

Computation Tree Logic (CTL)

Didier Buchs
University of Geneva
from S. Schwoon course uni-stuttgart

Programme for the upcoming lectures

- Introducing CTL
- Basic Algorithms for CTL
- Basic Decision Diagrams

BDD

Binary Decision
Diagram

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.

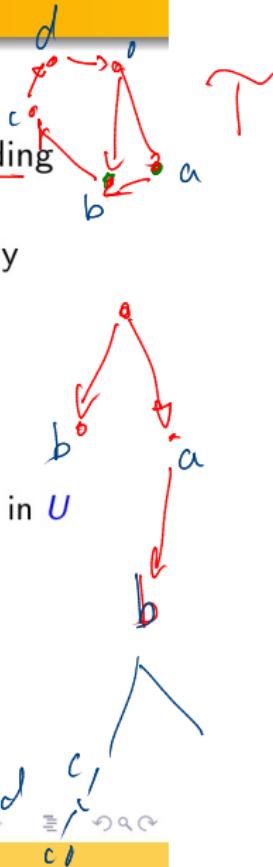
Computation tree

- 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} .

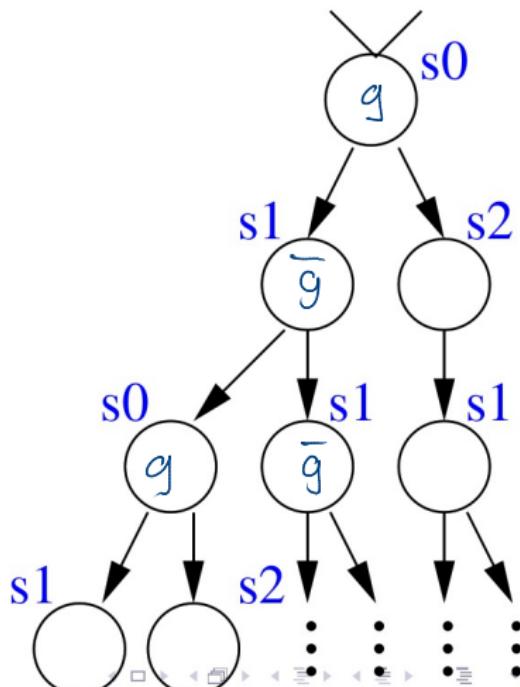
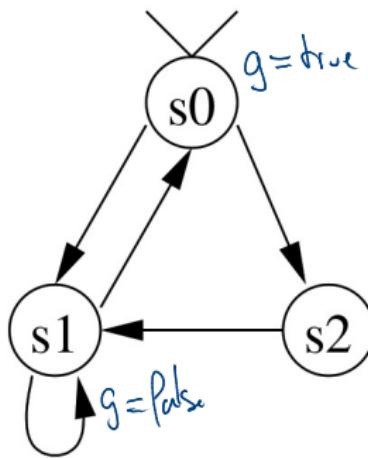
- Formally, we can define the unfolding as the least (possibly infinite) transition system $\langle U, \rightarrow', u^0 \rangle$ with a labelling $I: U \rightarrow S$ such that
 - $u^0 \in U$ and $I(u^0) = s^0$;
 - if $u \in U$, $I(u) = s$, and $s \rightarrow s'$ for some u, s, s' , then there is $u' \in U$ with $u \rightarrow' u'$ and $I(u') = s'$;
 - 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

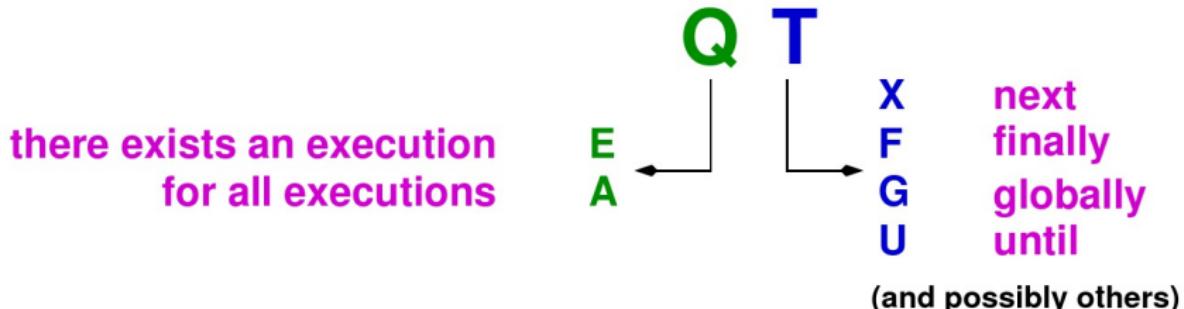
- A transition system and its computation tree (labelling / given in blue) :



CTL : Overview

- $CTL = \text{Computation-Tree Logic}$
- Combines temporal operators with quantification over runs

Operators have the following form :



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
 $\neg\phi_1$, $\underline{\phi_1 \vee \phi_2}$, $\underline{EX} \phi_1$, $\textcolor{magenta}{EG} \phi_1$, $\phi_1 \textcolor{blue}{EU} \phi_2$

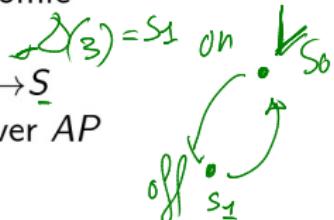
Exemples of expressions :

