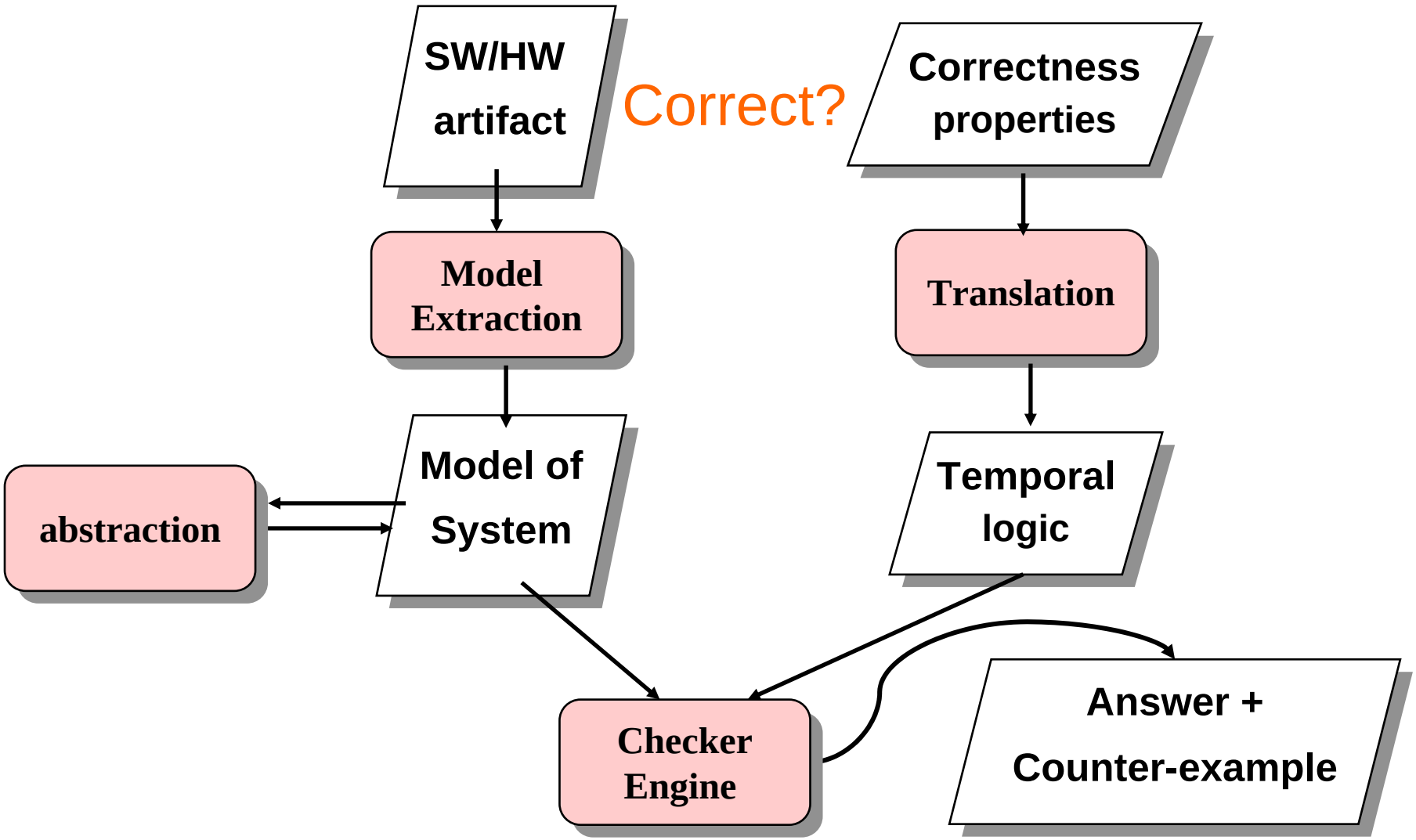
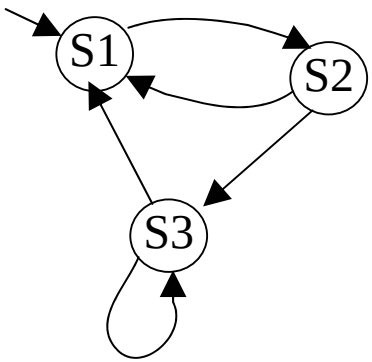
Model-Checking

Overview of Automated Verification

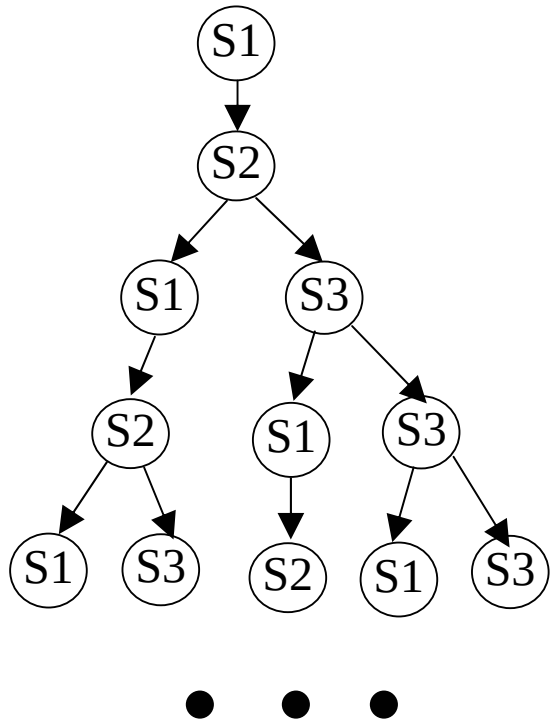


CTL Model-Checking

- CTL: Branching-time propositional temporal logic
- Model - a tree of computation paths
- Example:



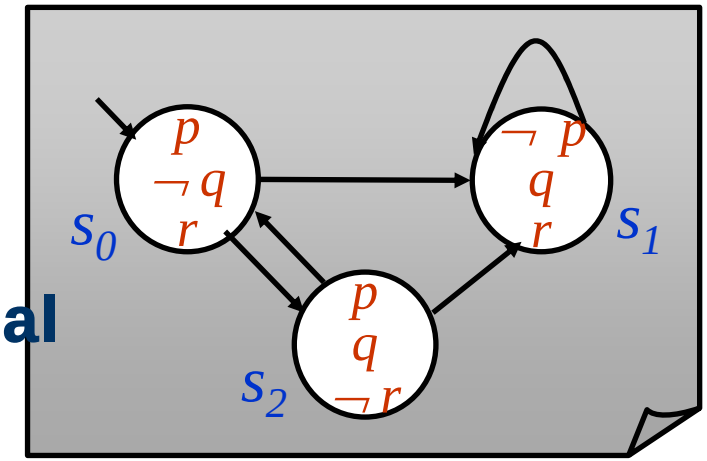
Kripke Structure



Tree of computation

Models: Kripke Structures

- Conventional state machines
  - $M = \langle S, A, s_0, I, R \rangle$
  - S is a (finite) set of states
  - A is a (finite) set of propositional variables
  - s_0 is a unique initial state ($s_0 \in S$)
  - $I: S \rightarrow 2^A$ is a labeling function that maps each state to the set of propositional variables that hold in it
  - $R \subseteq S \times S$ is a (total) transition relation



Kripke Structures (Our Model)

Formula is defined with respect to a model $M = \langle AP, S, s_0, R, I \rangle$, where

AP is a set of atomic propositions

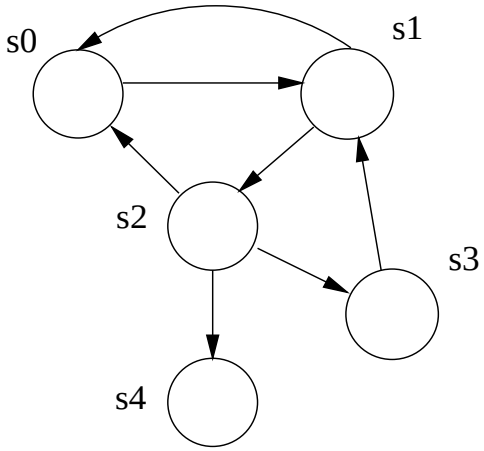
S is a set of states

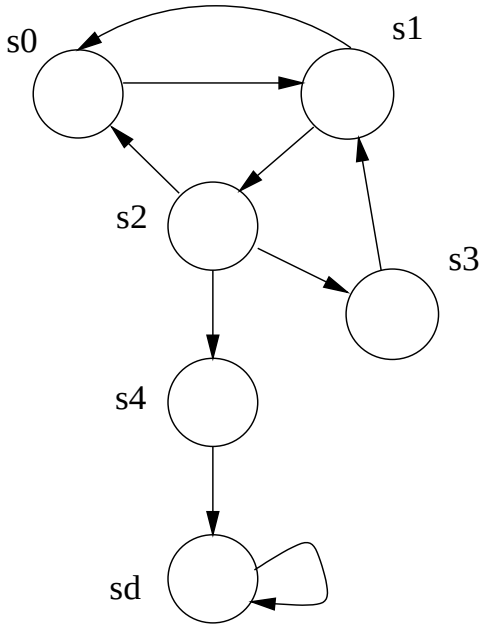
$s_0 \in S$ is the start state

R is a transition relation (every state has a successor)

I is a set of interpretations specifying which propositions are true in each state.

