a path are:

- Rather inefficient (Redundant paths);
- Difficult to check whether they represent the same formula (equivalence test). Example of two equivalent BDT's



Ordered Binary Decision Trees

Ordered Decision Tree (OBDT): from root to leaves variables are encountered always in the same order without repetitions along paths. Example: Ordered Decision tree for $\phi = (a \wedge b) \vee (c \wedge d)$



Reducing the Size of OBDDs

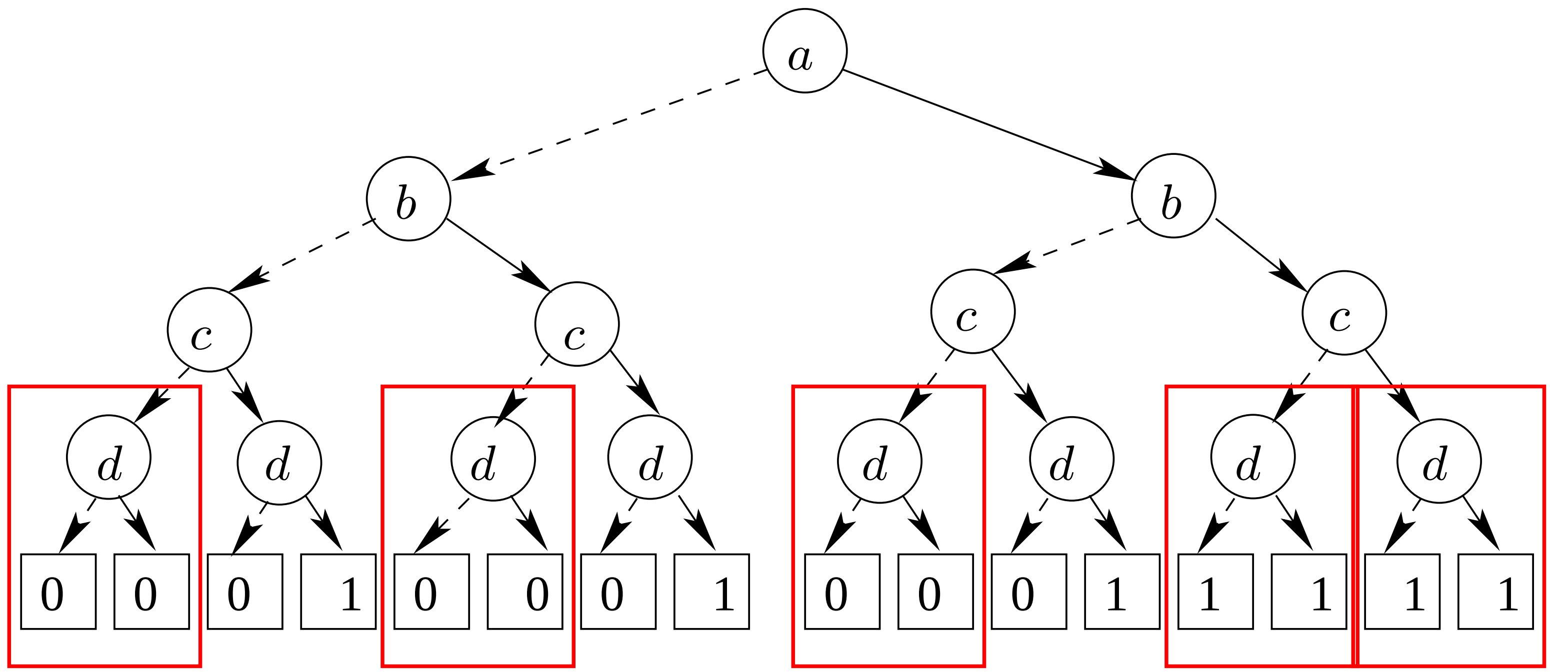
OBDT's are still exponential in the number of variables: Given n variables the OBDT's will have $2^{n+1} - 1$ nodes!

We can reduce the size of OBDT's by a recursive applications of the following reductions:

- *Remove Redundancies:* Nodes with same left and right children can be eliminated;
- *Share Subnodes:* Roots of structurally identical sub-trees can be collapsed.