P, q

$EX (P \vee \exists (EG q))$

CTL : Semantics

- Let $\mathcal{K} = (S, \rightarrow, s^0, AP, \nu)$ be a Kripke structure. AP is the set of atomic proposition and $\nu : AP \rightarrow P(S)$ assign atomic propositions to states.
 - Runs are path within the transition system : $\rho : \mathbb{N} \rightarrow S$
 - 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 :



$$\text{bague} \quad \vdash [a]_{\mathcal{K}} = \nu(a) \quad \text{for } a \in AP$$

on \Rightarrow AF(off)
off \Rightarrow AF(on)

$$\text{classification} \quad [\neg \phi_1]_{\mathcal{K}} = S \setminus [\phi_1]_{\mathcal{K}}$$

$$[\phi_1 \vee \phi_2]_{\mathcal{K}} = [\phi_1]_{\mathcal{K}} \cup [\phi_2]_{\mathcal{K}}$$

off \Rightarrow AF (on)

$$[\![\text{EX } \phi_1]\!]_{\mathcal{K}} = \{ s \mid \text{there is a } t \text{ s.t. } s \xrightarrow{*} t \text{ and } t \in [\![\phi_1]\!]_{\mathcal{K}} \}$$

$$[\text{EG } \phi_1]_{\mathcal{K}} = \{ s \mid \text{there is a run } \underline{\rho} \text{ with } \underline{\rho}(0) = \underline{s} \text{ and } \underline{\rho}(i) \in [\phi_1]_{\mathcal{K}} \text{ for all } i \geq 0 \}$$

$$\llbracket \phi_1 \text{ EU } \phi_2 \rrbracket_{\mathcal{K}} = \{ s \mid \text{there is a run } \rho \text{ with } \rho(0) = s \text{ and } \exists k \geq 0 \text{ s.t.} \\ \rho(i) \in \llbracket \phi_1 \rrbracket_{\mathcal{K}} \text{ for all } i < k \text{ and } \rho(k) \in \llbracket \phi_2 \rrbracket_{\mathcal{K}} \}$$



Remarks

- We say that \mathcal{K} satisfies ϕ (denoted $\mathcal{K} \models \phi$) iff $s^0 \in \llbracket \phi \rrbracket_{\mathcal{K}}$.
- We declare two formulas equivalent (written $\phi_1 \equiv \phi_2$) iff for every Kripke structure \mathcal{K} we have $\llbracket \phi_1 \rrbracket_{\mathcal{K}} = \llbracket \phi_2 \rrbracket_{\mathcal{K}}$.
- In the following, we omit the index \mathcal{K} from $\llbracket \cdot \rrbracket_{\mathcal{K}}$ if \mathcal{K} is understood.

CTL : Extended syntax

$$\phi_1 \wedge \phi_2 \equiv \neg(\neg\phi_1 \vee \neg\phi_2)$$

$$\mathbf{AX} \phi \equiv \neg \mathbf{EX} \neg\phi$$

$$\mathbf{true} \equiv a \vee \neg a$$

$$\mathbf{AG} \phi \equiv \neg \mathbf{EF} \neg\phi$$

$$\mathbf{false} \equiv \neg \mathbf{true}$$

$$\mathbf{AF} \phi \equiv \neg \mathbf{EG} \neg\phi$$

$$\phi_1 \mathbf{EW} \phi_2 \equiv \mathbf{EG} \phi_1 \vee (\phi_1 \mathbf{EU} \phi_2)$$

$$\phi_1 \mathbf{AW} \phi_2 \equiv \neg(\neg\phi_2 \mathbf{EU} \neg(\phi_1 \vee \phi_2))$$

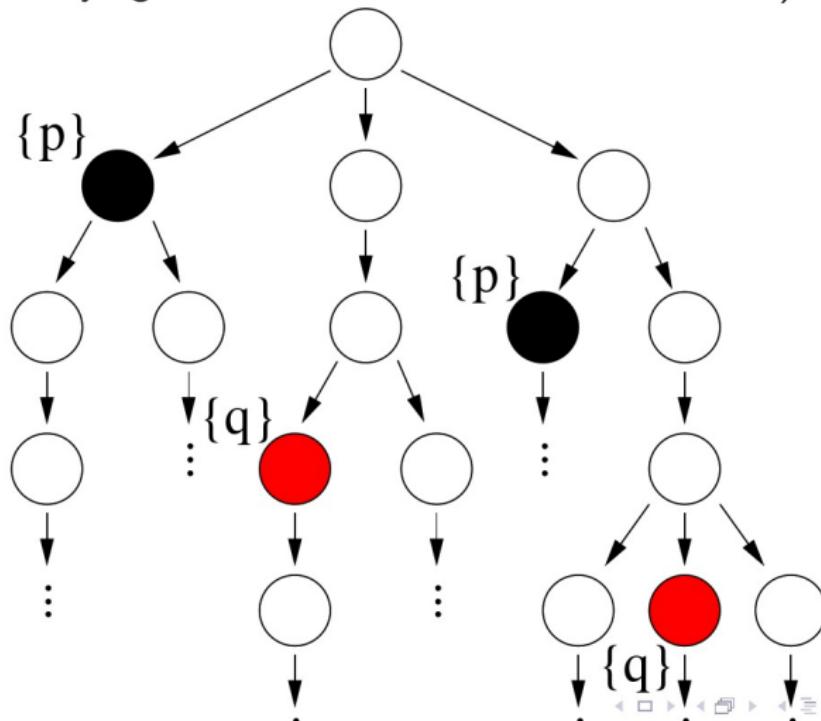
$$\mathbf{EF} \phi \equiv \mathbf{true} \mathbf{EU} \phi$$

$$\phi_1 \mathbf{AU} \phi_2 \equiv \mathbf{AF} \phi_2 \wedge (\phi_1 \mathbf{AW} \phi_2)$$

