Tecniche di Specifica e di Verifica

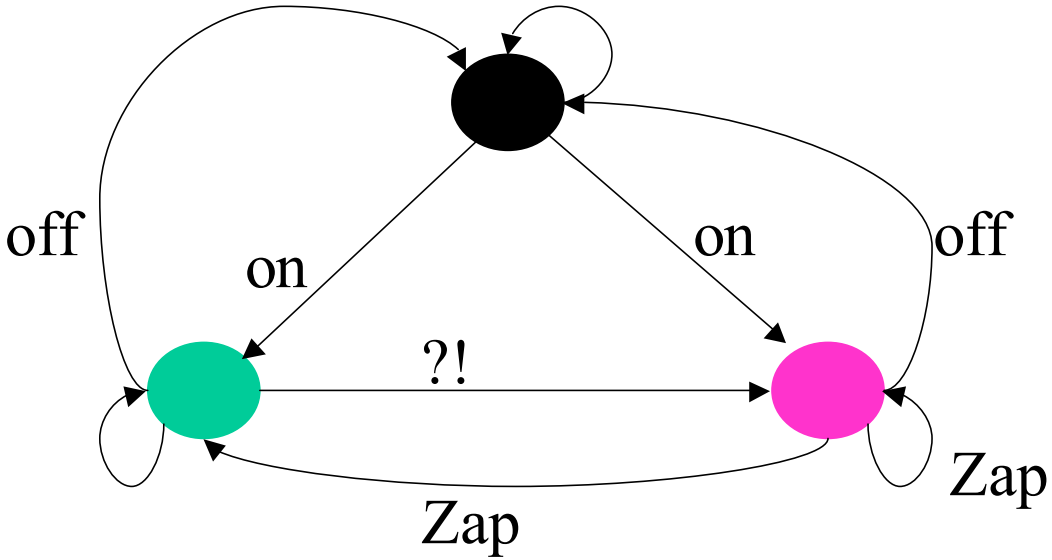
Branching Time Temporal Logics I

Outline

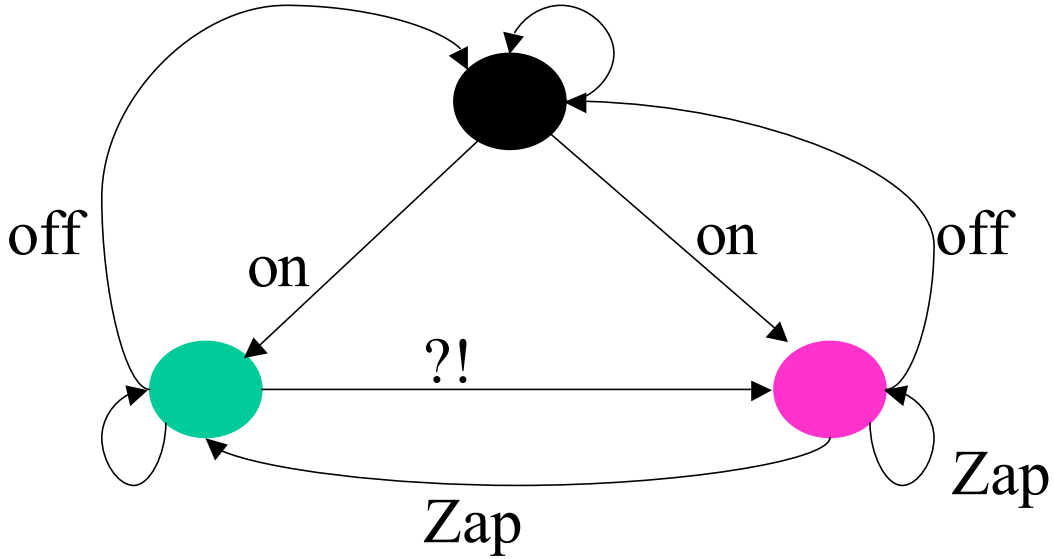
- ***CTL*** (**C**omputation **T**ree **L**ogic)
  - **Branching Time**
  - Unwindings --- computation trees
  - Syntax and semantics of CTL.

