Lecture 16:

Computation Tree Logic (CTL)

Programme for the upcoming lectures

Introducing CTL

Basic Algorithms for CTL

CTL and Fairness; computing strongly connected components

Basic Decision Diagrams

Tool demonstration: SMV

LTL and CTL

LTL (linear-time logic)

- Describes properties of individual executions.
- Semantics defined as a set of executions.

CTL (computation tree logic)

- Describes properties of a *computation tree*: formulas can reason about many executions at once. (CTL belongs to the family of *branching-time logics*.)
- Semantics defined in terms of states.

Computation tree

Let $\mathcal{T} = \langle S, \rightarrow, s^0 \rangle$ be a transition system.
Intuitively, the computation tree of $\mathcal{T}$ is the acyclic unfolding of $\mathcal{T}$.

Formally, we can define the unfolding as the least (possibly infinite) transition system $\langle U, \rightarrow', u^0 \rangle$ with a labelling $l \colon U \rightarrow S$ such that

- $u^0 \in U$ and $l(u^0) = s^0$;

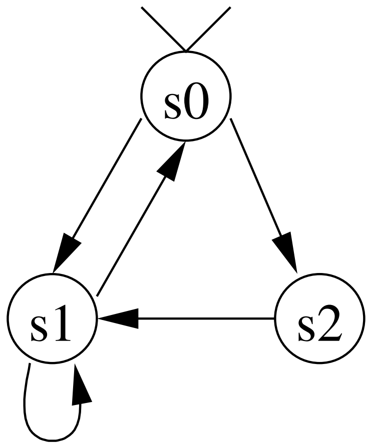
- if $u \in U$, $l(u) = s$, and $s \rightarrow s'$ for some u, s, s',
  then there is $u' \in U$ with $u \rightarrow' u'$ and $l(u') = s'$;

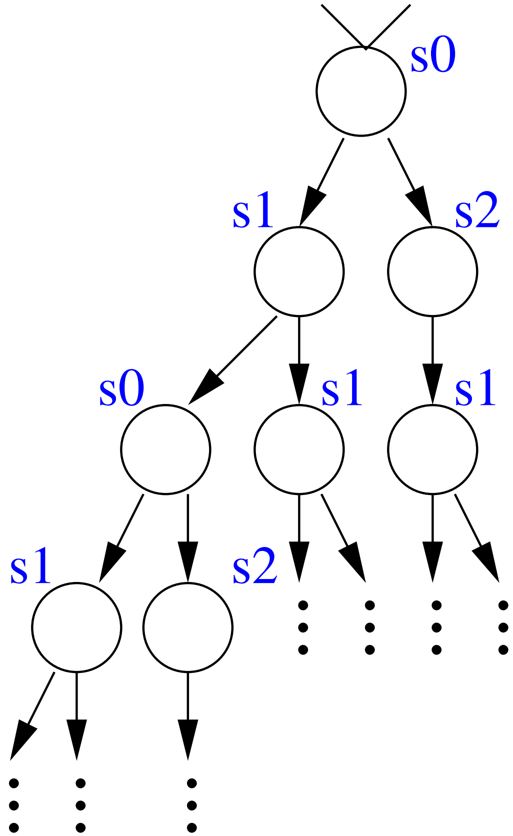
- u^0 does not have a direct predecessor, and all other states in U have exactly one direct predecessor.

Note: For model checking CTL, the construction of the computation tree will not be necessary. However, this definition serves to clarify the concepts behind CTL.

Computation tree: Example

A transition system and its computation tree (labelling *l* given in blue):





CTL: Overview

CTL = Computation-Tree Logic

Combines temporal operators with quantification over runs

Operators have the following form:

Q T

		X	**next**
there exists an execution	**E**	**F**	**finally**
for all executions	**A**	**G**	**globally**
		U	**until**

(and possibly others)

CTL: Syntax

We define a minimal syntax first. Later we define additional operators with the help of the minimal syntax.

Let *AP* be a set of atomic propositions: The set of CTL formulas over *AP* is as follows:

if $a \in AP$, then a is a CTL formula;

if ϕ_1, ϕ_2 are CTL formulas, then so are

$$\neg\phi_1, \qquad \phi_1 \vee \phi_2, \qquad \mathbf{EX}\,\phi_1, \qquad \mathbf{EG}\,\phi_1, \qquad \phi_1\,\mathbf{EU}\,\phi_2$$

CTL: Semantics

Let $\mathcal{K} = (S, \rightarrow, s^0, AP, \nu)$ be a Kripke structure.

We define the semantic of every CTL formula ϕ over AP w.r.t. $\mathcal{K}$ as a set of states $[\![\phi]\!]_{\mathcal{K}}$, as follows:

$$
\begin{array}{rcl}
[\![a]\!]_{\mathcal{K}} & = & \nu(a) \qquad \text{for } a \in AP \\
[\![\neg\phi_1]\!]_{\mathcal{K}} & = & S \setminus [\![\phi_1]\!]_{\mathcal{K}} \\
[\![\phi_1 \vee \phi_2]\!]_{\mathcal{K}} & = & [\![\phi_1]\!]_{\mathcal{K}} \cup [\![\phi_2]\!]_{\mathcal{K}} \\
[\![\mathbf{EX}\,\phi_1]\!]_{\mathcal{K}} & = & \{\, s \mid \text{there is a } t \text{ s.t. } s \rightarrow t \text{ and } t \in [\![\phi_1]\!]_{\mathcal{K}} \,\} \\
[\![\mathbf{EG}\,\phi_1]\!]_{\mathcal{K}} & = & \{\, s \mid \text{there is a run } \rho \text{ with } \rho(0) = s \\
 & & \quad \text{and } \rho(i) \in [\![\phi_1]\!]_{\mathcal{K}} \text{ for all } i \geq 0 \,\} \\
[\![\phi_1\,\mathbf{EU}\,\phi_2]\!]_{\mathcal{K}} & = & \{\, s \mid \text{there is a run } \rho \text{ with } \rho(0) = s \text{ and } k \geq 0 \text{ s.t.} \\
 & & \quad \rho(i) \in [\![\phi_1]\!]_{\mathcal{K}} \text{ for all } i < k \text{ and } \rho(k) \in [\![\phi_2]\!]_{\mathcal{K}} \,\}
\end{array}
$$

We say that $\mathcal{K}$ satisfies ϕ (denoted $\mathcal{K} \models \phi$) iff $s^0 \in [\![\phi]\!]_{\mathcal{K}}$.

We declare two formulas equivalent (written $\phi_1 \equiv \phi_2$) iff for every Kripke structure $\mathcal{K}$ we have $[\![\phi_1]\!]_{\mathcal{K}} = [\![\phi_2]\!]_{\mathcal{K}}$.

In the following, we omit the index $\mathcal{K}$ from $[\![\cdot]\!]_{\mathcal{K}}$ if $\mathcal{K}$ is understood.