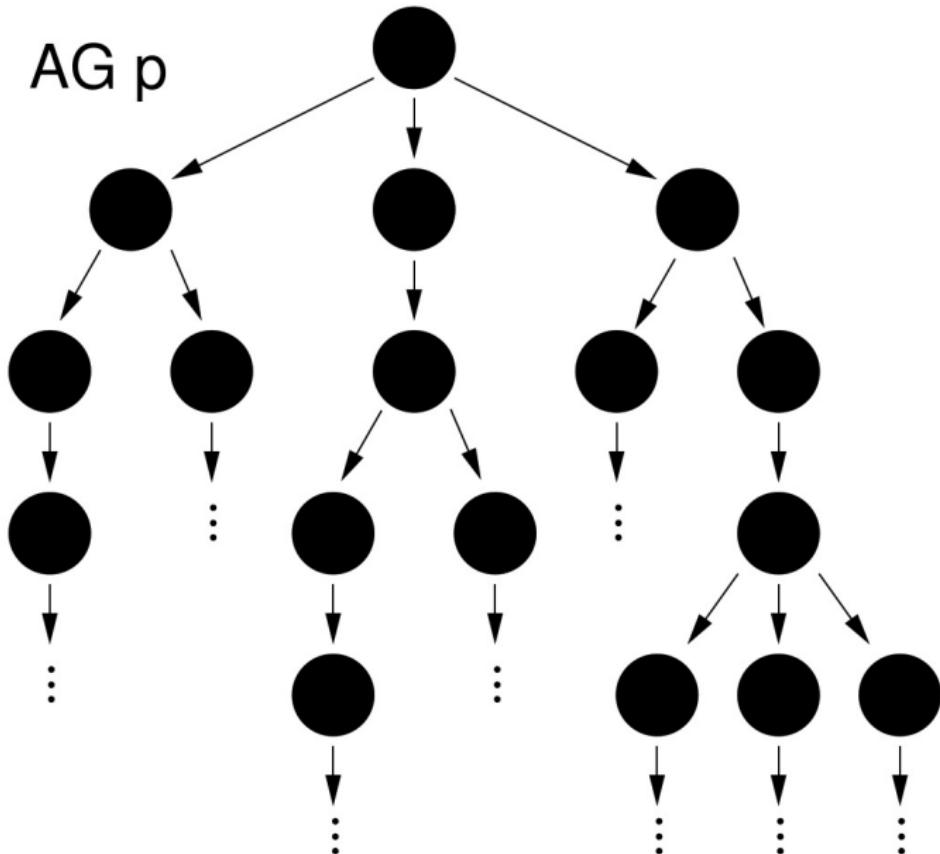
Other logical and temporal operators (e.g. \rightarrow), **ER**, **AR**, ... may also be defined.

CTL : Examples

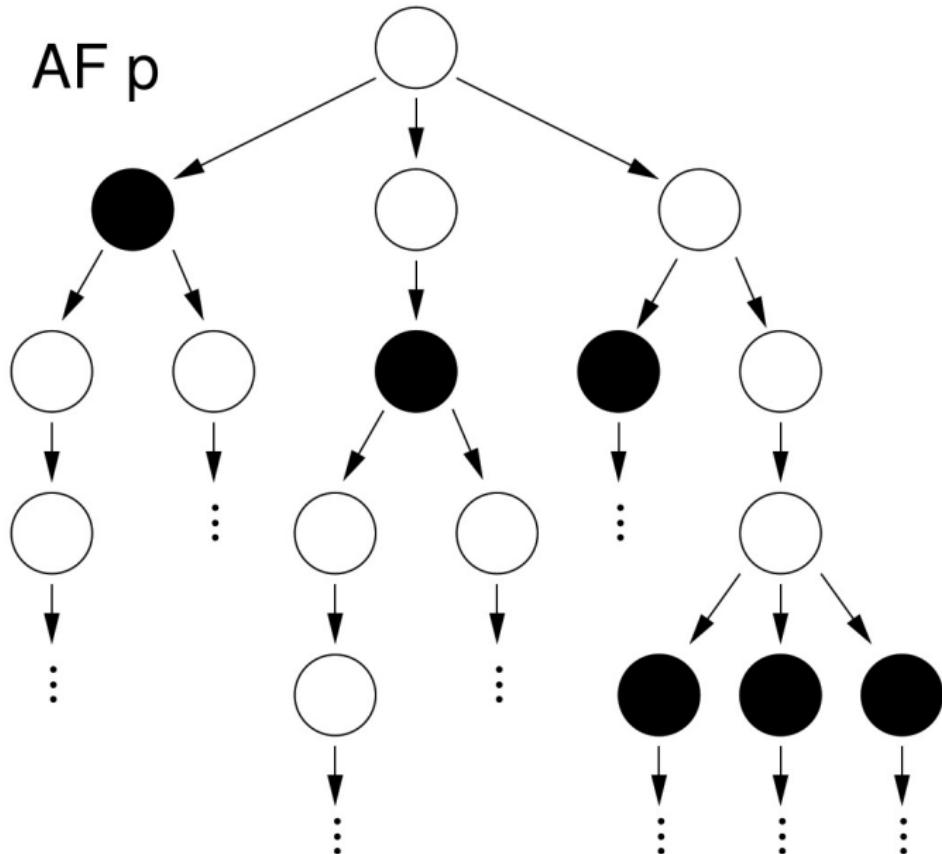
- We use the following computation tree as a running example (with varying distributions of red and black states) :