Example:





How to deal with deadlock states?

Propositional Variables

- **Fixed set of atomic propositions *{p, q, r}***
- **Atomic descriptions of a system**
  - **“Printer is busy”**
  - **“There are currently no requested jobs for the printer”**
  - **“Conveyer belt is stopped”**
- **How to choose them?**
- **Should not involve time!**

CTL: Computation Tree Logic

propositional temporal logic.

allows explicit quantification over possible futures

- **Syntax:**

True **(☮) and** ***False*** **(⊥) are CTL formulae;**
propositional variables are CTL formulae;

if ϕ **and** ψ **are CTL formulae, then so are:** $\neg\phi$, $\phi \wedge \psi$, $\phi \vee \psi$

EX ϕ **---** ϕ **holds in some next states;**

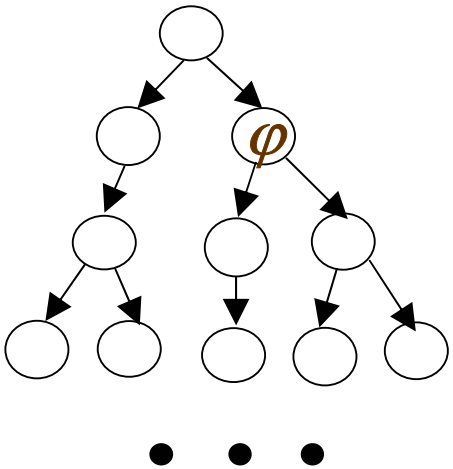
EF ϕ **--- along some path,** ϕ **is true in a future state;**

E[ϕ ***U*** ψ ***]*** **--- along some path,** ϕ **holds until** ψ **holds;**

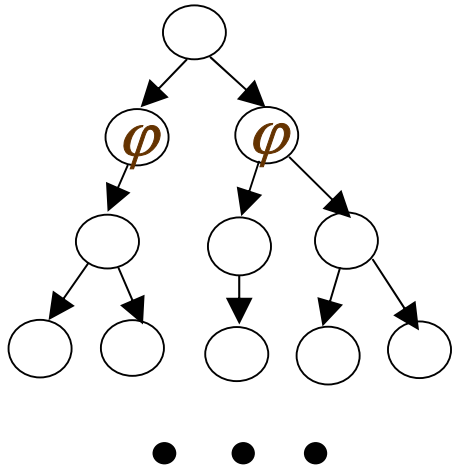
EG ϕ **--- along some path,** ϕ **holds in every state**

↳ **Universal quantification:** ***AX*** ϕ , ***AF*** ϕ , ***A[*** ϕ ***U*** ψ ***]***, ***AG*** ϕ

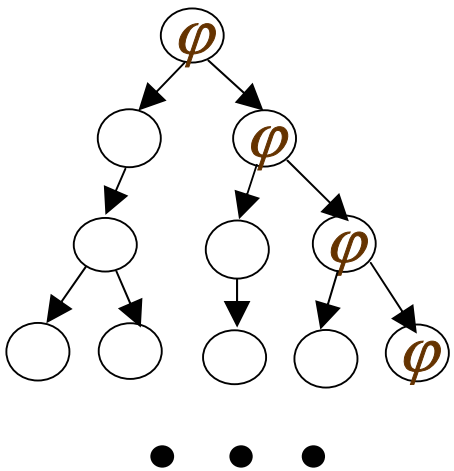
Examples



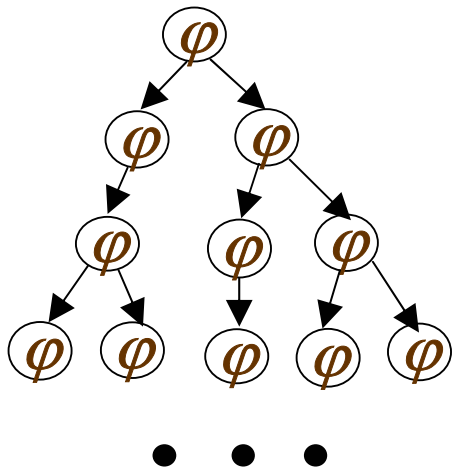
EX (exists next)



AX (all next)

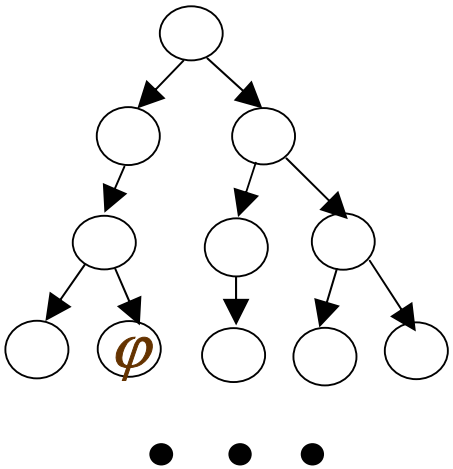


EG (exists global)

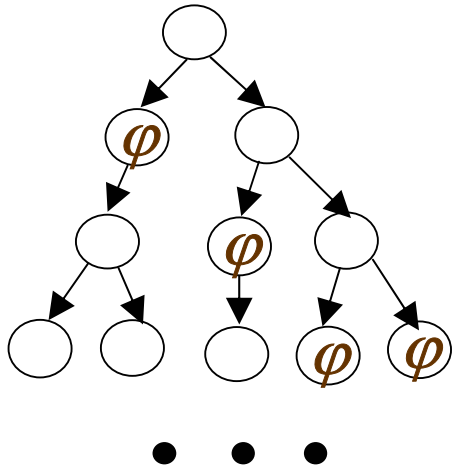


AG (all global)

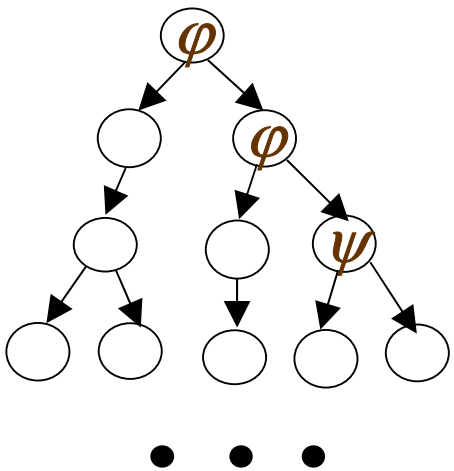
Examples, Cont'd



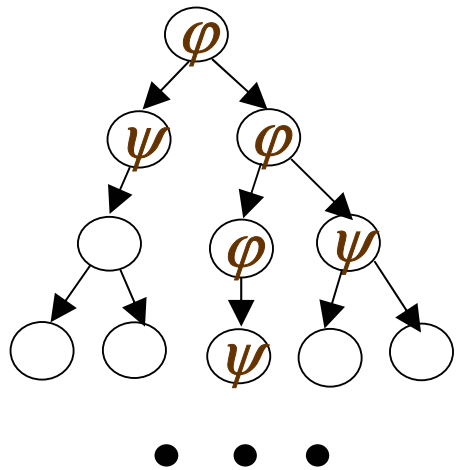
EF (exists future)



AF (all future)



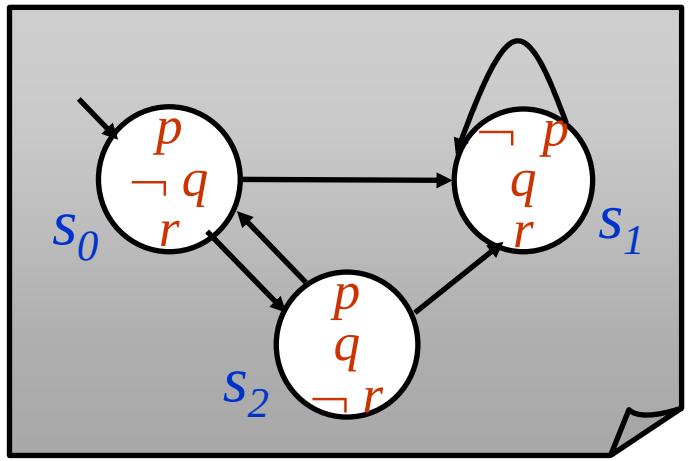
EU (exists until)



AU (all until)

CTL (Cont'd)

- **Examples:**
  - **Properties that hold:**
    - $(EX\ p)(s_0)$
    - $(A[p\ U\ q])(s_0)$
    - $(\ EX\ AF\ p\ )(s_0)$
  - **Properties that fail:**
    - $(A[\neg p\ U\ q])(s_0)$



Some Statements To Express

- It is possible to get to a state where started holds, but ready does not hold
  - EF (started $\wedge \neg$ready)
- When a request occurs, it will eventually be acknowledged
  - AG (request $\rightarrow$ AF acknowledge)
- A process is enabled infinitely often on every computation path
  - AG AF enabled
- A process will eventually be permanently deadlocked
  - AF AG deadlock
- It is always possible to get to a restart state
  - AG EF restart

Semantics of CTL

$M, s \models f$ – means that formula f is true in state s. M is often omitted since we always talk about the same model.

E.g. $s \models (x = 1) \wedge \exists n : nat \cdot y = 2 \times n$

means that in state s, variable x has value 1 and variable y has an even natural value.