CTL: Extended syntax

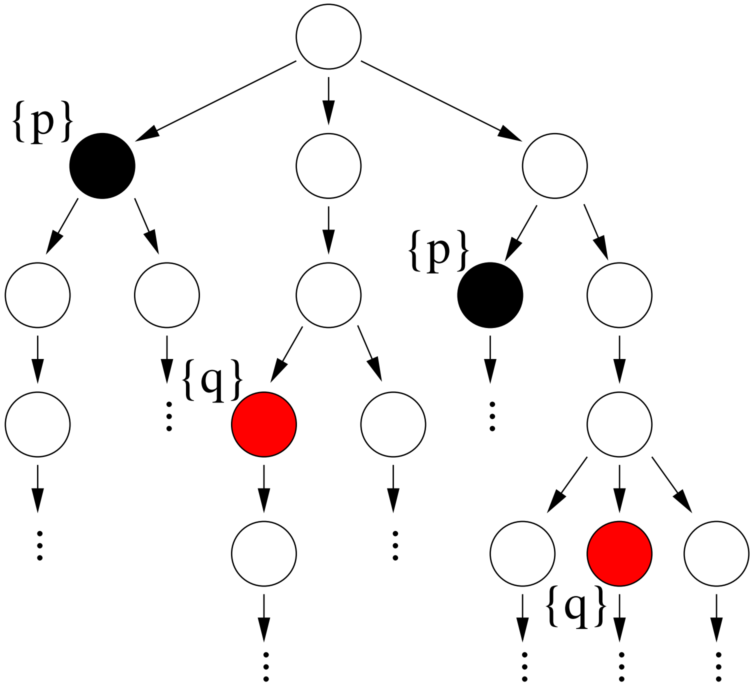
$$\begin{aligned}
\phi_1 \wedge \phi_2 &\equiv \neg(\neg\phi_1 \vee \neg\phi_2) \\
\mathbf{true} &\equiv a \vee \neg a \\
\mathbf{false} &\equiv \neg\mathbf{true} \\
\phi_1\,\mathbf{EW}\,\phi_2 &\equiv \mathbf{EG}\,\phi_1 \vee (\phi_1\,\mathbf{EU}\,\phi_2) \\
\mathbf{EF}\,\phi &\equiv \mathbf{true}\,\mathbf{EU}\,\phi
\end{aligned}$$

$$\begin{aligned}
\mathbf{AX}\,\phi &\equiv \neg\,\mathbf{EX}\,\neg\phi \\
\mathbf{AG}\,\phi &\equiv \neg\,\mathbf{EF}\,\neg\phi \\
\mathbf{AF}\,\phi &\equiv \neg\,\mathbf{EG}\,\neg\phi \\
\phi_1\,\mathbf{AW}\,\phi_2 &\equiv \neg(\neg\phi_2\,\mathbf{EU}\,\neg(\phi_1 \vee \phi_2)) \\
\phi_1\,\mathbf{AU}\,\phi_2 &\equiv \mathbf{AF}\,\phi_2 \wedge (\phi_1\,\mathbf{AW}\,\phi_2)
\end{aligned}$$

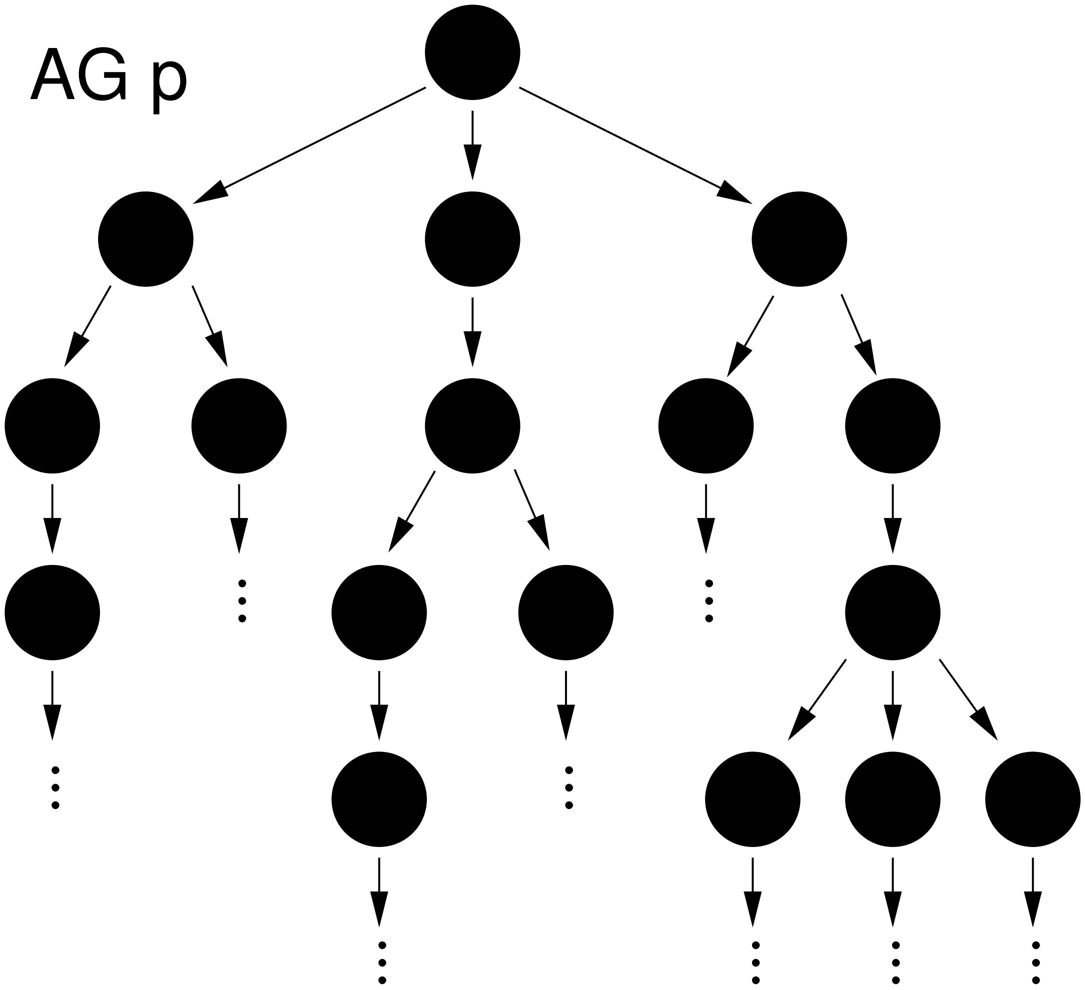
Other logical and temporal operators (e.g. $\longrightarrow$), $\mathbf{ER}$, $\mathbf{AR}$, ... may also be defined.