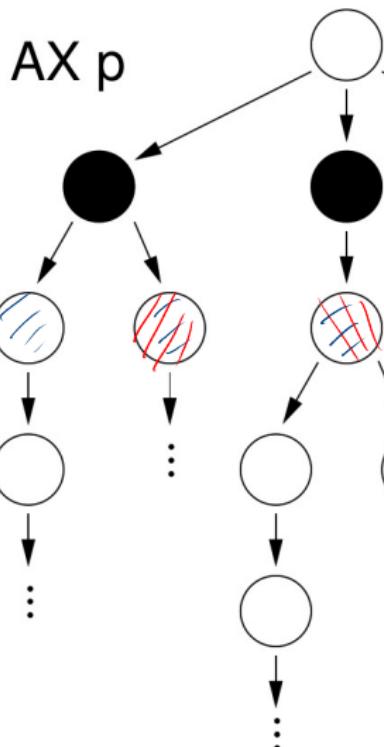


AG p



AF p



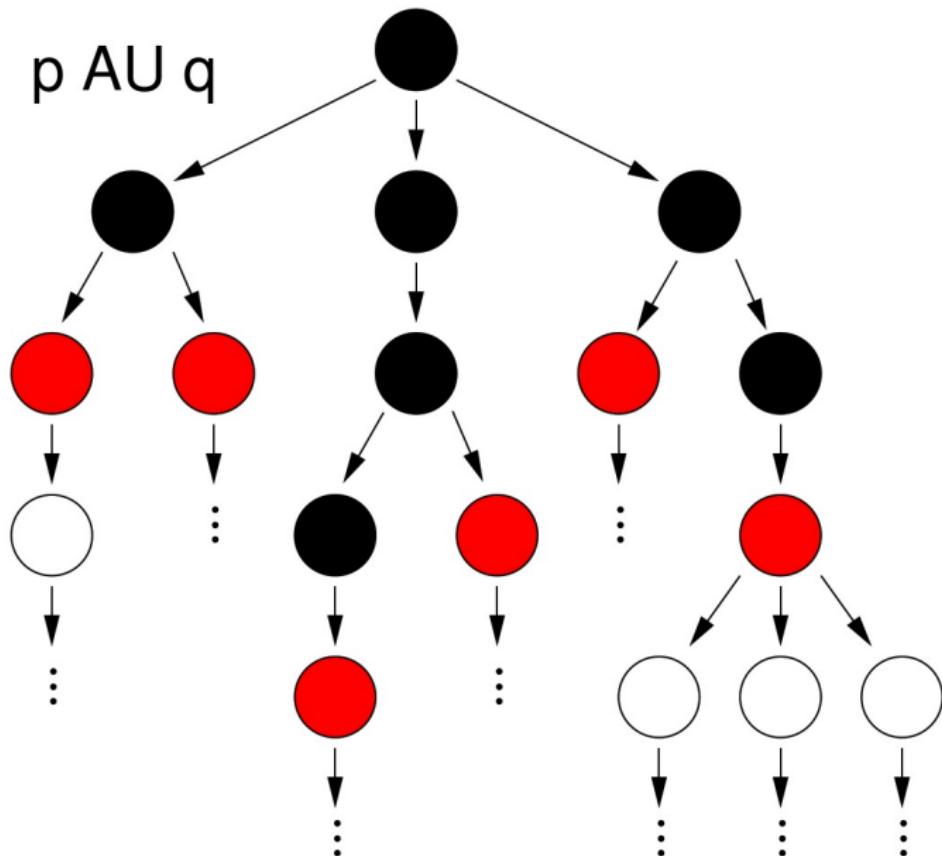


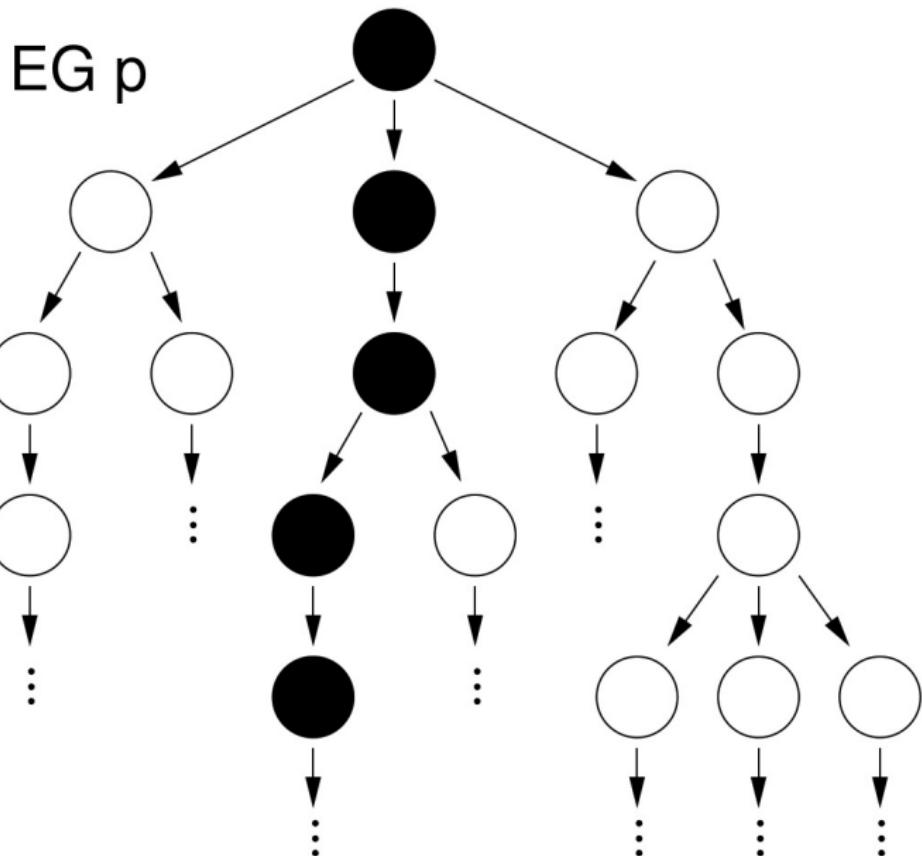
$$\text{AX}(\text{AX}(p)) = \emptyset$$

$$\text{AX}(\text{EX } q)$$