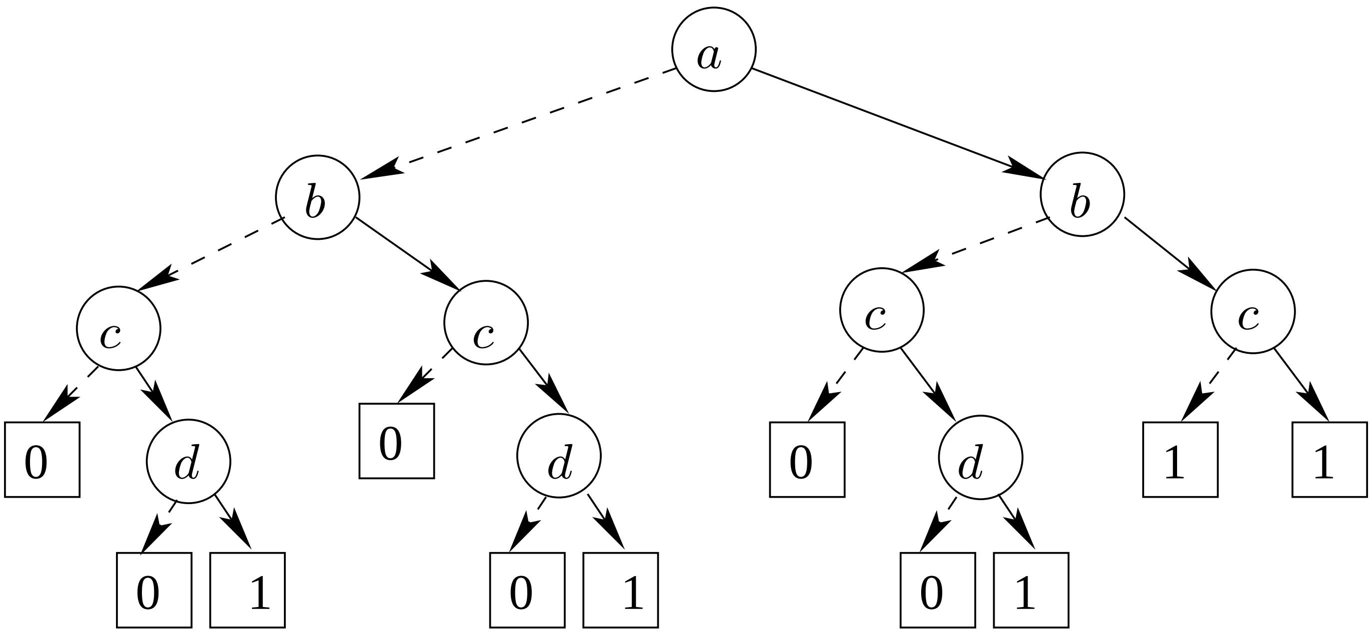
Reducing the Size of OBDDs

Remove Redundancies:



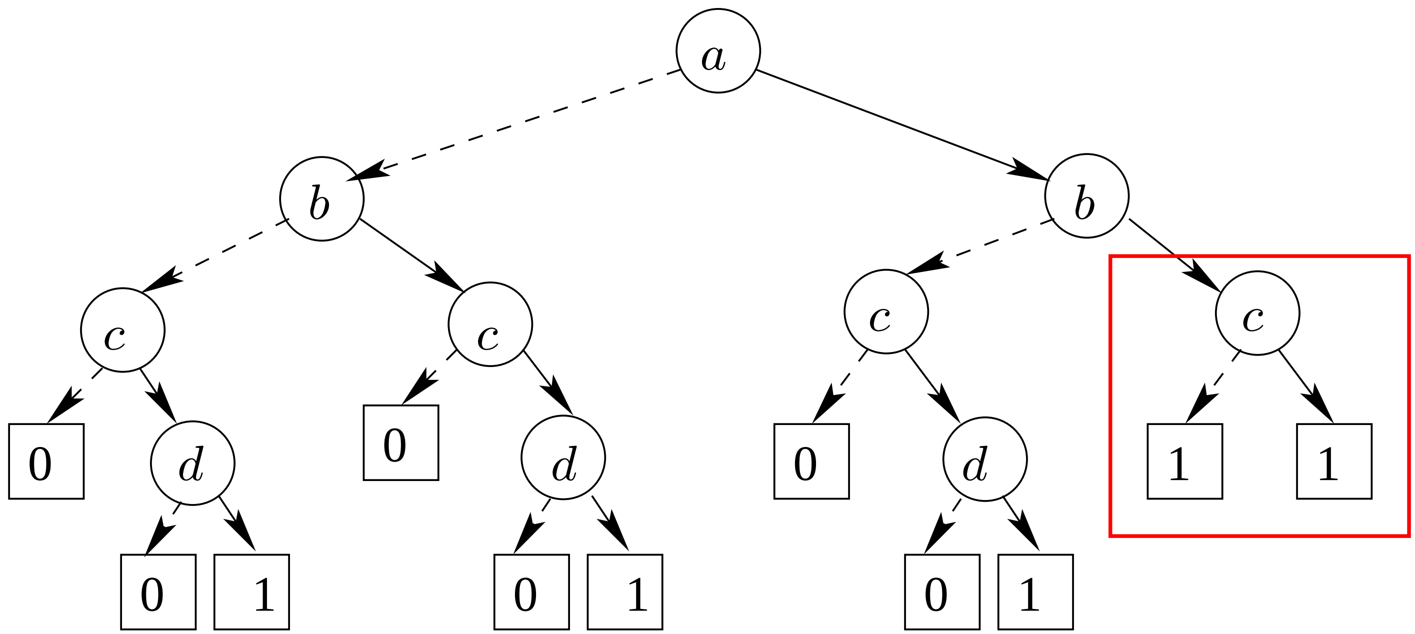
Reducing the Size of OBDDs

Remove Redundancies:



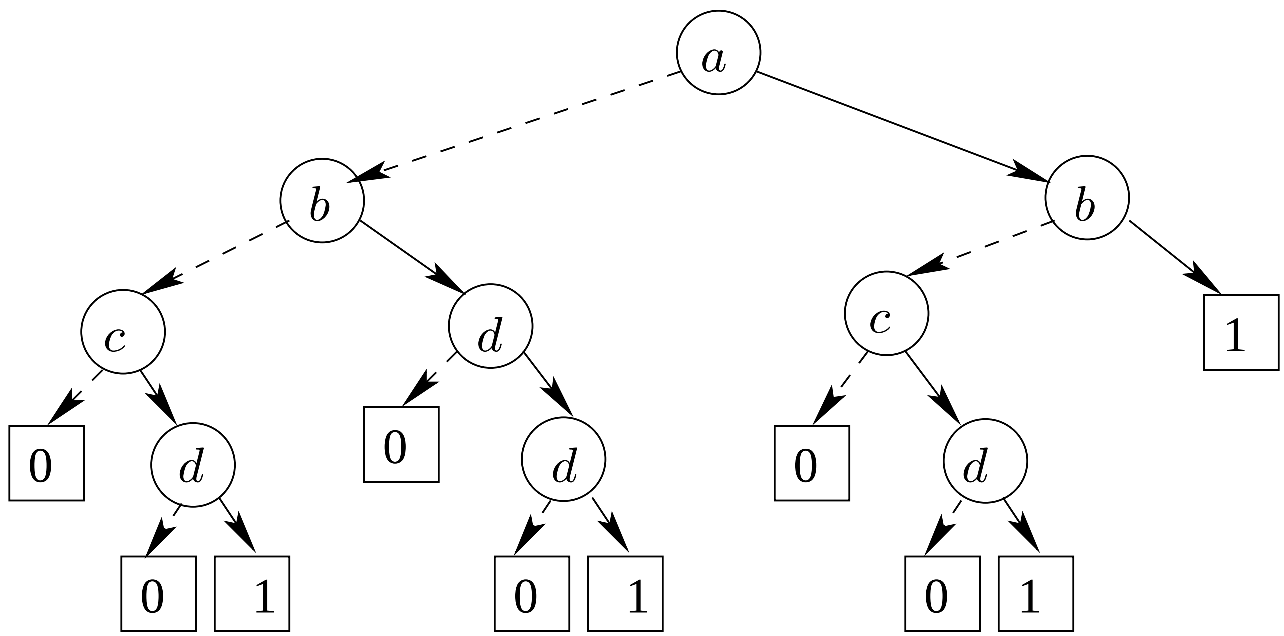
Reducing the Size of OBDDs

Remove Redundancies:



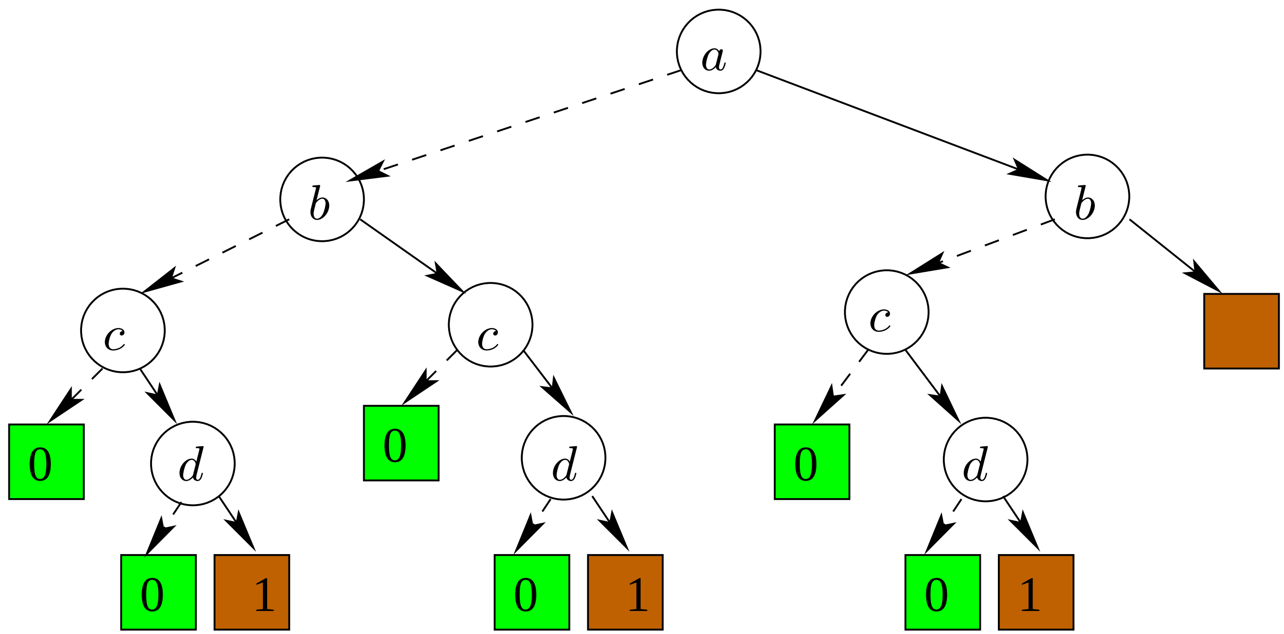
Reducing the Size of OBDDs

Remove Redundancies:



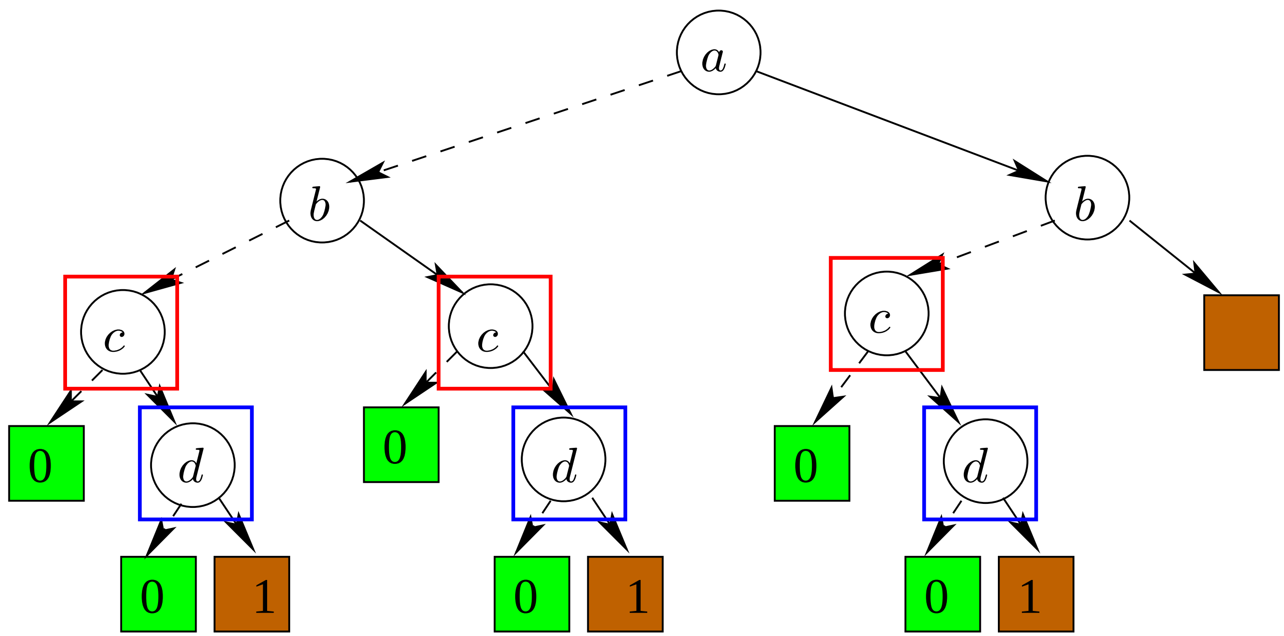
Reducing the Size of OBDDs

Share identical nodes:

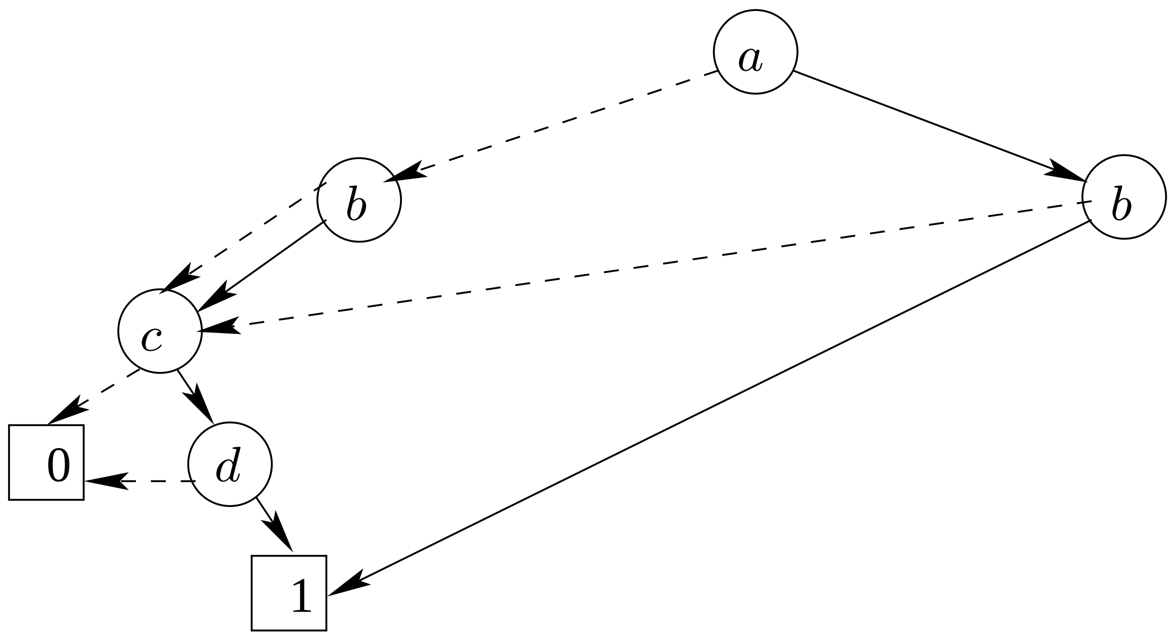


Reducing the Size of OBDDs

Share identical nodes:

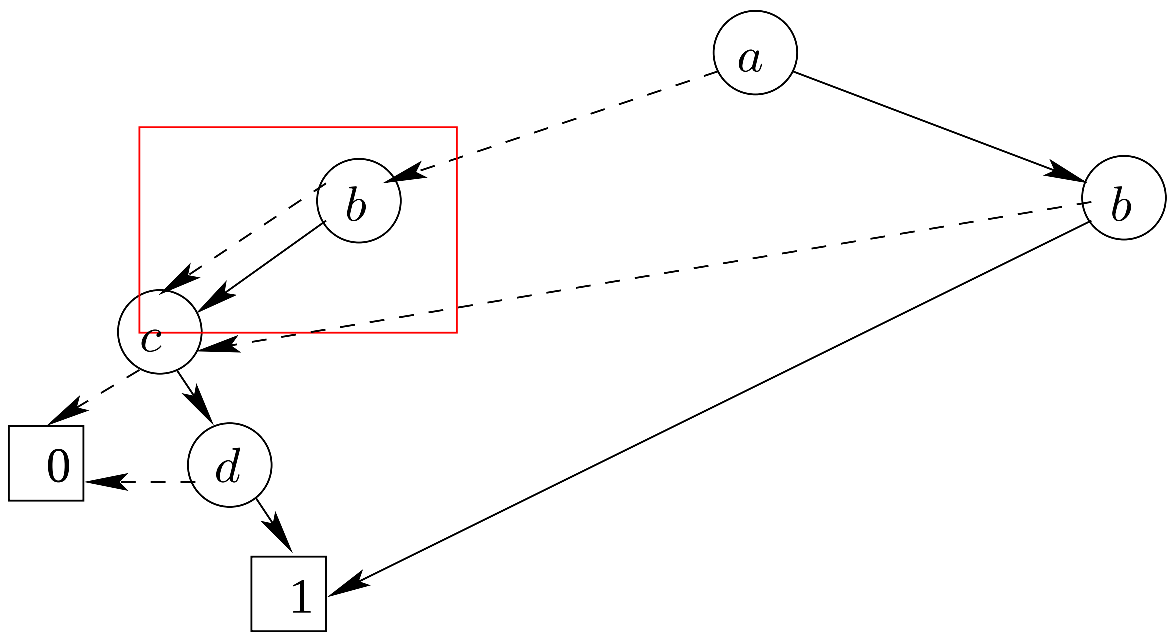


Reducing the Size of OBDDs



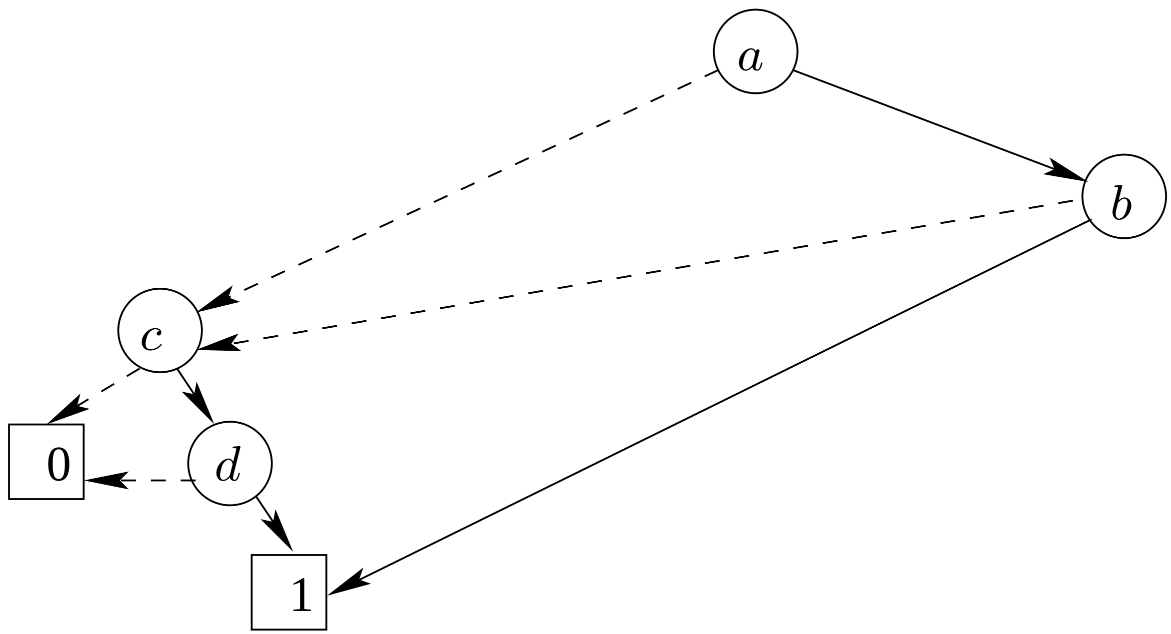
Reducing the Size of OBDDs

Remove Redundancies:



Reducing the Size of OBDDs

The final OBDD!