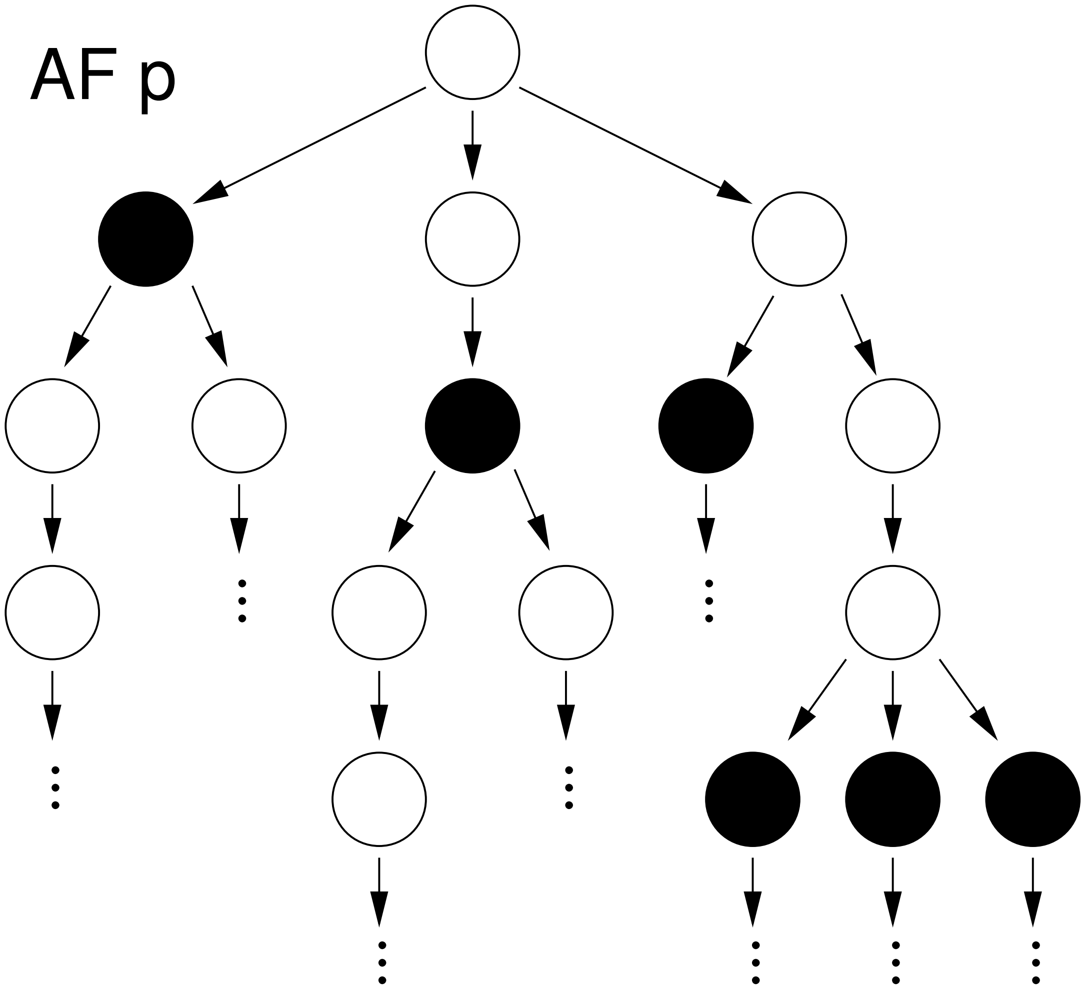
CTL: Examples

We use the following computation tree as a running example (with varying distributions of red and black states):