$\pi = \pi^0 \ \pi^1 \ \pi^2 \ \ldots$ is a path

π^0 is the current state (root)

π^{i+1} is π^i's successor state. Then,

AX $f = \forall \pi \cdot \pi^1 \models f$

EX $f = \exists \pi \cdot \pi^1 \models f$

AG $f = \forall \pi \cdot \forall i \cdot \pi^i \models f$

EG $f = \exists \pi \cdot \forall i \cdot \pi^i \models f$

AF $f = \forall \pi \cdot \exists i \cdot \pi^i \models f$

EF $f = \exists \pi \cdot \exists i \cdot \pi^i \models f$

Semantics (Cont'd)

A[f U g] = $\forall \pi \cdot \exists i \cdot \pi^i \models g \ \wedge \ \forall j \cdot 0 \leq j < i \rightarrow \pi^j \models f$

E[f U g] = $\exists \pi \cdot \exists i \cdot \pi^i \models g \ \wedge \ \forall i \cdot 0 \leq j < i \rightarrow \pi^j \models f$

A[f R g] = $\forall \pi \cdot \forall j \geq 0 \cdot (\forall i < j \cdot \pi^i \not\models f) \rightarrow \pi^j \models g$

E[f R g] = $\exists \pi \cdot \forall j \geq 0 \cdot (\forall i < j \cdot \pi^i \not\models f) \rightarrow \pi^j \models g$

Note: the i in $\exists i \cdot$ could be 0.

Relationship between CTL operators

$$\neg\, \mathrm{AX}\, f = \mathrm{EX}\, \neg f$$

$$\neg\, \mathrm{AF}\, f = \mathrm{EG}\, \neg f$$

$$\neg\, \mathrm{EF}\, f = \mathrm{AG}\, \neg f$$

$$\mathrm{AF}\, f = \mathrm{A}[\top\, \mathrm{U}\, f]$$

$$\mathrm{EF}\, f = \mathrm{E}[\top\, \mathrm{U}\, f]$$

$$\mathrm{A}[\bot\, \mathrm{U}\, f] = \mathrm{E}[\bot\, \mathrm{U}\, f] = f$$

$$\mathrm{A}[f\, \mathrm{U}\, g] = \neg\, \mathrm{E}[\neg g\, \mathrm{U}\, (\neg f \wedge \neg g)] \wedge \neg\, \mathrm{EG}\, \neg g$$

$$\mathrm{A}[f\, \mathrm{W}\, g] = \neg\, \mathrm{E}[\neg g\, \mathrm{U}\, (\neg f \wedge \neg g)]$$

$$\mathrm{E}[f\, \mathrm{U}\, g] = \neg\, \mathrm{A}[\neg g\, \mathrm{W}\, (\neg f \wedge \neg g)]$$

$$\mathrm{AG}\, f = f \wedge \mathrm{AX}\, \mathrm{AG}\, f$$

$$\mathrm{EG}\, f = f \wedge \mathrm{EX}\, \mathrm{EG}\, f$$

$$\mathrm{AF}\, f = f \vee \mathrm{AX}\, \mathrm{AF}\, f$$

$$\mathrm{EF}\, f = f \vee \mathrm{EX}\, \mathrm{EF}\, f$$

$$\mathrm{A}[f\, \mathrm{U}\, g] = g \vee (f \wedge \mathrm{AX}\, \mathrm{A}[f\, \mathrm{U}\, g])$$

$$\mathrm{E}[f\, \mathrm{U}\, g] = g \vee (f \wedge \mathrm{EX}\, \mathrm{E}[f\, \mathrm{U}\, g])$$

$$\neg\, \mathrm{E}[f\, \mathrm{U}\, g] = \mathrm{A}[\neg f\, \mathrm{R}\, \neg g]$$

$$\neg\, \mathrm{A}[f\, \mathrm{U}\, g] = \mathrm{E}[\neg f\, \mathrm{R}\, \neg g]$$

Adequate Sets

Definition: A set of connectives is *adequate* if all connectives can be expressed using it.

Example: $\{\neg, \wedge\}$ is adequate for propositional logic

Theorem: The set of operators $\bot$, $\neg$ and $\wedge$ together with EX, EG, and EU are adequate for CTL.

Other adequate sets: $\{\text{AU, EU, EX}\}$, $\{\text{AF, EU, EX}\}$

Theorem: Every adequate set has to include EU.

LTL

- If p is an atomic propositional formula, it is a formula in LTL.
- If p and q are LTL formulas, so are $p \wedge q$, $p \vee q$, $\neg p$, p U q, p W q, p R q, $\circ p$ (next), $\diamond p$(eventually), $\Box p$ (always)

Interpretation: over *computations* $\pi : \omega \Rightarrow 2^{AP}$ which assigns truth values to the elements of AP at each time instant:

- $\pi \models \circ f$ iff $\pi^1 \models f$
- $\pi \models f \ \mathrm{U} \ g$ iff $\exists i \cdot \pi^i \models f \ \wedge \ \forall j \cdot 0 \leq j < i \rightarrow \pi^j \models f$
- $\pi \models \Box f$ iff $\forall i \cdot \pi^i \models f$
- $\pi \models \diamond f$ iff $\exists i \cdot \pi^i \models f$

Here, π^0 – initial state of the system

Two other operators:

- p W q = $\Box p \vee (p \ \mathrm{U} \ q)$ (p unless q, p waiting for q, p week-until q)
- p R q = $\neg(\neg p \ \mathrm{U} \ \neg q)$ (release)

Some Temporal Properties

$$
\begin{aligned}
\neg \circ p &= \circ \neg p \\
\Diamond p &= \mathit{True}\ \mathrm{U}\ p \\
\Box p &= \neg \Diamond \neg p \\
p\ \mathrm{W}\ q &= \Box p \vee (p\ \mathrm{U}\ q) \\
p\ \mathrm{R}\ q &= \neg(\neg p\ \mathrm{U}\ \neg q) \\
\Box p &= p \wedge \circ \Box p \\
\Diamond p &= p \vee \circ \Diamond p \\
p\ \mathrm{U}\ q &= q \vee (p \wedge \circ(p\ \mathrm{U}\ q))
\end{aligned}
$$

A property φ holds in a model if it holds on every path emanating from the initial state.

Expressing Properties in LTL

Good for safety ($\Box\neg$) and liveness ($\Diamond$) properties.

- $p \rightarrow \Diamond q$ – If p holds in initial state s_0, then q holds at s_j for some $j \geq 0$.
- $\Box \Diamond q$ – Each path contains infinitely many q's.
- $\Diamond \Box q$ – At most a finite number of states in each path satisfy $\neg q$. Property q eventually stabilizes.
- $\Box(p \text{ U } q)$ – always p remains true at least until q becomes true.
- $\neg(\Diamond(p \text{ U } q))$ – never is there a point in the execution such that p remains true at least until q becomes true.

Express: it is not true that p is true at least until the point s.t. for all paths q is true at least until r is true

Comparison of LTL and CTL

Syntactically: LTL simpler than CTL

Semantically: incomparable!

- CTL formula EF p is not expressible in LTL
- LTL formula $\diamond\Box p$ not expressible in CTL.

Question: What about AF AG p?

Model: self-loop on p, transition on $\neg p$ to a state with a self-loop on p.

Most useful formulas expressible in both:

- Invariance: $\Box p$ and AG p
- Liveness (response): $\Box(p \rightarrow \diamond q)$ and AG($p \rightarrow$ AFq).

LTL and CTL coincide if the model has only one path!

CTL Model-Checking

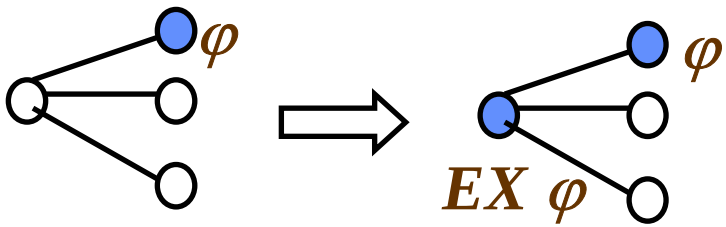
- **Receive:**
  - **Kripke structure *K***
  - **Temporal logic formula φ**
- **Assumptions:**
  - **Finite number of processes**
    - **Each having a finite number of finite-valued variables**
  - **Finite length of a CTL formula**
- **Algorithm:**
  - **Label states of *K* with subformulas of that φ are satisfied there and working outwards towards φ.**
  - **Output states labeled with φ**

Example: EX AG ($p \rightarrow$ E[p U q])

CTL Model-Checking (Cont'd)