OBDDs as Canonical Forms

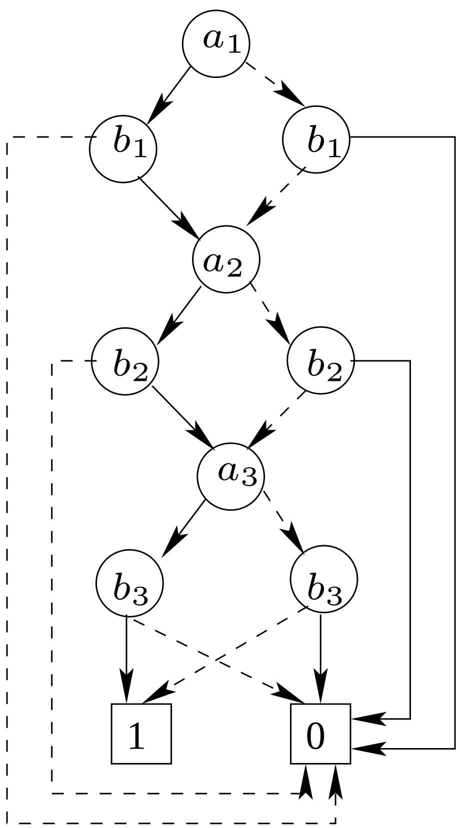
Theorem. A Reduced OBDD is a Canonical Form of a Boolean formula: Once a variable ordering is established (i.e., OBDD's have compatible variable ordering), equivalent formulae are represented by the same OBDD:

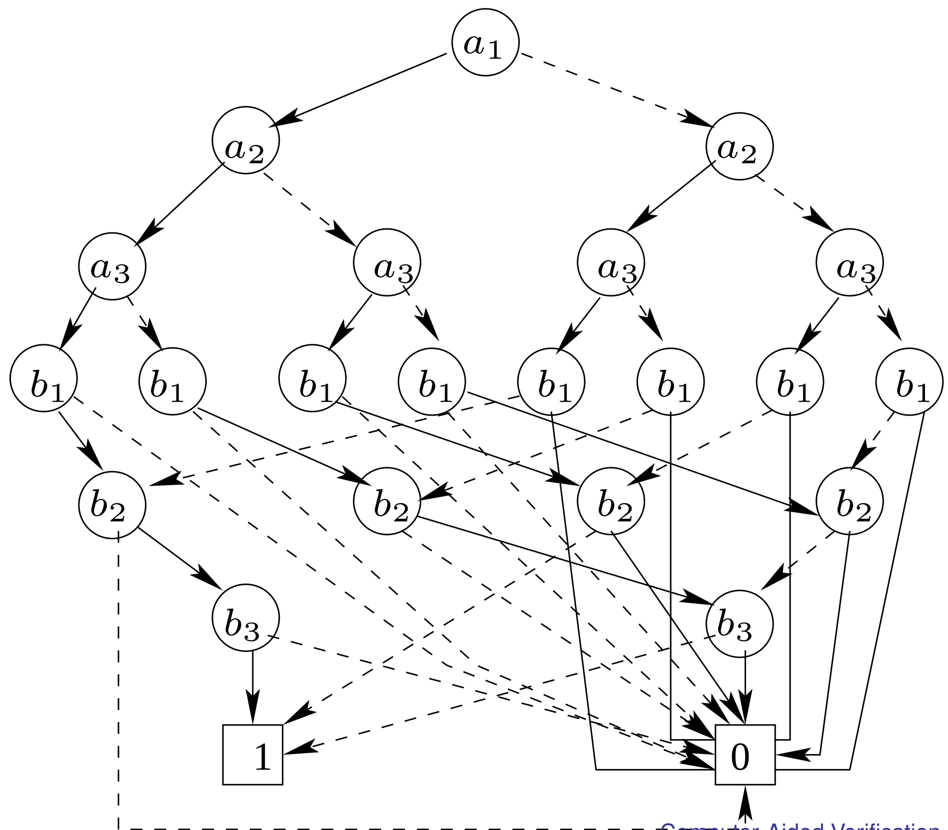
$$\phi_1 \Leftrightarrow \phi_2 \quad \text{iff} \quad OBDD(\phi_1) = OBDD(\phi_2)$$

Impact of Variable Ordering

Changing the ordering of variables may increase the size of OBDD's. Example, two OBDD's for the formula:

$$\phi = (a_1 \Leftrightarrow b_1) \wedge (a_2 \Leftrightarrow b_2) \wedge (a_3 \Leftrightarrow b_3)$$





BDD Operations

We do not cover the algorithm for constructing BDDs of propositional operators ($\wedge, \vee, \neg$). You can find the algorithm in

Randy Bryant, *Graph-Based Algorithms for Boolean Function Manipulation*.

BDD-based Reachability Analysis

```
BDD frontier = InitStates;
BDD current = bddZero();
BDD ReachableStates = InitStates;

while (ReachableStates != current)
{
        current = ReachableStates;
        BDD image = frontier * Transitions;
        frontier = Unprime(image);
        ReachableStates = current + frontier;
}
```