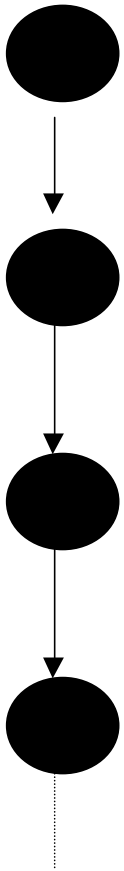
Branching Time Structures

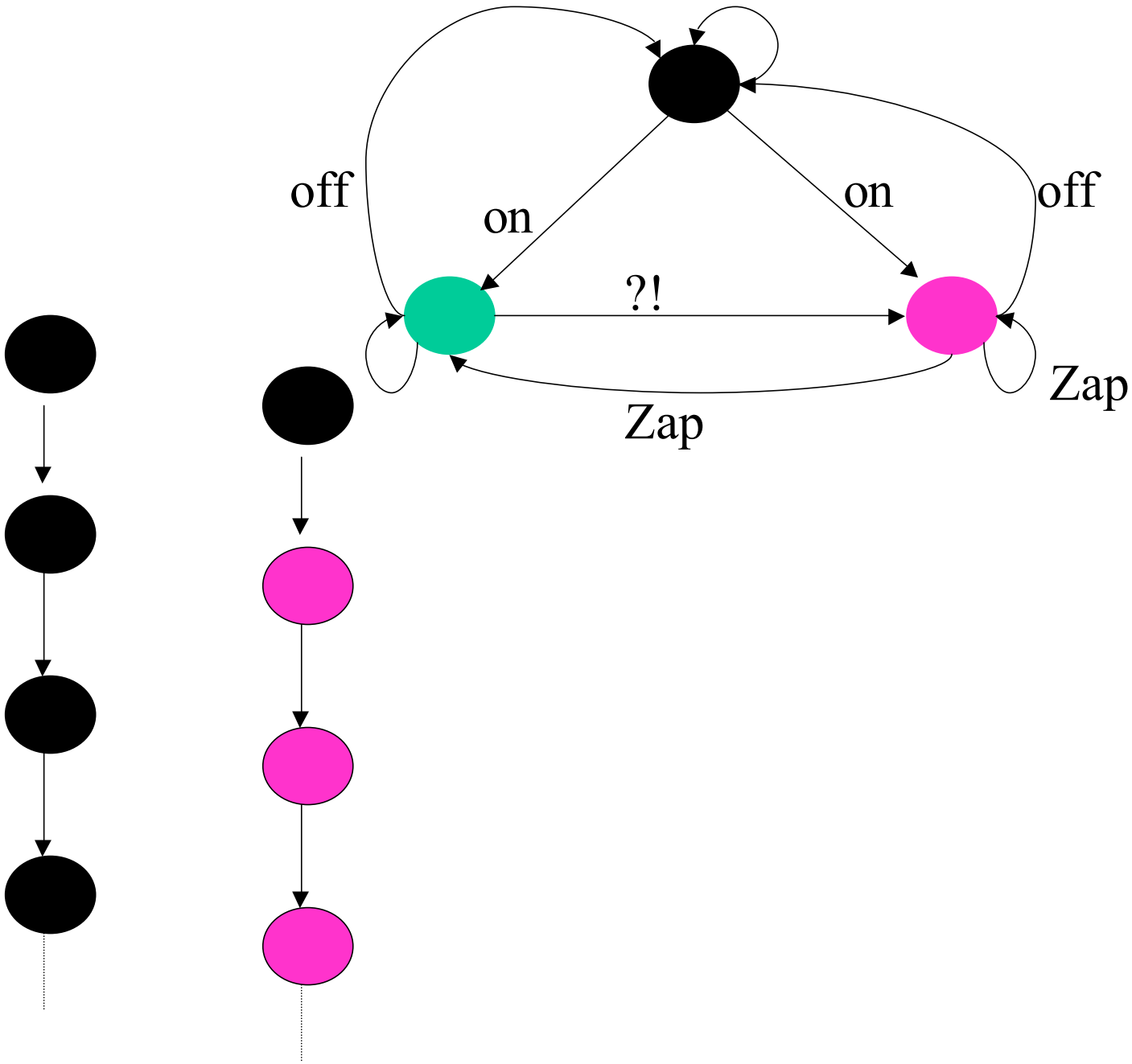
- **Linear Time**:
  - A ***computation*** at its first state satisfies a property.
  - Property ---- **LTL** formula
- **Branching Time**
  - The ***computation tree*** at its root satisfies a property.
  - Property: **CTL** (**CTL***, **m-calculus**) formula.
  - **Computation Tree**
    - **All *computations*** starting from a state ***glued together*** (to form a tree structure).
- In branching time, ***the decisions*** taken during a run are taken into account.