EX φ

- Label any state with EX φ if any of its successors are labeled with φ



AF φ

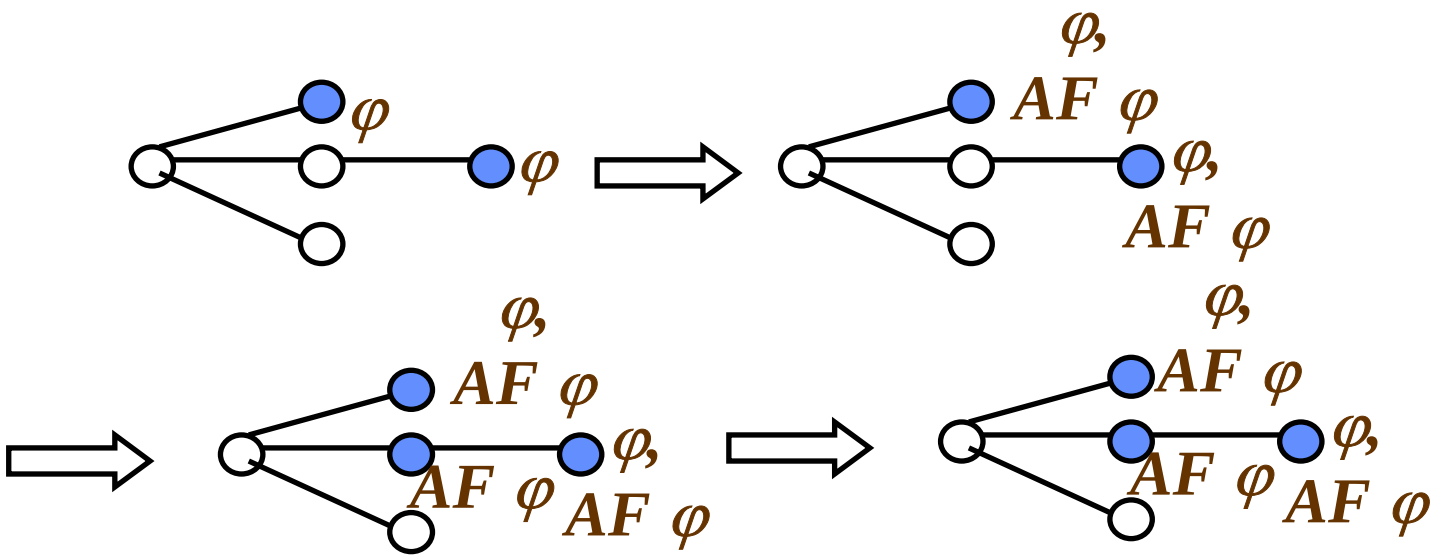
- If any state s is labeled with φ, label it with AFφ
- Repeat:

  label any state with AFφ

  if all of its successors

  are labeled with AFφ

until there is no change



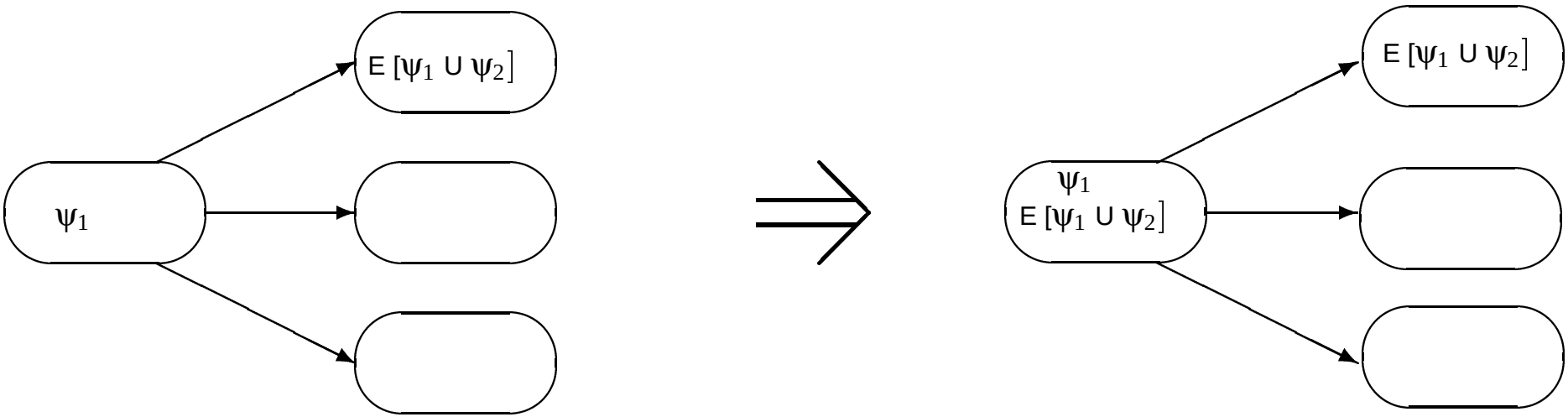
Labeling Algorithm (Cont'd)

- E [ψ_1 U ψ_2]:

  - If any state s is labeled with ψ_2, label it with E[ψ_1 U ψ_2].

  - Repeat: label any state with E[ψ_1 U ψ_2] if it is labeled with ψ_1 and at least one of its successors is labeled with E[ψ_1 U ψ_2], until there is no change.

Ex:

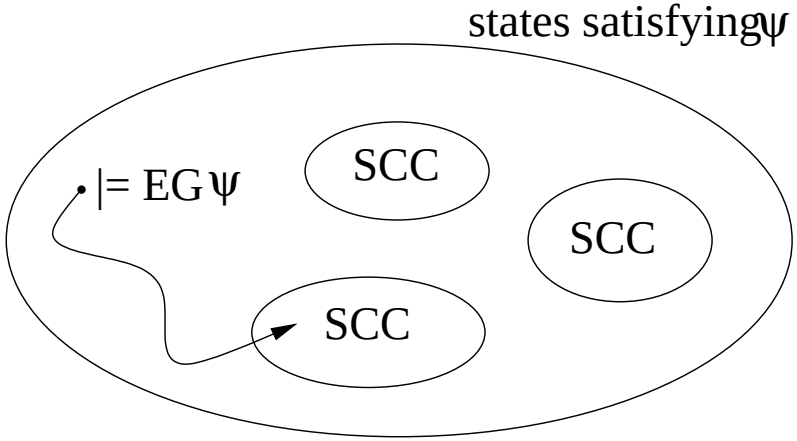


Output states labeled with f.

Complexity: $O(|f| \times S \times (S + |R|))$ (linear in the size of the formula and quadratic in the size of the model).

Better Handling of EG

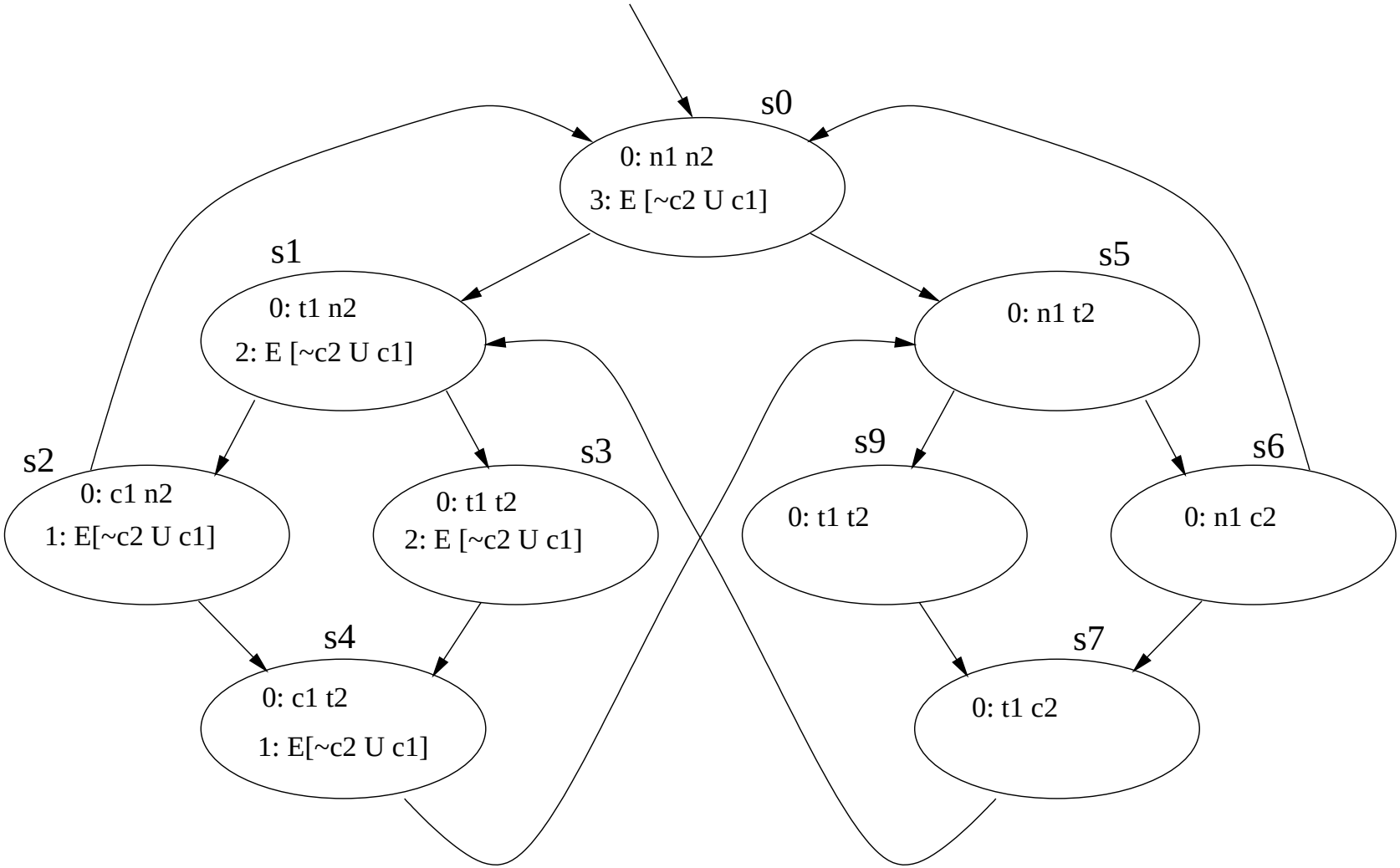
- restrict the graph to states satisfying ψ_1, i.e., delete all other states and their transitions;
- find the maximal *strongly connected components* (SCCs); these are maximal regions of the state space in which every state is linked with every other one in that region.
- use breadth-first searching on the restricted graph to find any state that can reach an SCC.