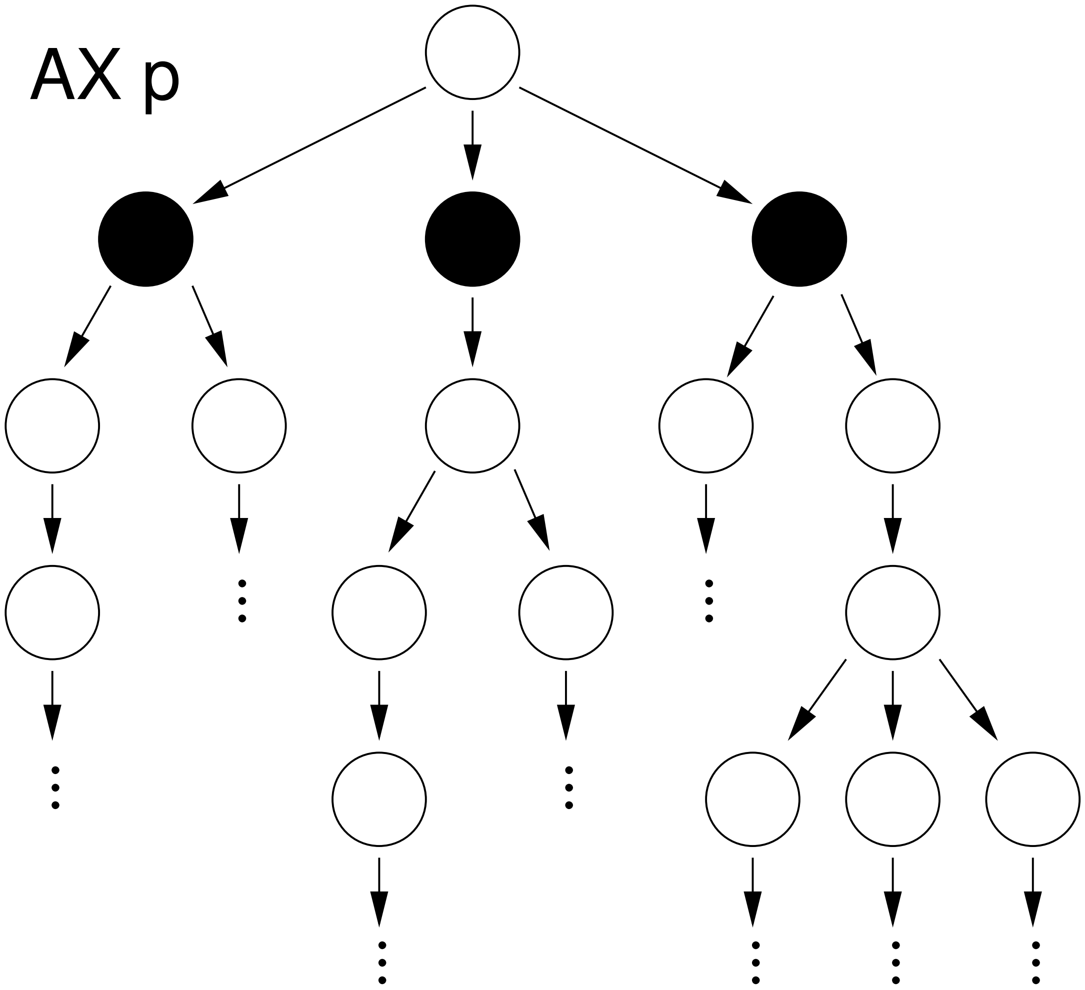


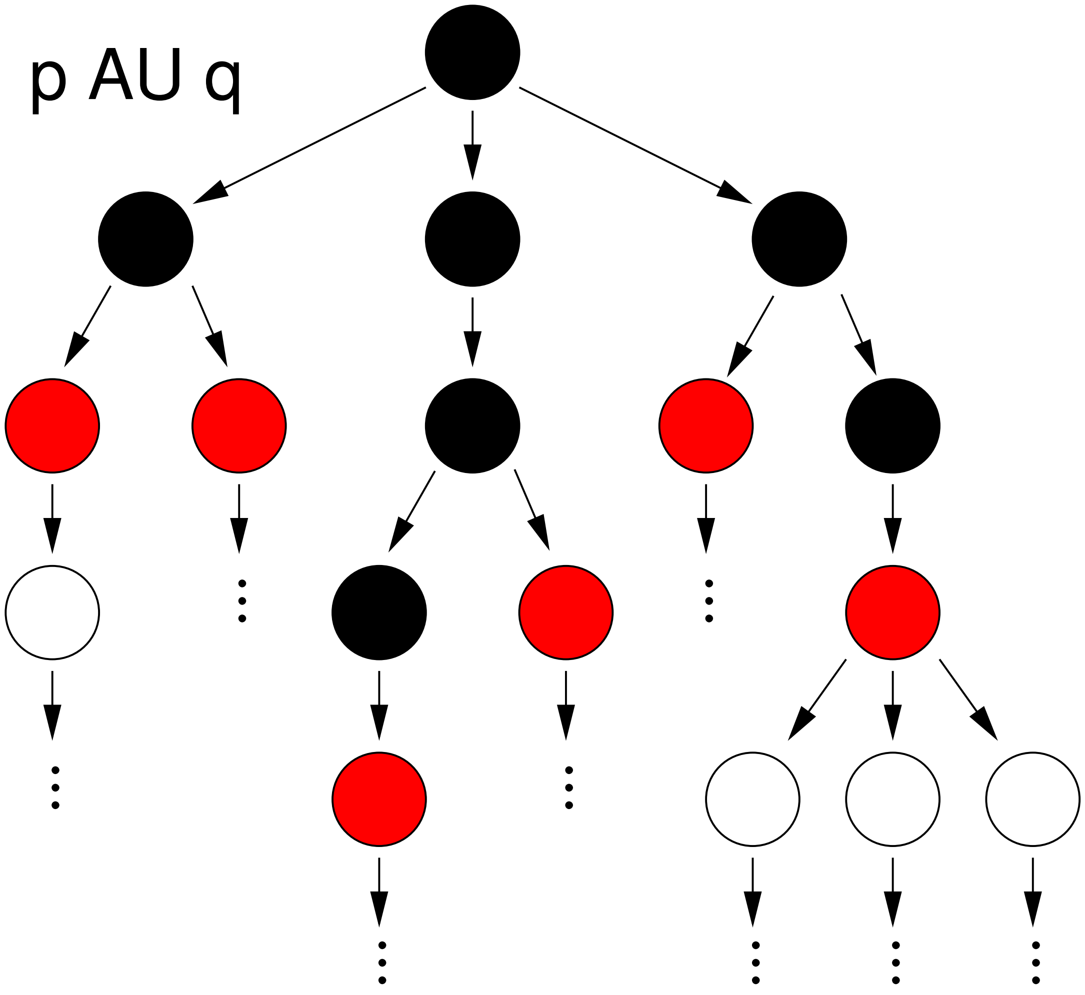
In the following slides, the topmost state satisfies the given formula if the black states satisfy *p* and the red states satisfy *q*.