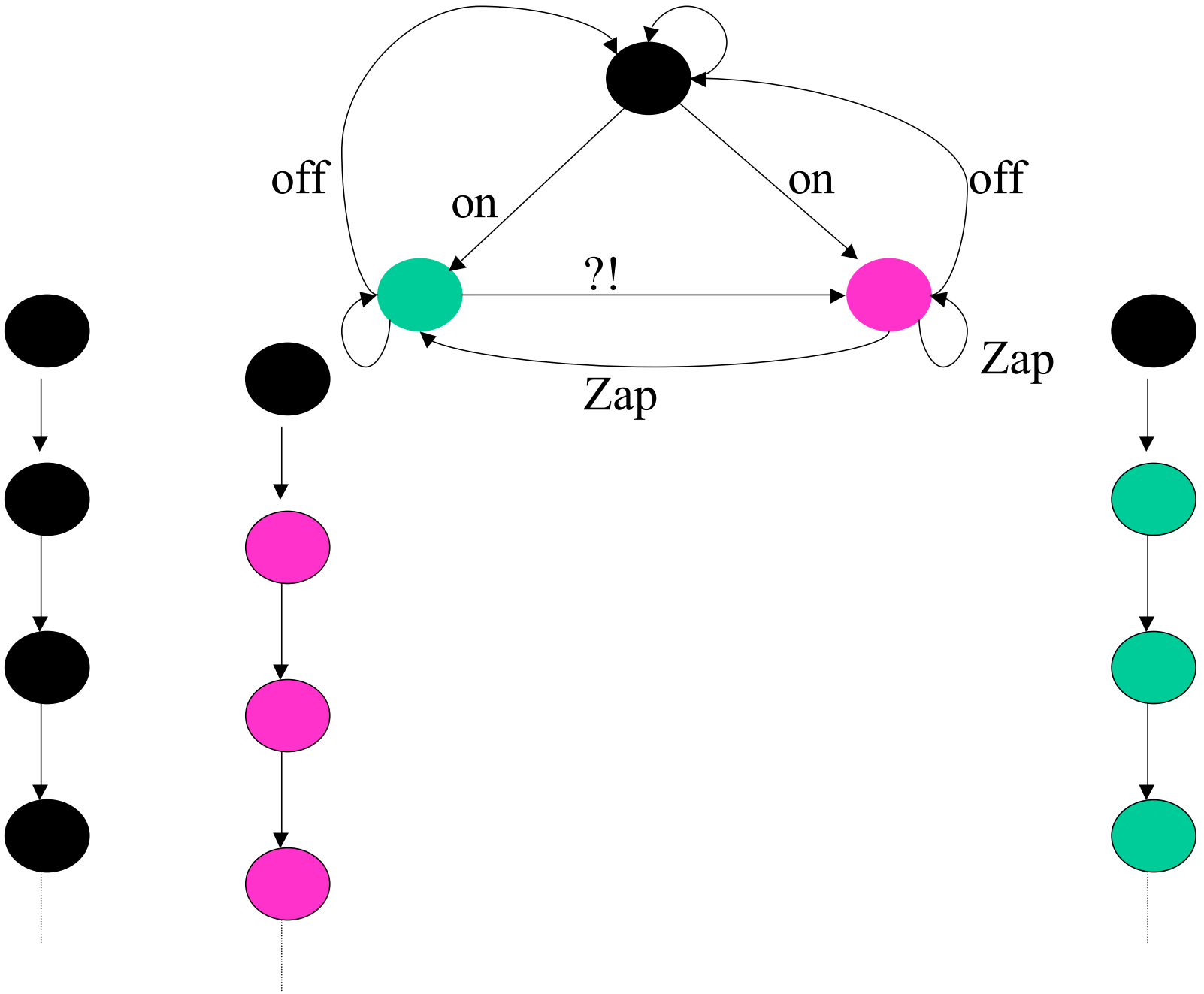


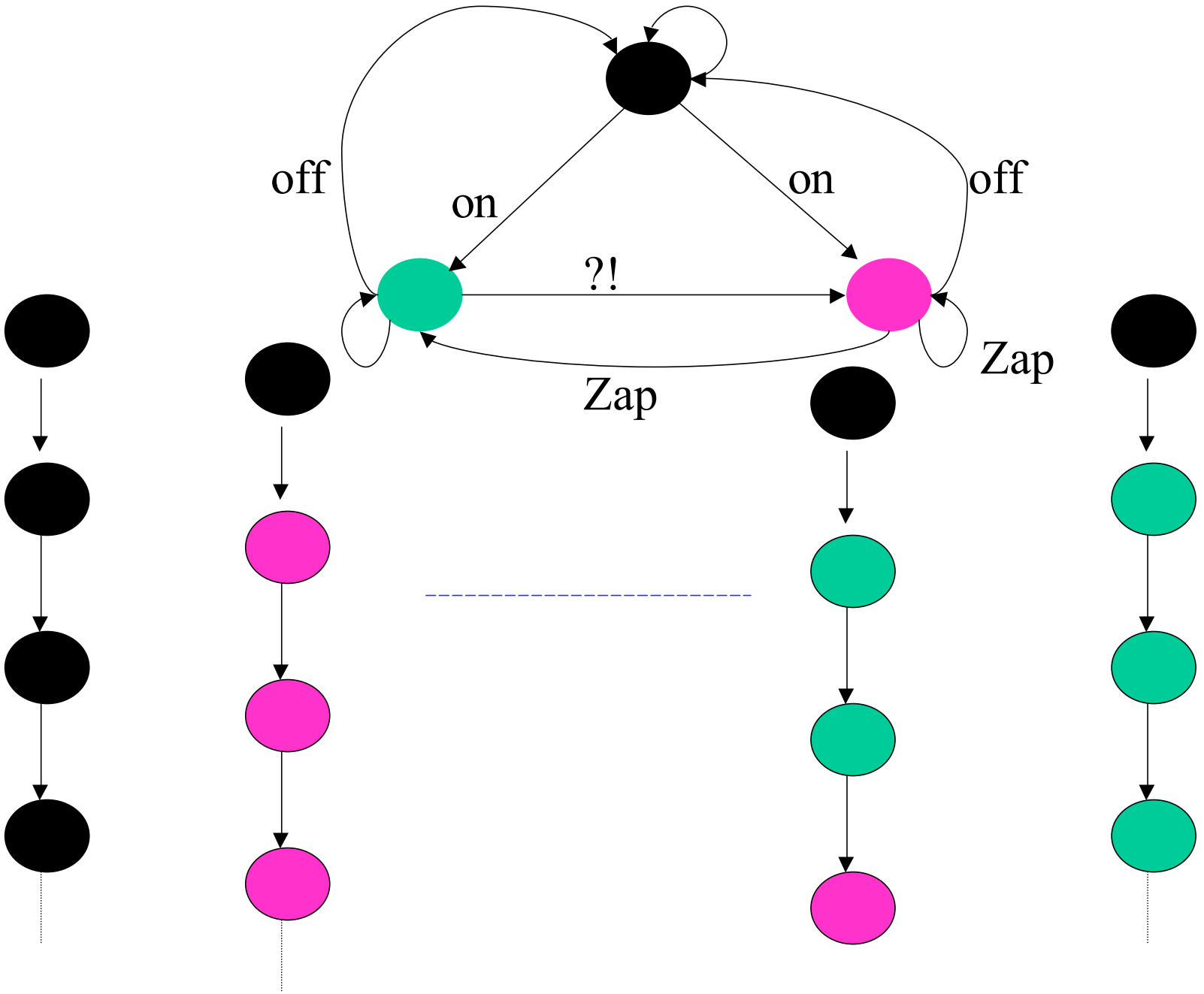
The TV Example


off
on
on
off
?!
Zap
Zap

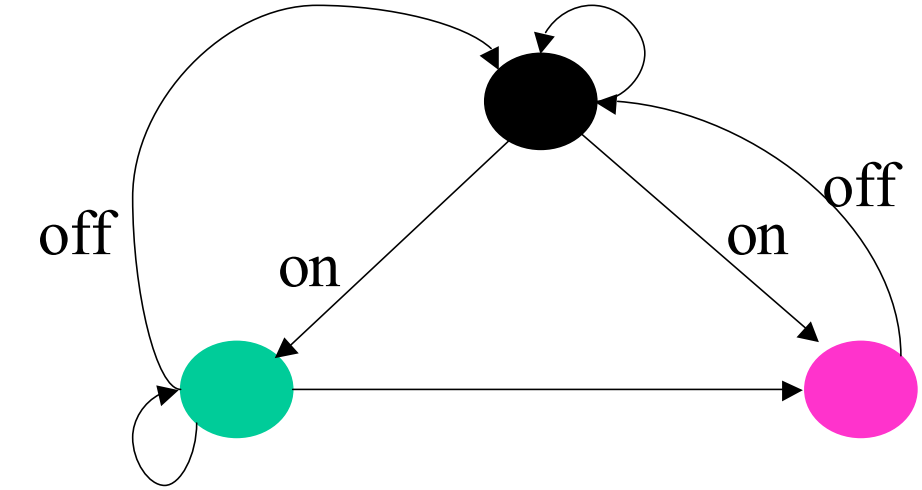


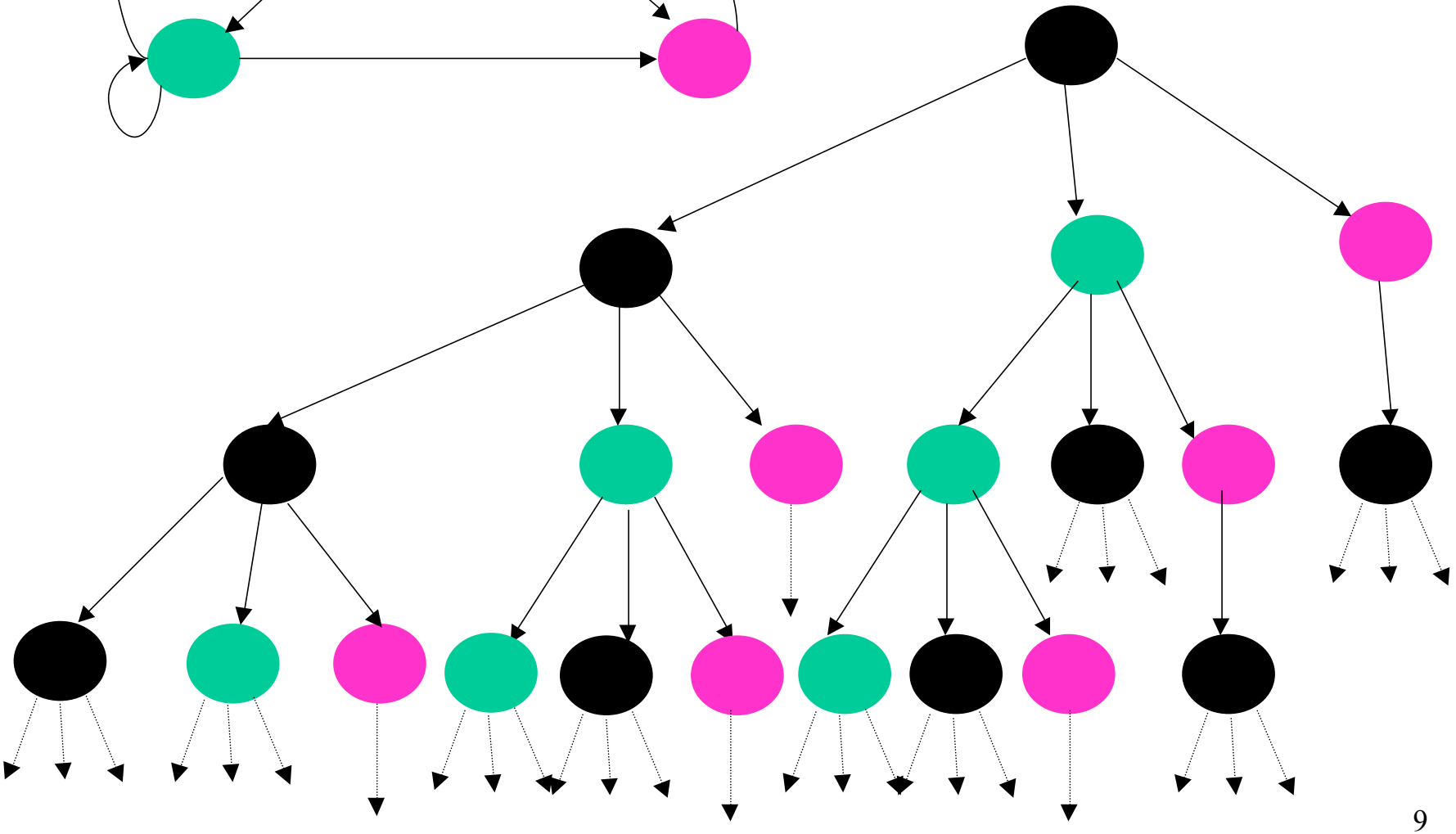