Complexity: $O(|f| \times (S + |R|))$ (linear in size of model and size of formula).

Example

Verifying E[$\neg c_2$ U c_1] on the mutual exclusion example.

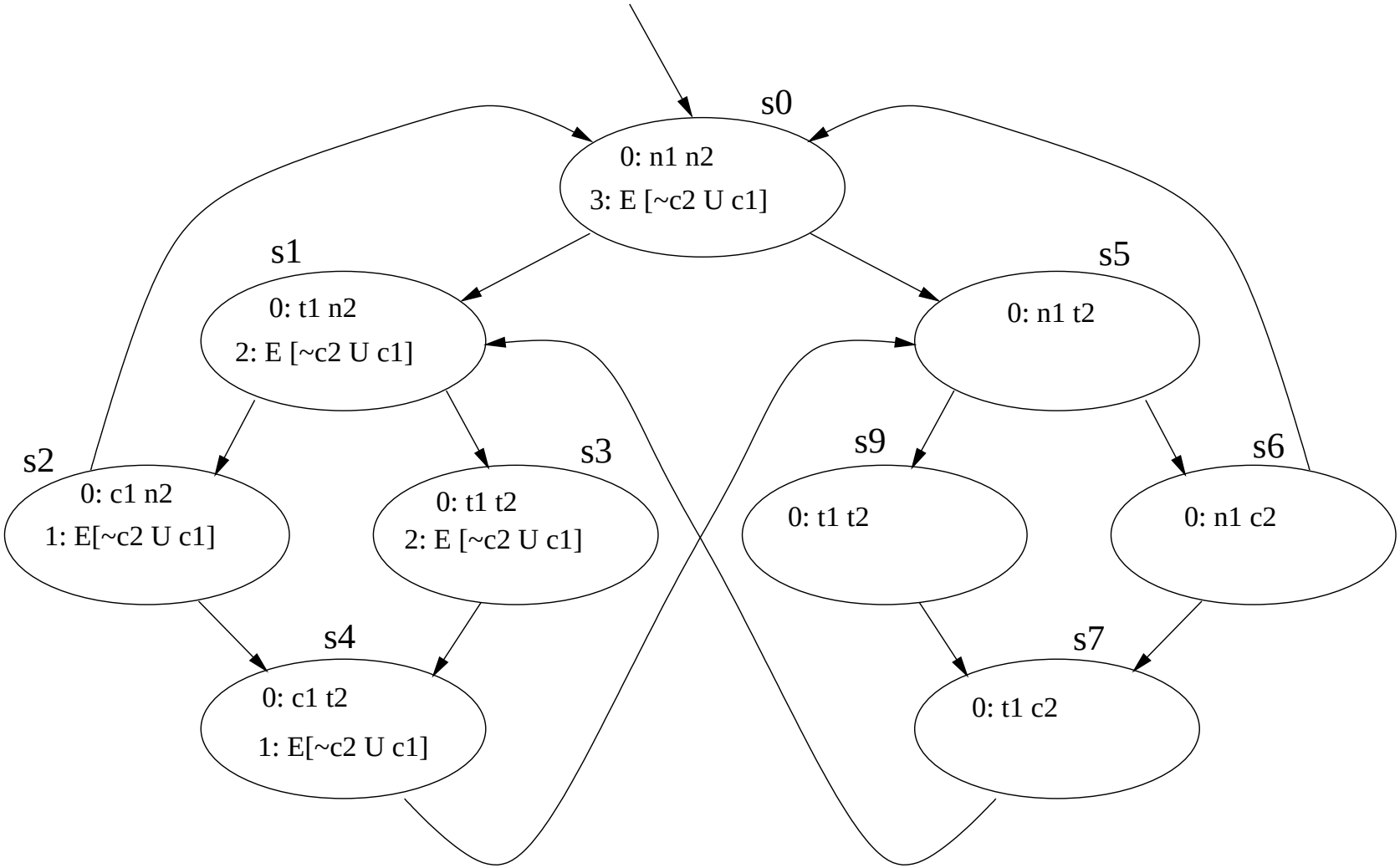


CTL Model-Checking

- Michael Browne, CMU, 1989.
- Usually for verifying concurrent *synchronous* systems (hardware, SCR specs...)
- Specify correctness criteria: safety, liveness...
- Instead of keeping track of labels for each state, keep track of a set of states in which a certain formula holds.

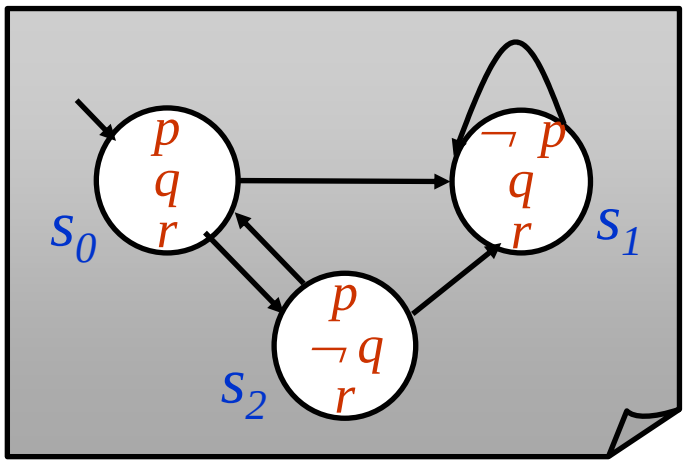
Example

Verifying E[$\neg c_2$ U c_1] on the mutual exclusion example.

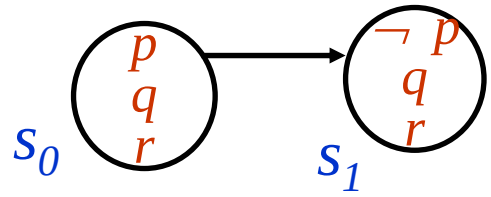


Counterexamples

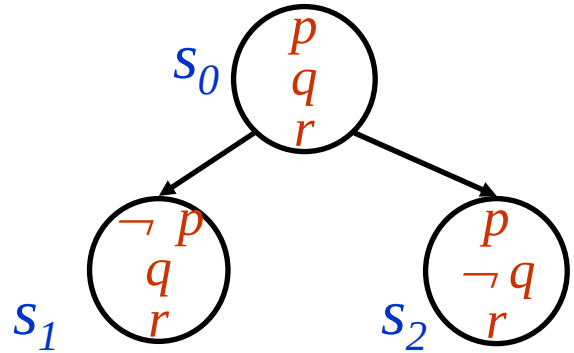
- **Explain:**
  - **Why the property fails to hold**
  - **to disprove that ϕ holds on all elements of S, produce a single element $s \in S$ s.t. $\neg\phi$ holds on s.**
    - **counterexamples restricted to universally-quantified formulas**
    - **counterexamples are paths (trees) from initial state illustrating the failure of property**



AG p



AX p $\vee$ AX q



Counterexamples and Witnesses

- Counterexamples
  - explains why a property is false
  - typically a violating path for universal properties
  - how to explain that something does not exist?
- Witnesses
  - explains why a property is true
  - typically a satisfying path for existential properties
  - how to explain that something holds on all paths?

Generating Counterexamples

Only works for universal properties

- AXp
- $AG(p \Rightarrow AFq)$
- etc.

Step 1: negate the property and express it using EX, EU, and EG

- e.g. $AG(p \Rightarrow AFq)$ becomes $EF(p \wedge EG\neg q)$

Step 2:

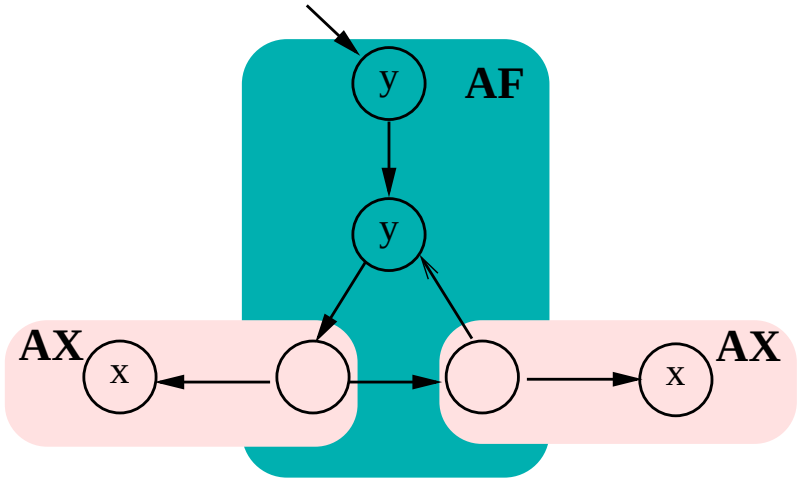
- For EXp – find a successor state labeled with p
- For EGp – follow successors labeled with EGp until a loop is found
- For $E[pUq]$ – remove all states not labeled with p or q, then look for path to q

Counterexamples and Witnesses (Cont'd)

- What about properties that combine universal and existential operators?
- Are they really different?
  - a counterexample for φ is a witness to its negation
  - a counterexample for a universal property is a witness to some existential property
  - e.g. AGp and $EF\neg p$
- One alternative
  - build a proof instead of a counterexample
  - works for all properties (but proofs can be big)
  - see:
    - A. Gurfinkel and M. Chechik. "Proof-like Counterexamples", Proceedings of TACAS'03.
    - M. Chechik, A. Gurfinkel. "A Framework for Counterexample Generation and Exploration", FASE'2005.