AG p

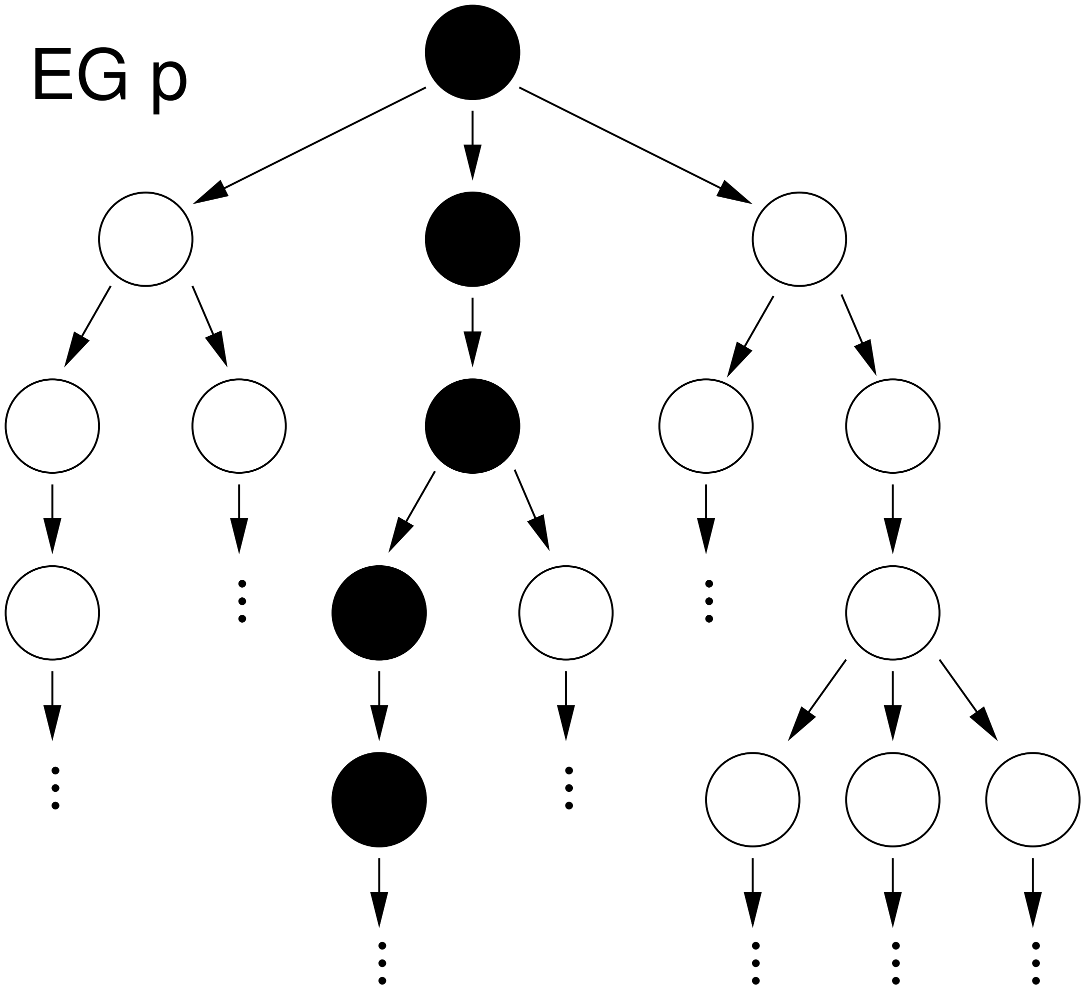

AF p

AX p

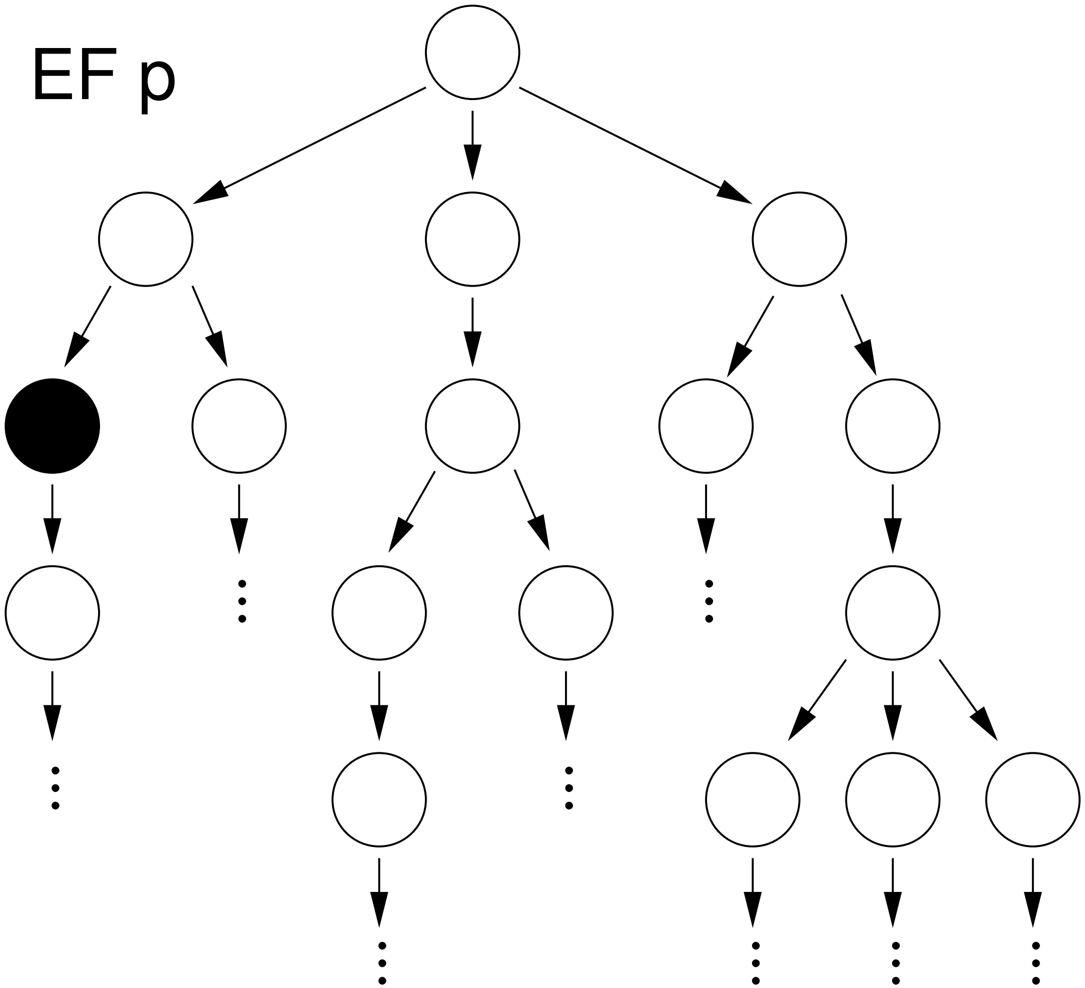



p AU q

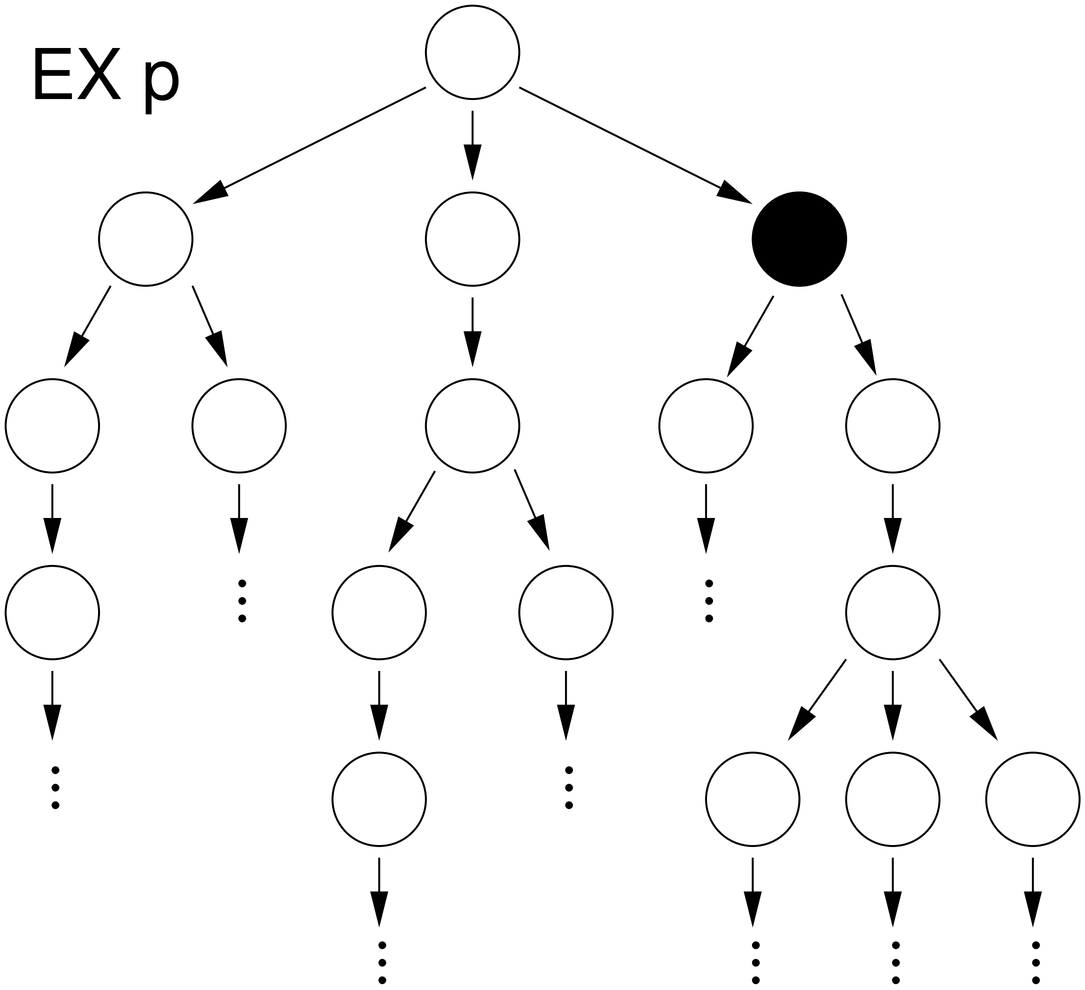
EG p



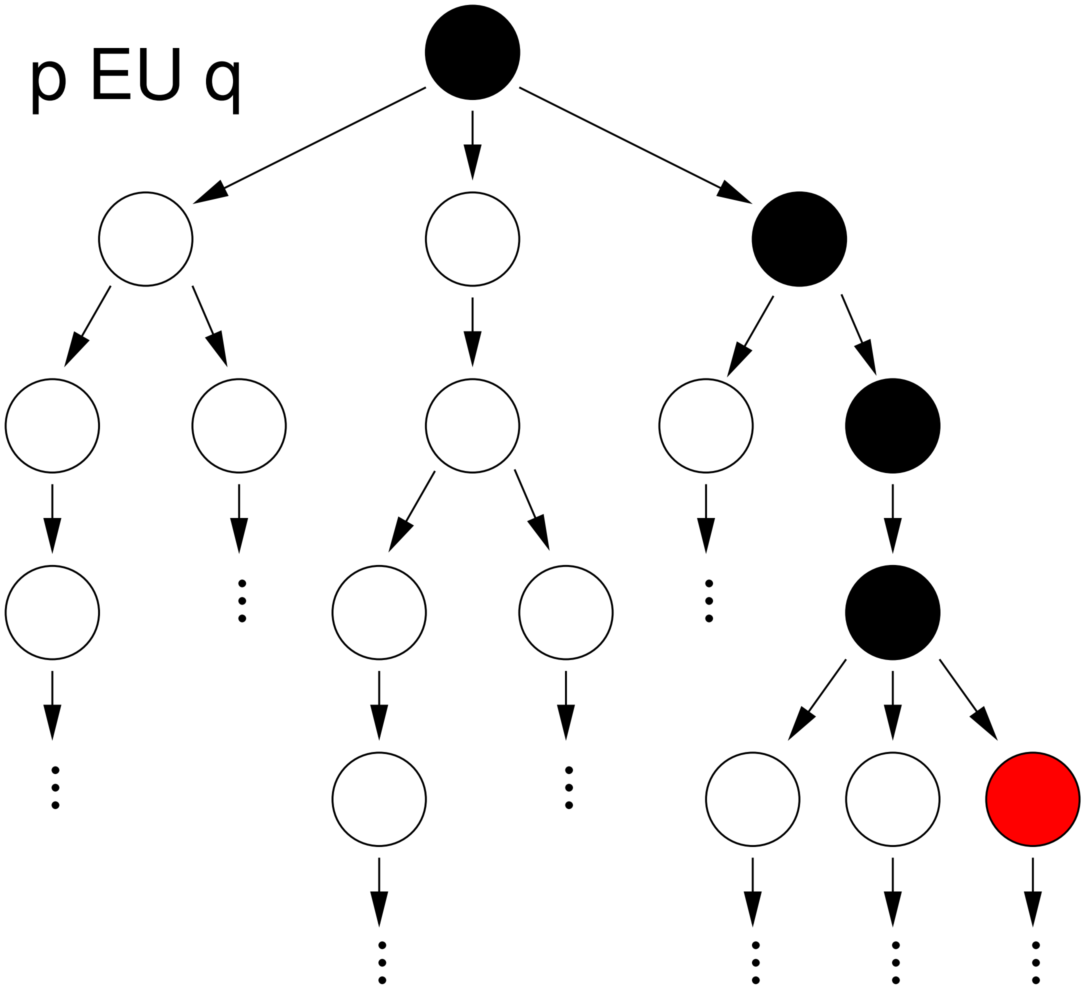