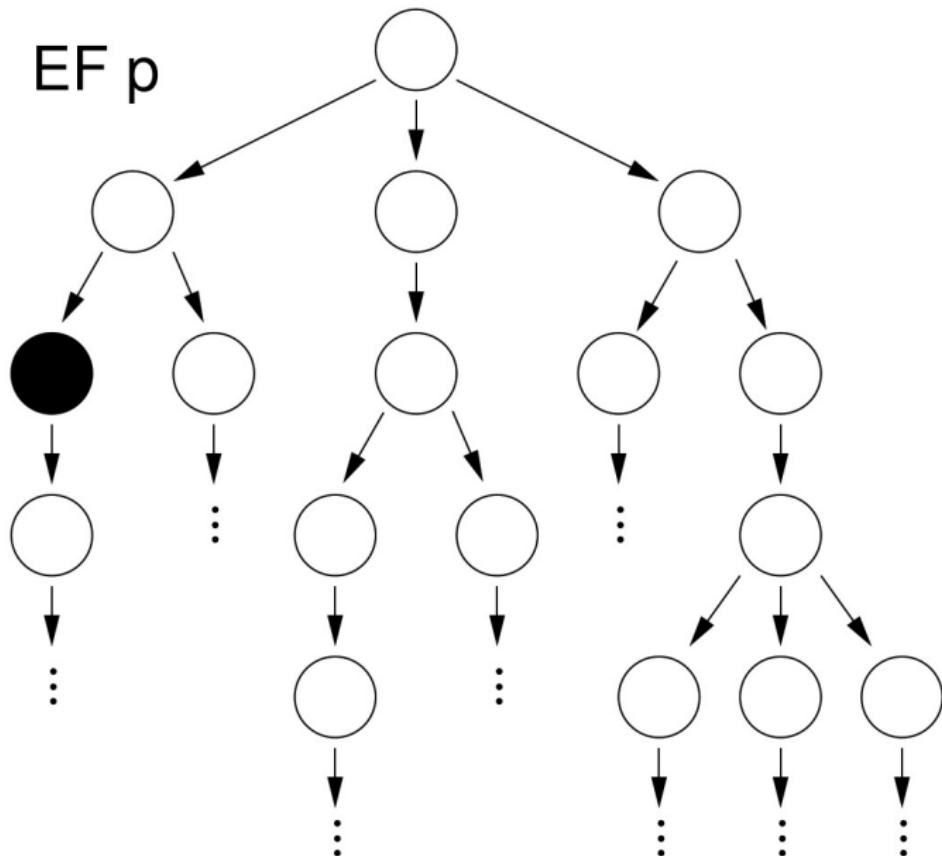
$$\text{EX}(\text{AX } q) \quad w$$

$p \text{ AU } q$

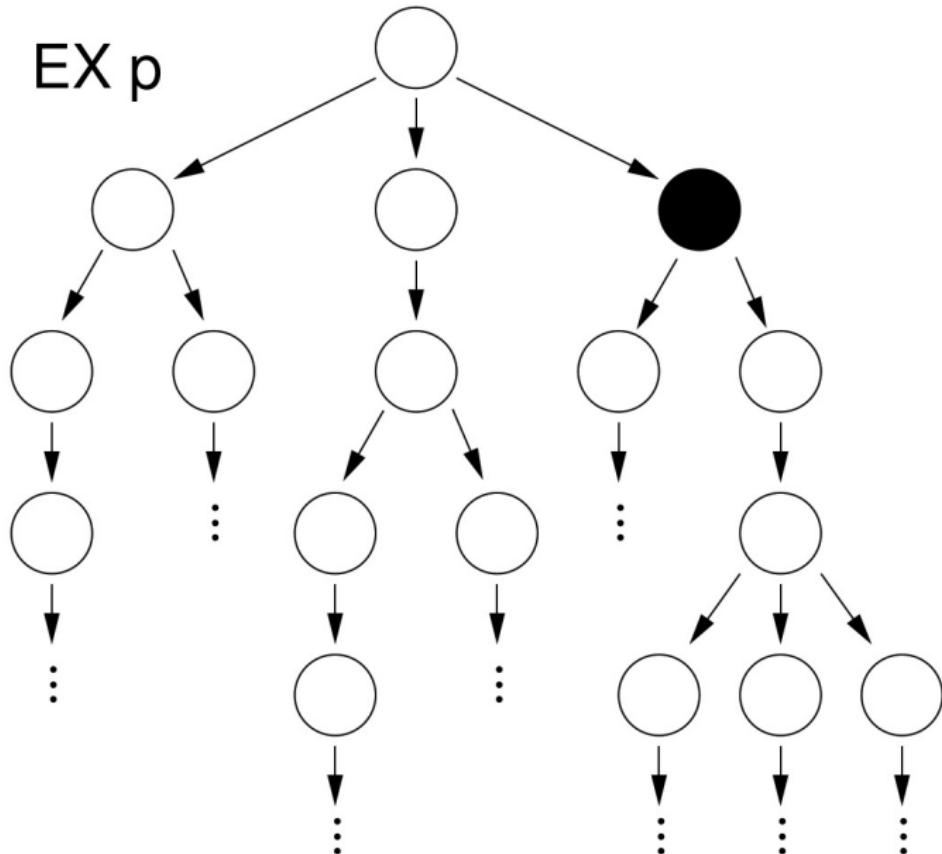




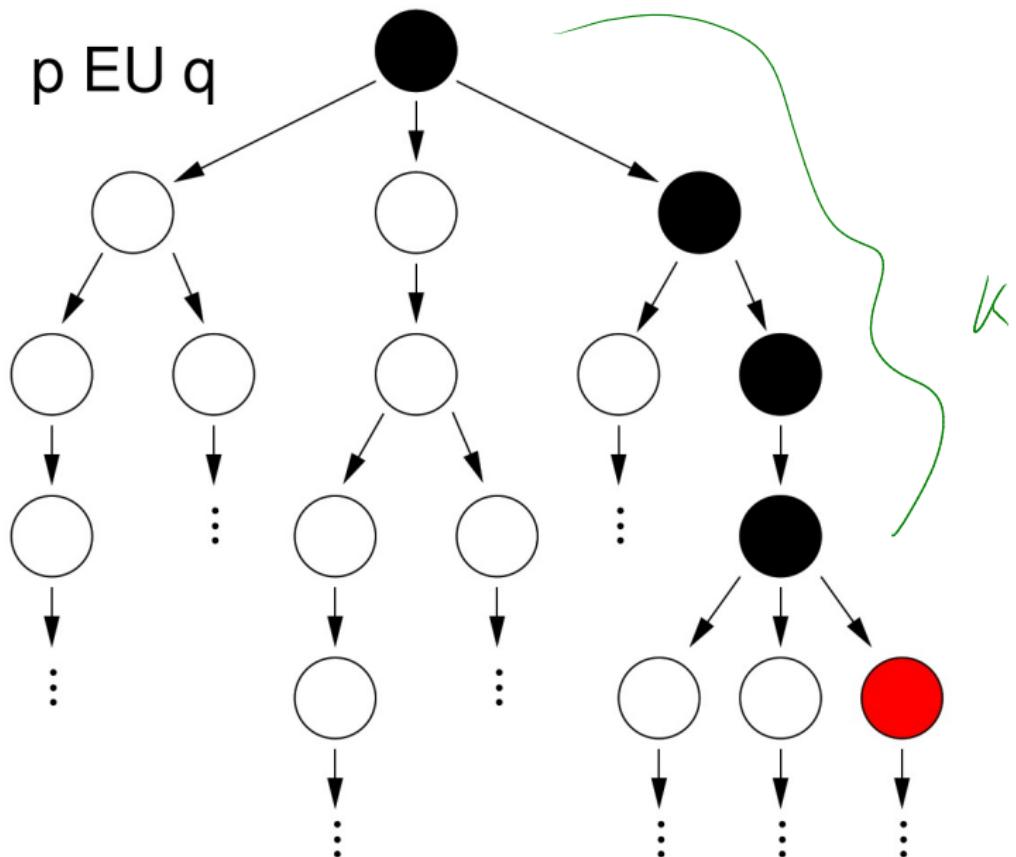
EF p



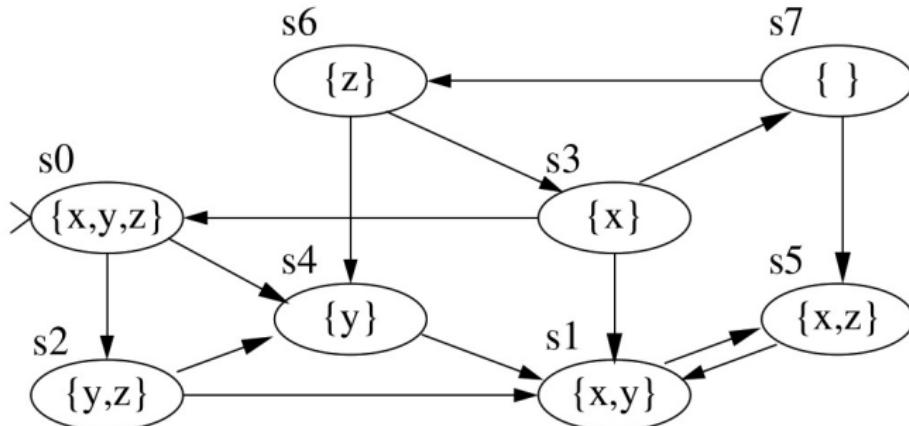
EX p