Are counterexamples always linear?

- SMV only supports linear counterexamples
- But what about $(AX\,p) \vee (AX\,q)$?
- Counterexample for $AF\,(\neg y \wedge AX\,\neg x)$



  - See: E. Clarke et al. “Tree-Like Counterexamples in Model Checking”, Proceedings of LICS’02.

State Explosion

Imagine that you a Kripke structure of size n. What happens if we add another boolean variable to our model?

How to deal with this problem?

- Symbolic model checking with efficient data structures (BDDs). Don't need to represent and manipulate the entire model. Model-checker SMV [McMillan, 1993].

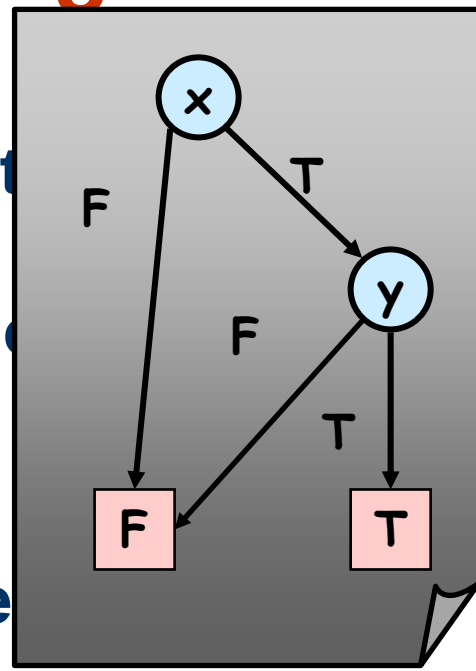
- Abstraction: we abstract away variables in the model which are not relevant to the formula being checked (see later in the course).

- Partial order reduction: for asynchronous systems, several interleavings of component traces may be equivalent as far as satisfaction of the formula to be checked is concerned.

- Composition: break the verification problem down into several simpler verification problems.

Symbolic model-checking

- Idea of model-checking
  - recursively go through the structure of t CTL property…
  - associating each subformula with a set c states where each subproperty is true
- Symbolic model-checking
  - effective cure for state explosion proble
  - use symbolic representation for sets of states
  - use symbolic representation for transition relation
  - use binary decision diagrams (BDDs) to encode these
- Example:
  - $x \wedge y$ in classical logic



SMV

Symbolic model verifier – a model-checker that uses symbolic model checking algorithm. The language for describing the model is a simple parallel assignment.

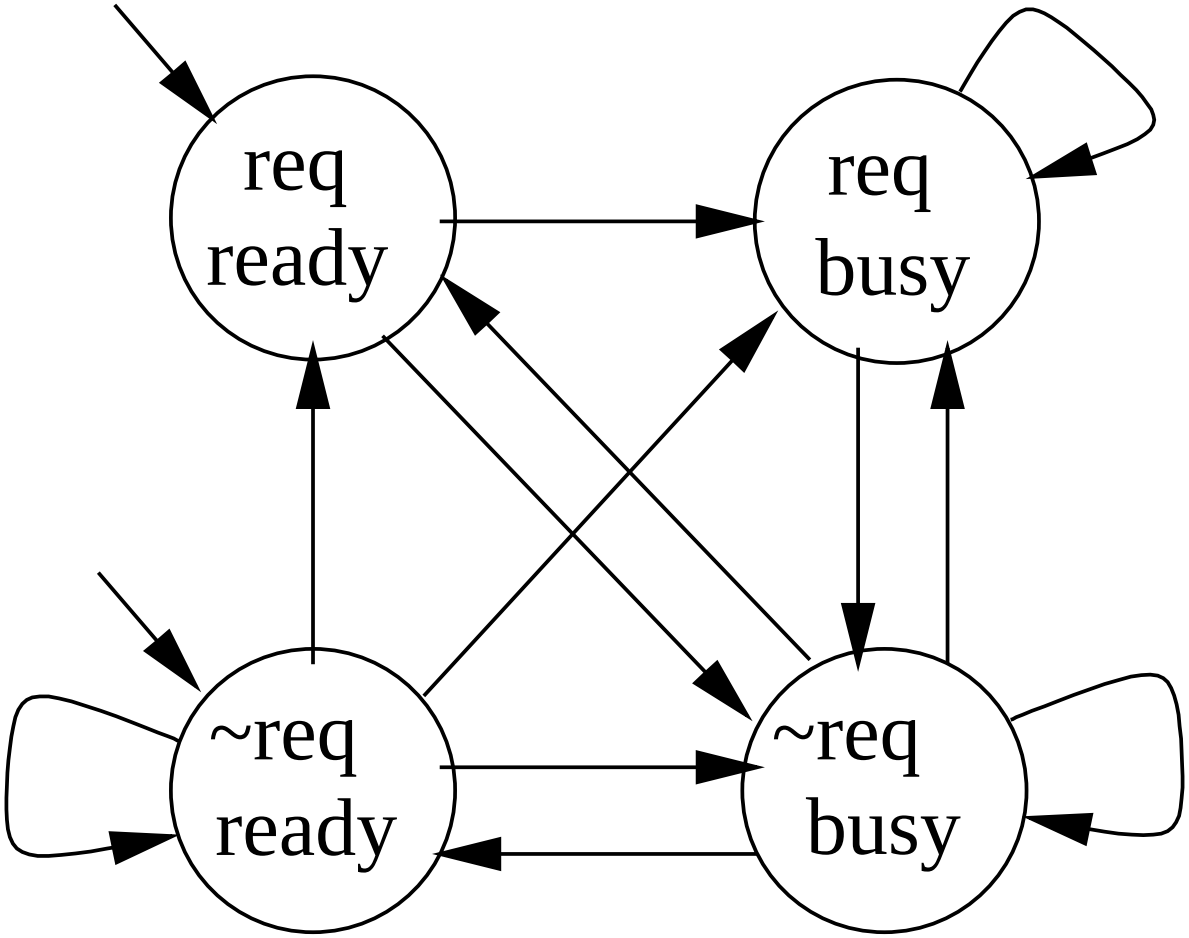
- Can have synchronous or asynchronous parallelism.
- Model environment non-deterministically.
- Also use non-determinism for systems which are not fully implemented or are abstract models of complex systems.

First SMV Example

```
MODULE main
VAR
  request : boolean;
  state : {ready, busy};
ASSIGN
  init(state) := ready;
  next(state) := case
     request : busy;
     1: {ready, busy}
  esac;
SPEC
  AG(request -> AF state = busy)
```

Note that `request` never receives an assignment – this models input.

Model for First SMV Example



Binary Decision Diagrams

- Representation of Boolean Functions
- BDDs, OBDDs, ROBDDs
- Operations
- Model-Checking over BDDs

Boolean Functions

Boolean functions: $\mathcal{B} = \{0, 1\}$,

$$f : \mathcal{B} \times \cdots \times \mathcal{B} \to \mathcal{B}$$

Boolean expressions:

$$t ::= x \mid 0 \mid 1 \mid \neg t \mid t \wedge t \mid t \vee t \mid t \to t \mid t \leftarrow t$$

Truth assignments: ρ,

$$[v_1/x_1, v_2/x_2, \cdots, v_n/x_n]$$

Satisfiable: Exists ρ s.t. $t[\rho] = 1$

Tautology: Forall ρ, $t[\rho] = 1$

What is a good representation of boolean functions?

Perfect representation is hopeless:

Theorem 1 (Cook's Theorem)

Satisfiability of Boolean expressions is NP-complete.

(Tautology-checking is co-NP-complete)

Good representations are

compact and

efficient

on real-life examples

Shannon Expansion

Def: $x \rightarrow y_0, y_1 = (x \wedge y_0) \vee (\neg x \wedge y_1)$

x is the test expression and thus this is an if-then-else.

We can represent all operators using if-then-else on unnegated variables and constants 0(false) and 1(true). This is called INF.

Shannon expansion w.r.t. x:

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

Any boolean expression is equivalent to an expression in INF.

Example

$t = (x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2)$. Represent this in INF form with order x_1, y_1, x_2, y_2.