EF p



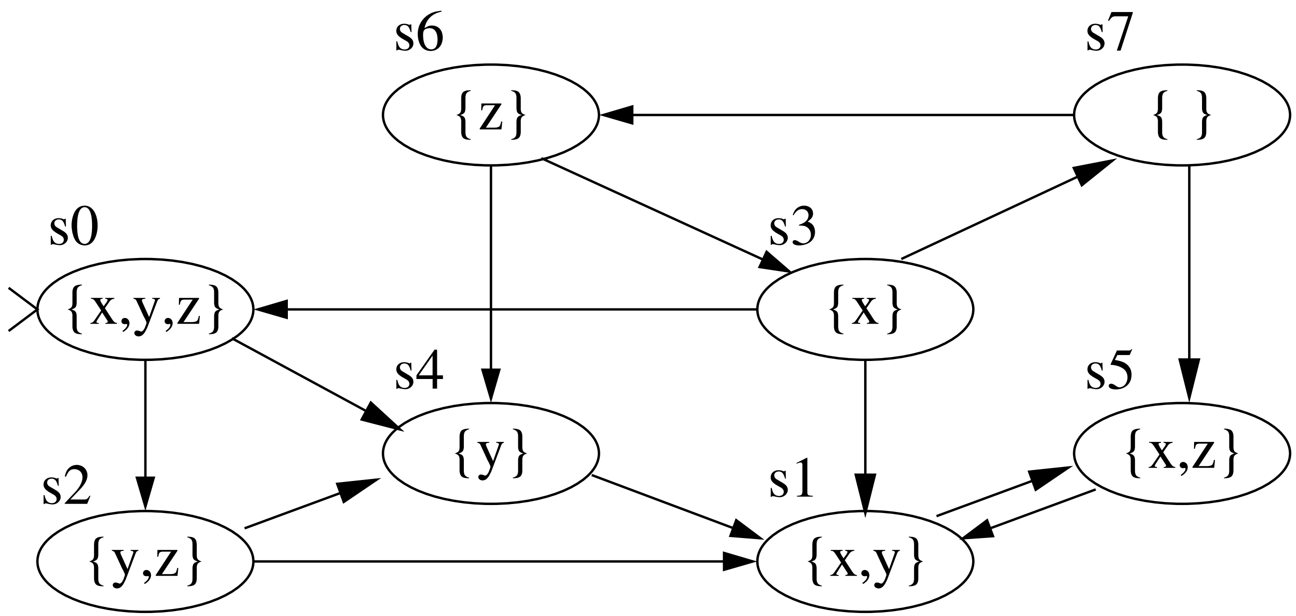
EX p



p EU q



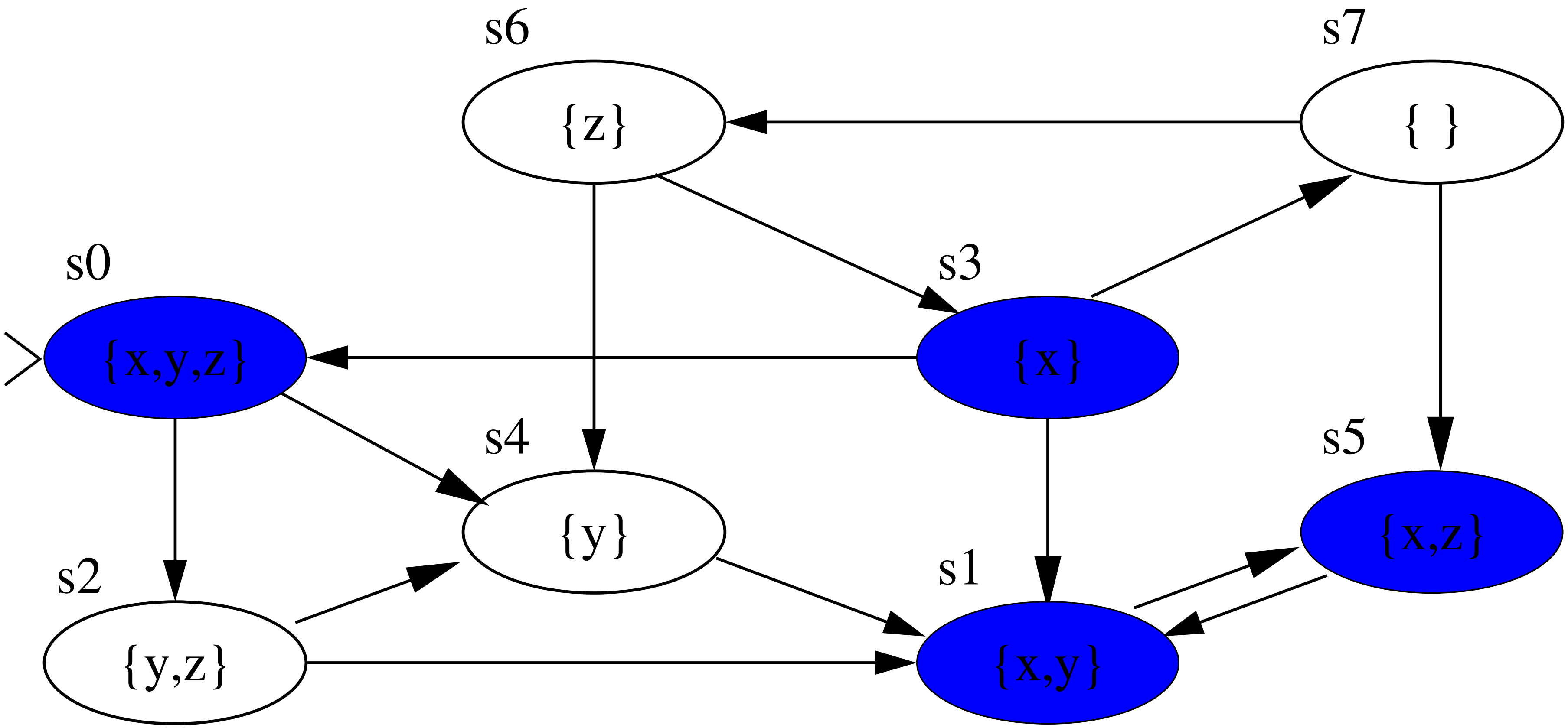
Solving nested formulas: Is $s_0 \in [\![\mathbf{AF}\ \mathbf{AG}\, x]\!]$?