$p \rightarrow q$

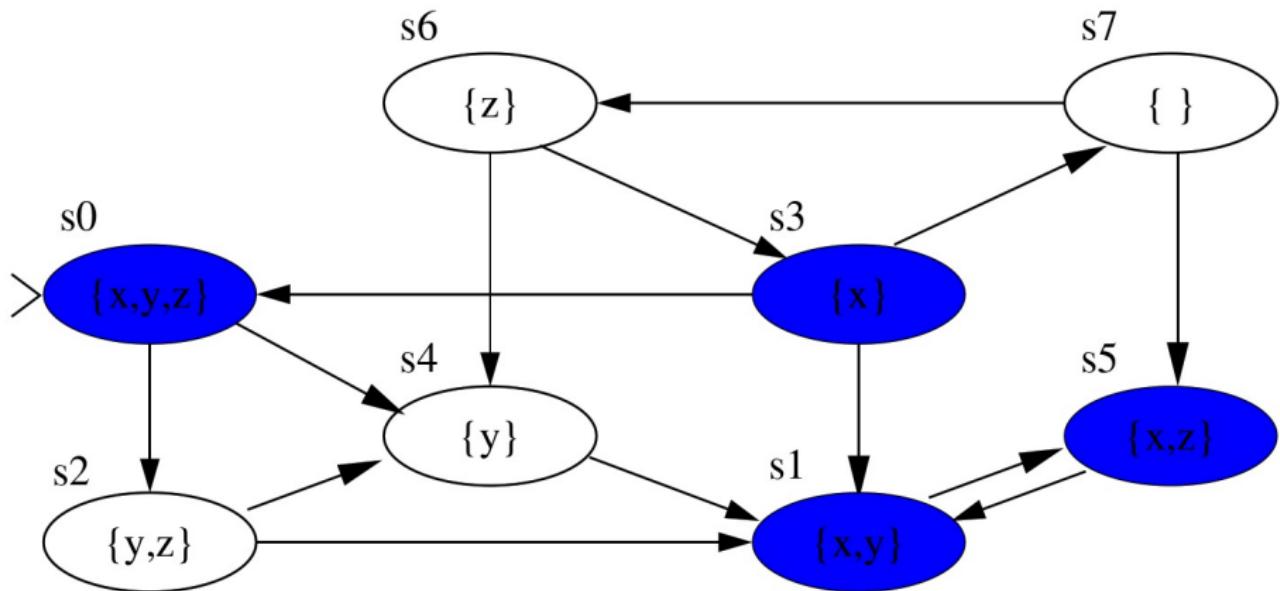


Solving nested formulas : Is $s_0 \in [\text{AFAG } x]$?

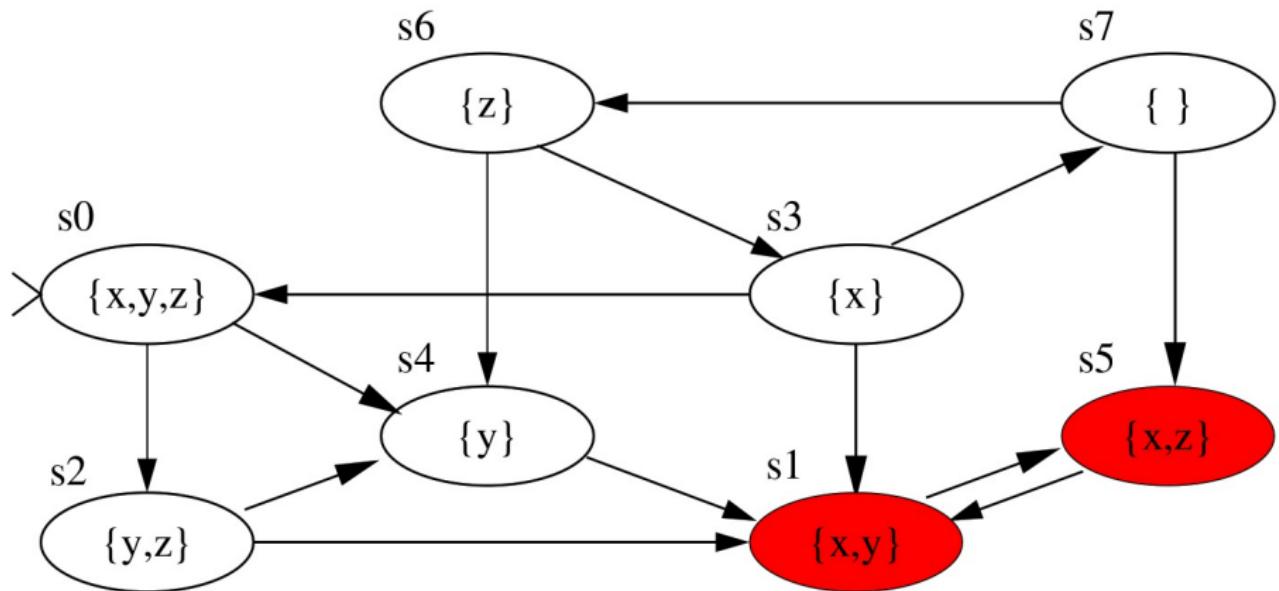


- To compute the semantics of formulas with nested operators, we first compute the states satisfying the innermost formulas ; then we use those results to solve progressively more complex formulas.
- In this example, we compute $[x]$, $[\text{AG } x]$, and $[\text{AFAG } x]$, in that order.