$t = x_1 \rightarrow t_1, t_0$

$t_0 = y_1 \rightarrow 0, t_{00}$

(since $x_1 = 1, y_1 = 0 \rightarrow t = 0$)

$t_1 = y_1 \rightarrow t_{11}, 0$

(since $x_1 = 0, y_1 = 1 \rightarrow t = 0$)

$t_{00} = x_2 \rightarrow t_{001}, t_{000}$

$t_{11} = x_2 \rightarrow t_{111}, t_{000}$

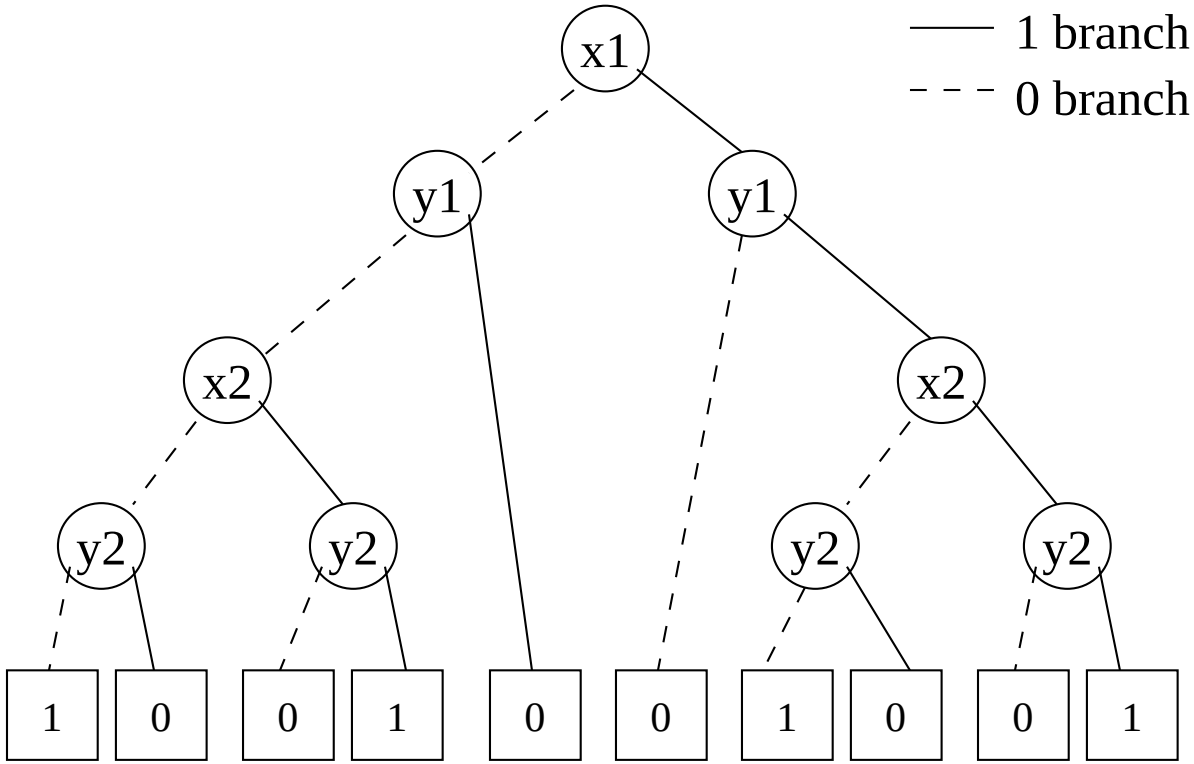
$t_{000} = y_2 \rightarrow 0, 1 \quad (x_1 = 0, y_1 = 0, x_2 = 0)$

$t_{001} = y_2 \rightarrow 1, 0 \quad (x_1 = 0, y_1 = 0, x_2 = 1)$

$t_{110} = y_2 \rightarrow 0, 1 \quad (x_1 = 1, y_1 = 1, x_2 = 0)$

$t_{111} = y_2 \rightarrow 1, 0 \quad (x_1 = 1, y_1 = 1, x_2 = 1)$

Decision Tree:



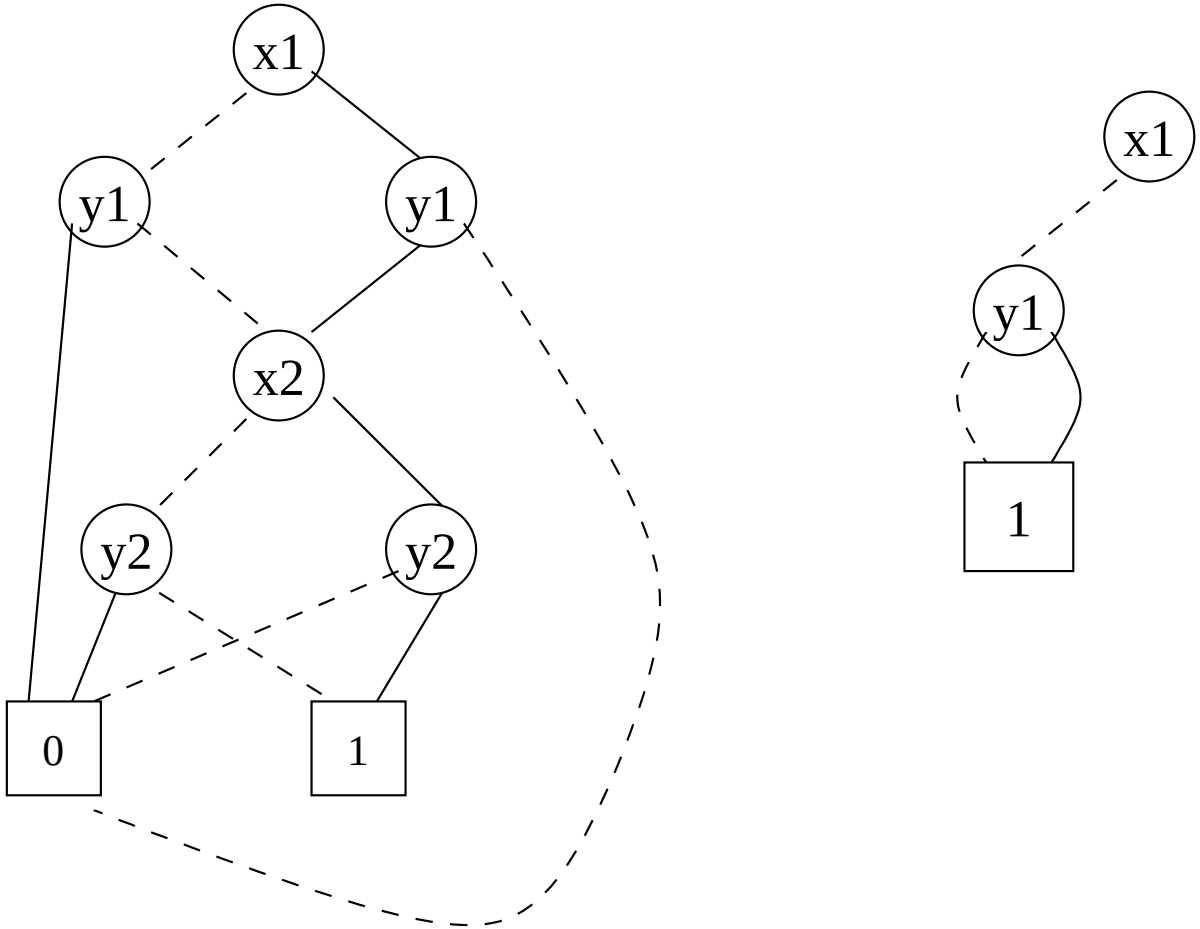
Lots of common subexpressions:

- identify them!

BDDs – directed acyclic graph of Boolean expressions. If the variables occur in the same ordering on all paths from root to leaves, we call this OBDD.

Example OBDD

OBDD for $(x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2)$ with ordering $x_1 < y_1 < x_2 < y_2$



If an OBDD does not contain any redundant tests, it is called ROBDD.

ROBDDs

A *Binary Decision Diagram* is a rooted, directed, acyclic graph (V, E). V contains (up to) two terminal vertices, $0, 1 \in V$. $v \in V \setminus \{0, 1\}$ are non-terminal and have attributes *var*(v), and *low*(v), *high*(v) $\in V$.

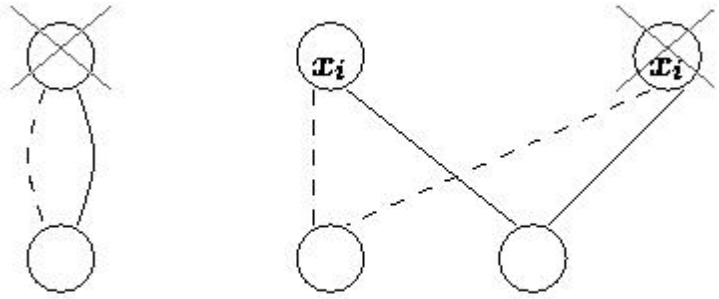
A BDD is *ordered* if on all paths from the root the variables respect a given total order.