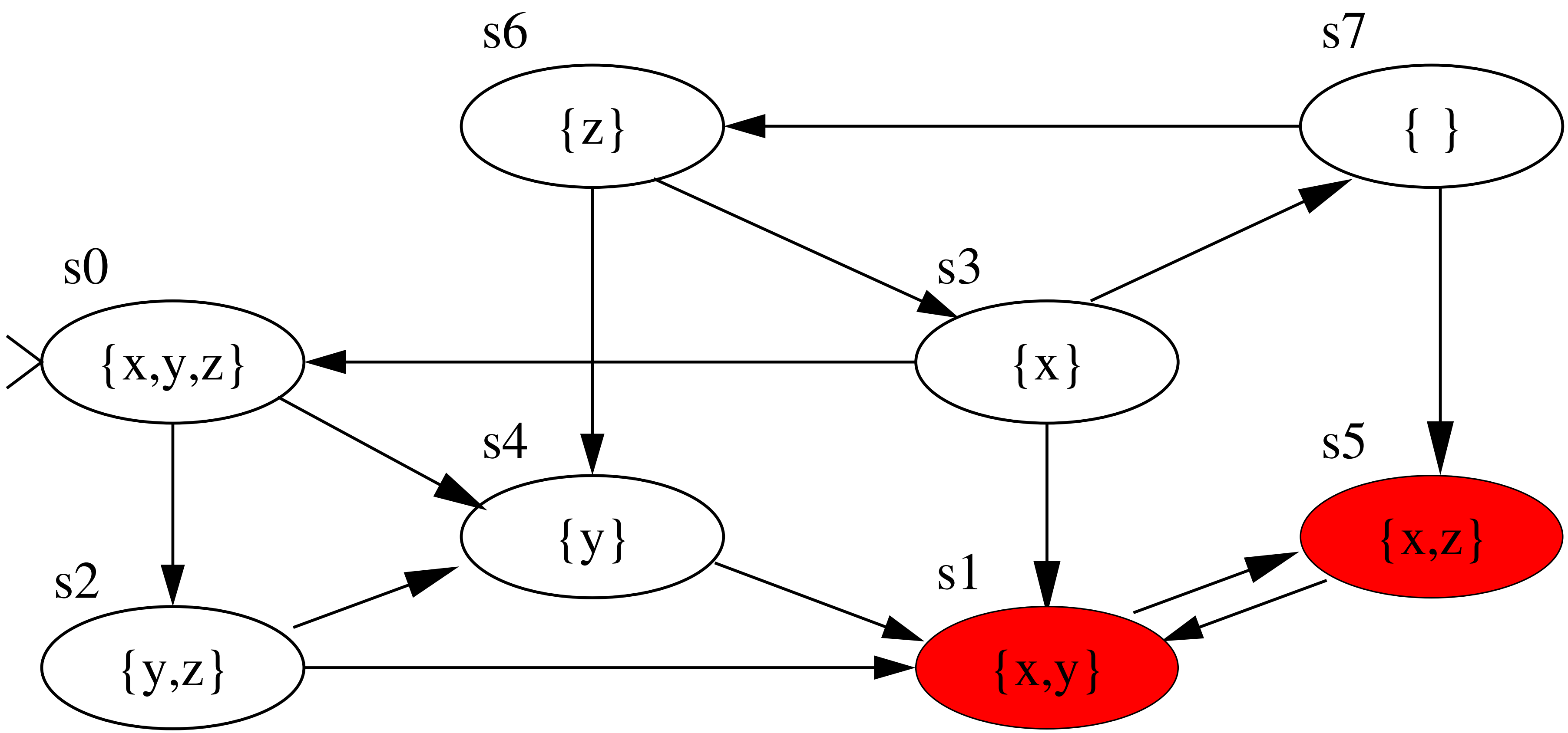
To compute the semantics of formulas with nested operators, we first compute the states satisfying the innermost formulas; then we use those results to solve progressively more complex formulas.

In this example, we compute $[\![x]\!]$, $[\![\mathbf{AG}\, x]\!]$, and $[\![\mathbf{AF}\ \mathbf{AG}\, x]\!]$, in that order.

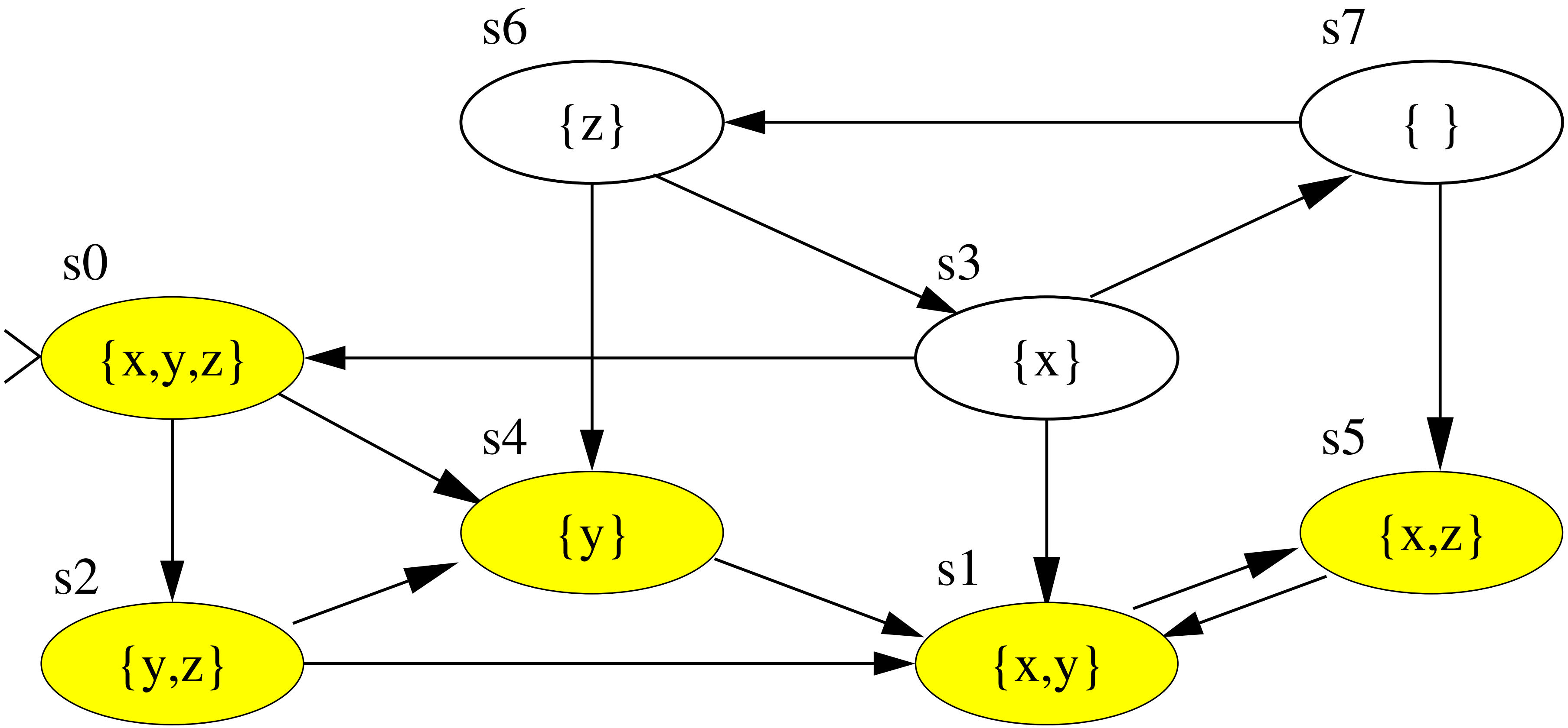
Bottom-up method (1): Compute $[\![x]\!]$