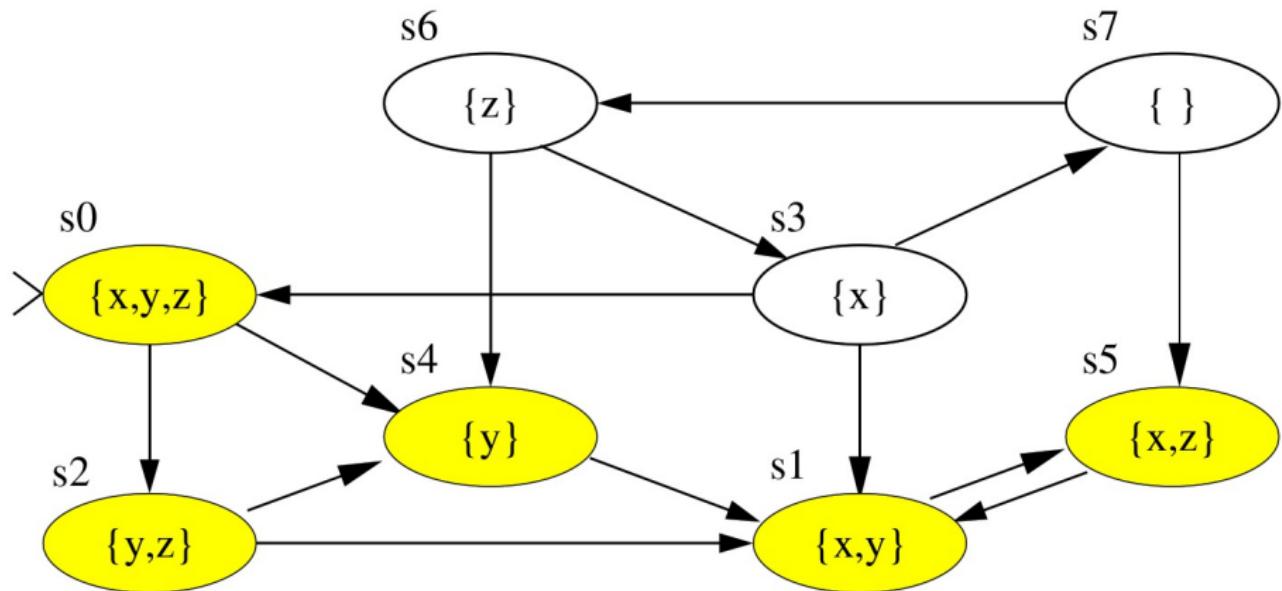
Bottom-up method (1) : Compute $\llbracket x \rrbracket$



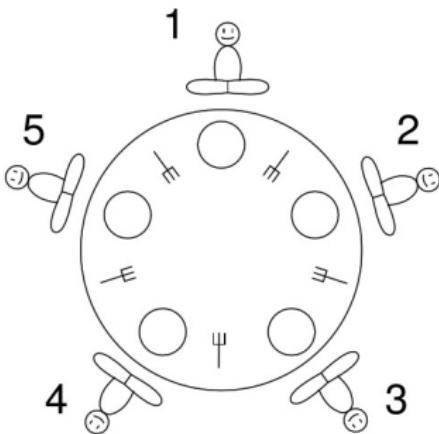
Bottom-up method (2) : Compute $\llbracket \text{AG } x \rrbracket$



Bottom-up method (3) : Compute **[AF 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 :
 - $e_i \hat{\equiv}$ philosopher i is currently eating
 - $f_i \hat{\equiv}$ philosopher i has just finished eating

Expected properties

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

$\forall G \neg e_1 \wedge e_4$



Expected properties

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

$$\text{AG } \neg(e_1 \wedge e_4)$$

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

$$\text{AG} \left(f_4 \Rightarrow \neg e_4 \wedge \text{U } e_3 \right)$$



AW
e₃ n_{on}
par garantie

-

Expected properties

- “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.”

$$\begin{array}{c} \cancel{\gamma e_1 \wedge \gamma e_3 \rightarrow e_4, \gamma e_5} \quad \wedge \vee \quad e_2 \\ \gamma(e_1 \vee \dots \vee e_5) \end{array}$$

Expected properties

- “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 (if you hear about it)

- CTL and LTL have a large overlap, i.e. properties expressible in both logics.
- 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.
- Even though CTL considers the whole computation tree, its state-based semantics. expressible in LTL but not in CTL.
- Also, *fairness conditions* are not directly expressible in CTL
- However there is another way to extend CTL with fairness conditions.

Conclusion

- *Conclusion* : The expressiveness of CTL allows to express complex properties such as liveness or safety properties.
- *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.