A BDD is *reduced* if for all non-terminal vertices u, v,

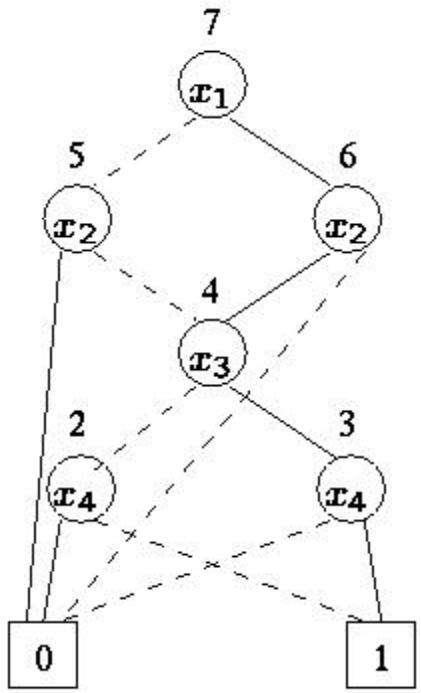
1) *low*(u) $\neq$ *high*(u)

2) *low*(u) = *low*(v), *high*(u) = *high*(v), *var*(u) = *var*(v) implies $u = v$.

ROBDD Examples



reducedness



$(x_1 \Leftrightarrow x_2) \wedge (x_3 \Leftrightarrow x_4)$

Canonicity of ROBDDs

Lemma 1 (Canonicity lemma) For any function $f : \mathcal{B}^n \to \mathcal{B}$ there is exactly one ROBDD b with variables $x_1 < x_2 < \cdots < x_n$ such that

$$t_b[v_1/x_1, \cdots, v_n/x_n] = f(v_1, \cdots, v_n)$$

for all $(v_1, ..., v_n) \in \mathcal{B}^n$.

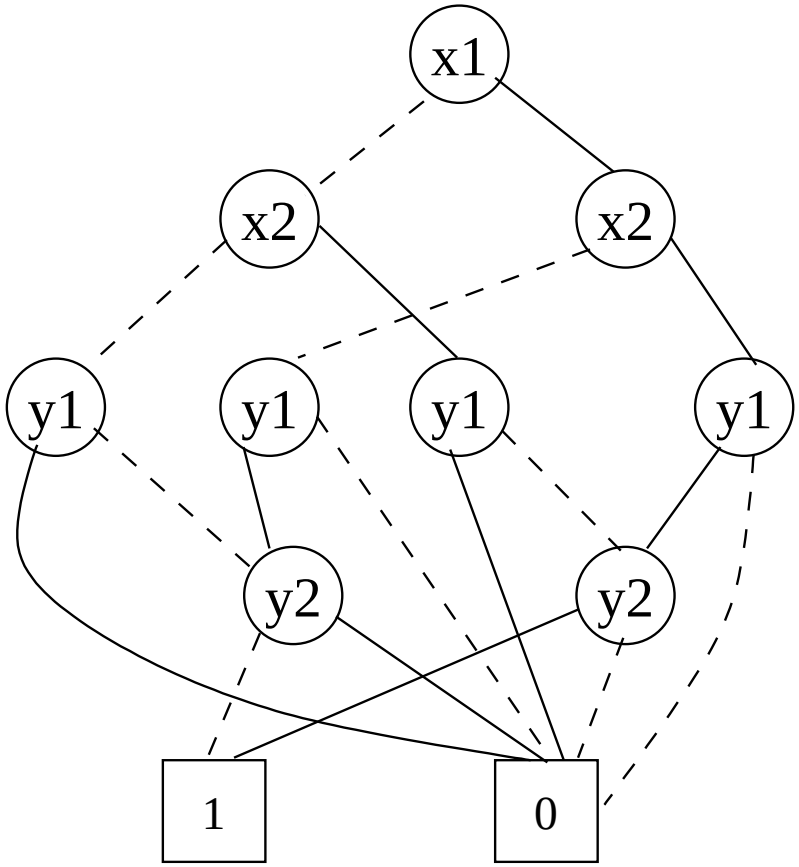
Consequences:

- b is a tautology if and only if $b = \boxed{1}$
- b is satisfiable if and only if $b \neq \boxed{0}$

But...

The size of ROBDD depends *significantly* on the chosen variable ordering!

Example: ROBDD for $(x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2)$ with ordering $x_1 < x_2 < y_1 < y_2$



Under ordering $x_1 < y_1 < x_2 < y_2$ had 6 nodes.

Furthermore...

- The size according to one ordering may be exponentially smaller than another ordering.
- Figuring out the optimal ordering of variables is co-NP-complete.
- Some functions have small size independent of ordering, e.g. parity.
- Some functions have large size independent of ordering, e.g., multiplication

Representing Boolean Functions

Representation of boolean functions	compact?	satisf'ty	validity
Prop. formulas	often	hard	hard
Formulas in DNF	sometimes	easy	hard
Formulas in CNF	sometimes	hard	easy
Ordered truth tables	never	hard	hard
Reduced OBDDs	often	easy	easy

Representation of boolean functions	$\wedge$	$\vee$	$\neg$
Prop. formulas	easy	easy	easy
Formulas in DNF	hard	easy	hard
Formulas in CNF	easy	hard	hard
Ordered truth tables	hard	hard	hard
Reduced OBDDs	medium	medium	easy

Symbolic model checking

Why?

Saves us from constructing a model’s state space explicitly. Effective ”cure” for state space explosion problem.

How?

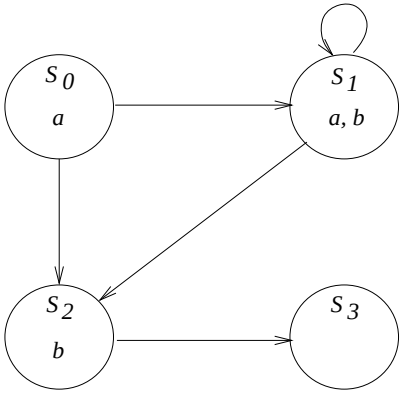
Sets of states and the transition relation are represented by formulas. Set operations are defined in terms of formula manipulations.

Data structures

ROBDDs - allow for efficient storage and manipulation of logic formulas.

Representing Models Symbolically

- A system state represents an interpretation (truth assignment) for a set of propositional variables V.
  - Formulas represent sets of states that satisfy it
    - False - $\emptyset$, True - S
    - a - set of states in which a is true - ($\{s_0, s_1\}$)
    - b - set of states in which b is true - ($\{s_1, s_2\}$)
    - $a \vee b = \{s_0, s_1\} \cup \{s_1, s_2\} = \{s_0, s_1, s_2\}$



- State transitions are described by relations over two sets of variables, V (source state) and V' (destination state)
  - Trans. from s_2 to s_3 is described by $(\neg a \wedge b \wedge \neg a' \wedge \neg b')$.
  - Trans. from s_0 to s_1 and s_2, and from s_1 to s_2 and to itself is described by $(a \wedge b')$.
  - Relation R is described by $(a \wedge b') \vee (\neg a \wedge b \wedge \neg a' \wedge \neg b')$

Symbolic Model-Checking Algorithm on BDDs

Procedure $\mathtt{MC}(p)$
Case

$p \in A$	:	**return** *Build*("p")
$p = \neg\varphi$	:	**return** *Apply*('$\neg$', MC(φ))
$p = \varphi \wedge \psi$	:	**return** *Apply*('$\wedge$', MC(φ), MC(ψ))
$p = \varphi \vee \psi$	:	**return** *Apply*('$\wedge$', MC(φ), MC(ψ))
$p = EX\varphi$	:	**return** *existQuantify*(V',
		Apply('$\wedge$', R, *Prime*(MC(φ)))
$p = AX\varphi$	:	**return** *Apply*('$\neg$', MC($EX\ \neg\varphi$))
$p = E[\varphi U \psi]$	:	Q_0 = *Build*('$\bot$')
		Q_{i+1} = *Apply*('$\vee$', Q_i, *Apply*('$\vee$', MC(ψ),
		Apply('$\vee$', MC($EX\ Q_i$)))
		return Q_n when $Q_n = Q_{n+1}$
$p = EG\varphi$	:	Q_0 = *Build*('$\top$')
		Q_{i+1} = *Apply*('$\wedge$', MC(φ), MC($EX\ Q_i$))))
		return Q_n when $Q_n = Q_{n+1}$

Pros and Cons of Model-Checking

- Often cannot express full requirements
  - Instead check several smaller properties
- Few systems can be checked directly
  - Must generally abstract
- Work better for certain types of problems
  - Very useful for control-centered concurrent systems
    - Avionics software
    - Hardware
    - Communication protocols
  - Not very good at data-centered systems
    - User interfaces, databases

Some State of the Art Model-Checkers

- SMV, NuSMV, Cadence SMV
  - CTL and LTL model-checkers
  - Based on symbolic decision diagrams or SAT solvers
  - Mostly for hardware
- Spin
  - LTL model-checker (automata-theoretic)
  - Explicit state exploration
  - Mostly for communication protocols
- HyTech, Kronos, Upaal
  - For real-time and hybrid systems
- STeP and PVS
  - Combining model-checking with theorem-proving

Software Model-Checking

- Goal: check programs without manually extracting models
- State of the art (explicit-state)
  - JavaPathfinder (NASA)
  - BOGOR (framework for building them, Kansas State)
  - Zing (Microsoft)
- Symbolic model-checkers
  - SLAM (Microsoft)
  - BLAST (Berkley)
  - Yasm (Toronto)