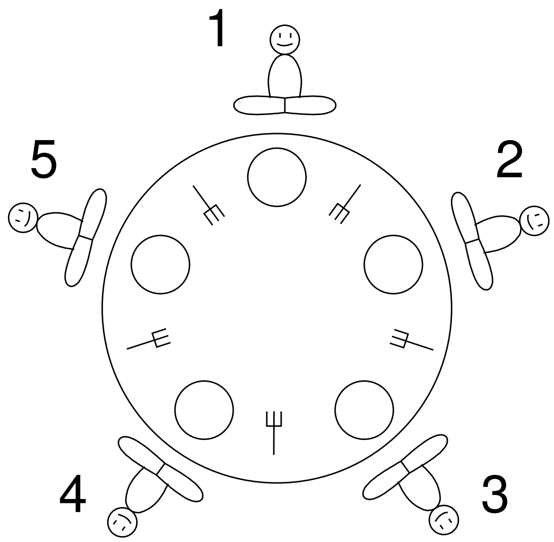
Bottom-up method (2): Compute $[\![\mathbf{AG}\,x]\!]$



Bottom-up method (3): Compute $[\![\mathbf{AF}\ \mathbf{AG}\ x]\!]$



Example: Dining Philosophers



Five philosophers are sitting around a table, taking turns at thinking and eating.

We shall express a couple of properties in CTL. Let us assume the following atomic propositions:

$e_i \;\widehat{=}\;$ philosopher i is currently eating

$f_i \;\widehat{=}\;$ philosopher i has just finished eating

"Philosophers 1 and 4 will never eat at the same time."

"Philosophers 1 and 4 will never eat at the same time."

$$\mathbf{AG}\,\neg(e_1 \wedge e_4)$$

"Whenever philosopher 4 has finished eating, he cannot eat again until philosopher 3 has eaten."

“Philosophers 1 and 4 will never eat at the same time.”

$$\mathbf{AG}\,\neg(e_1 \wedge e_4)$$

“Whenever philosopher 4 has finished eating, he cannot eat again until philosopher 3 has eaten.”

$$\mathbf{AG}(f_4 \rightarrow (\neg e_4\ \mathbf{AW}\ e_3))$$

“Philosopher 2 will be the first to eat.”

"Philosophers 1 and 4 will never eat at the same time."

$$\mathbf{AG}\,\neg(e_1 \wedge e_4)$$

"Whenever philosopher 4 has finished eating, he cannot eat again until philosopher 3 has eaten."

$$\mathbf{AG}(f_4 \rightarrow (\neg e_4\ \mathbf{AW}\ e_3))$$

"Philosopher 2 will be the first to eat."

$$\neg(e_1 \vee e_3 \vee e_4 \vee e_5)\ \mathbf{AU}\ e_2$$

Expressiveness of CTL and LTL (1/4)

CTL and LTL have a large overlap, i.e. properties expressible in both logics.
Examples:

Invariants (e.g., "p never holds.")

$$\mathbf{AG}\,\neg p \qquad \text{or} \qquad G\neg p$$

Reactivity ("Whenever p happens, eventually q will happen.")

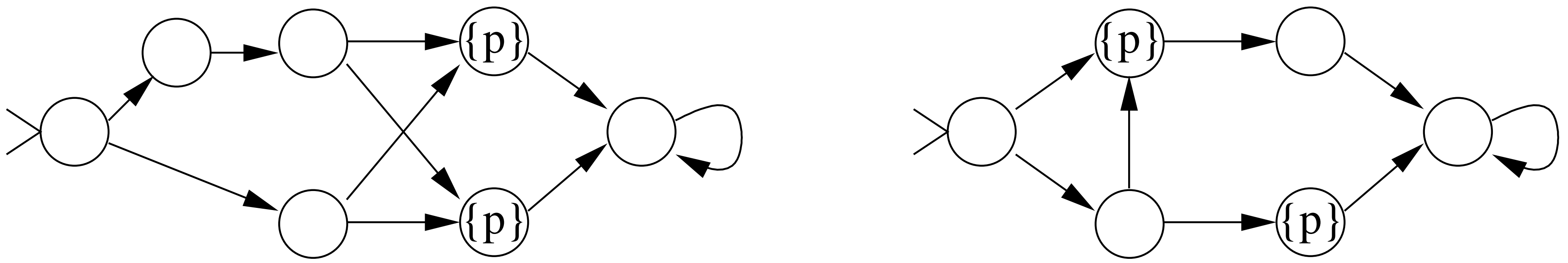
$$\mathbf{AG}(p \rightarrow \mathbf{AF}\,q) \qquad \text{or} \qquad \mathbf{G}(p \rightarrow \mathbf{F}\,q)$$

Expressiveness of CTL and LTL (2/4)

CTL considers the whole computation tree whereas LTL only considers individual runs. Thus CTL allows to reason about the *branching behaviour*, considering multiple possible runs at once. Examples:

The CTL property $\mathbf{AG}\,\mathbf{EF}\,p$ (“reset property”) is not expressible in LTL.

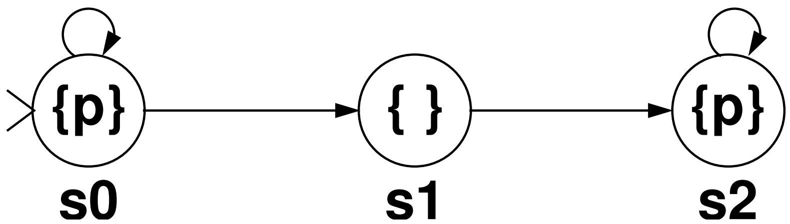
The CTL property $\mathbf{AF}\,\mathbf{AX}\,p$ distinguishes the following two systems, but the LTL property $\mathbf{F}\,\mathbf{X}\,p$ does not:



Expressiveness of CTL and LTL (3/4)

Even though CTL considers the whole computation tree, its state-based semantics is subtly different from LTL. Thus, there are also properties expressible in LTL but not in CTL. Examples:

The LTL property $\mathbf{F}\,\mathbf{G}\,p$ is not expressible in CTL:



$$\mathcal{K} \models \mathbf{F}\,\mathbf{G}\,p \quad \text{but} \quad \mathcal{K} \not\models \mathbf{AF}\,\mathbf{AG}\,p$$

Expressiveness of CTL and LTL (4/4)

Also, fairness conditions are not directly expressible in CTL:

$$(\mathbf{G}\,\mathbf{F}\,p_1 \wedge \mathbf{G}\,\mathbf{F}\,p_2) \rightarrow \phi$$

However, as we shall see later, there is another way to extend CTL with fairness conditions.

Conclusion: The expressiveness of CTL and LTL is incomparable; there is an overlap, and each logic can express properties that the other cannot.

Remark: There is a logic called CTL* that combines the expressiveness of CTL and LTL. However, we will not deal with it in this course.