off
on
on
off
?!
Zap
Zap


off
on
on
off
?!
Zap
Zap

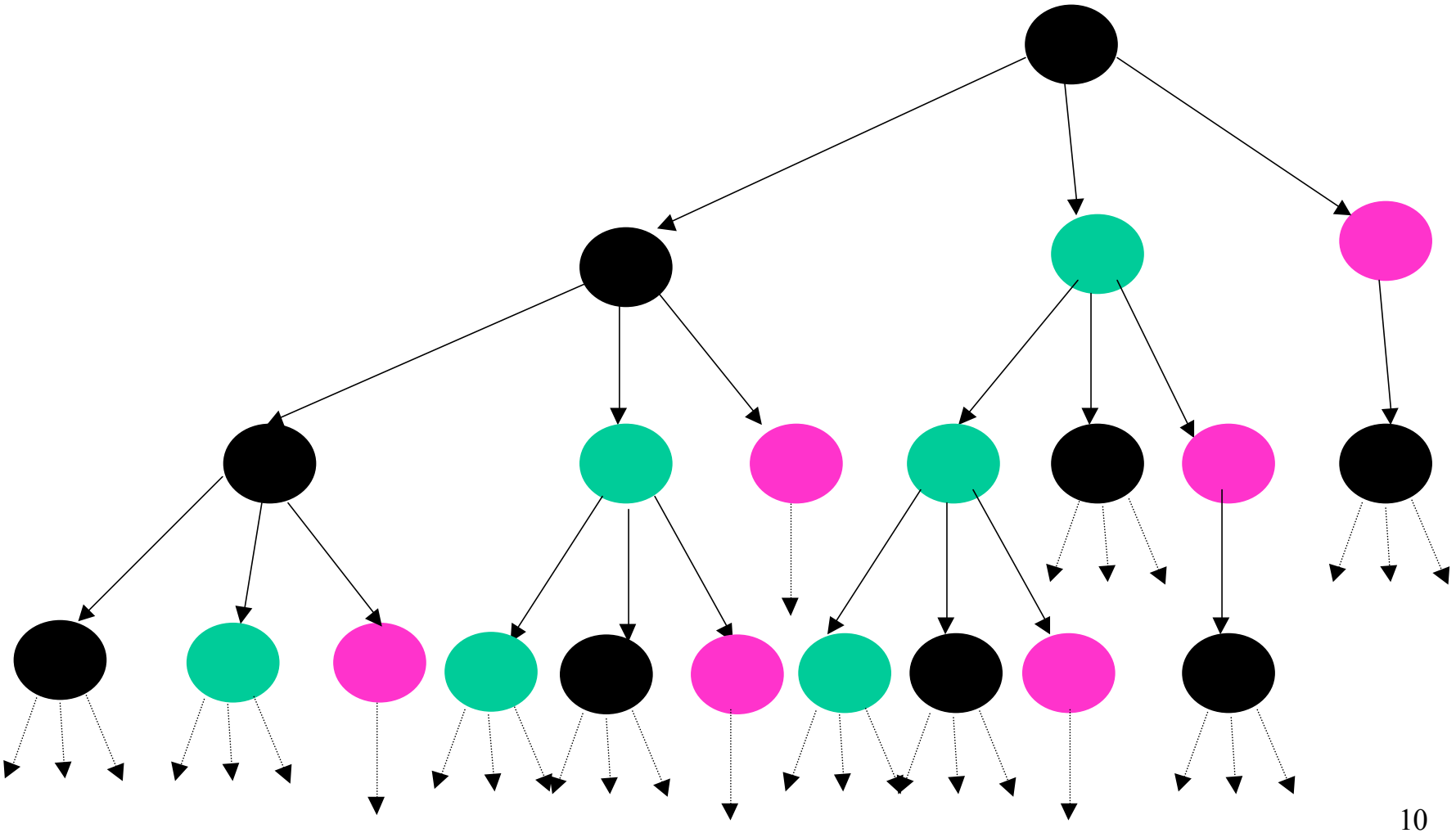

off
on
on
off
?!
Zap
Zap

A Modified Example





For every path p and every state **s** on that path, there is a path p' starting from **s** and a state **s'** on p' which is **green**.



Branching Time Temporal Logic

- **K = (S, S_0, R, AP, L)**
- **K, s ⊨ y** -- the computation tree rooted at **s** satisfies y.
- **K ⊨ y** iff **K, s_0 ⊨ y** for every s_0 Î S_0.
- **Branching Time Temporal Logics**:
  - **CTL**
  - **CTL***
  - (The modal) **m-calculus**

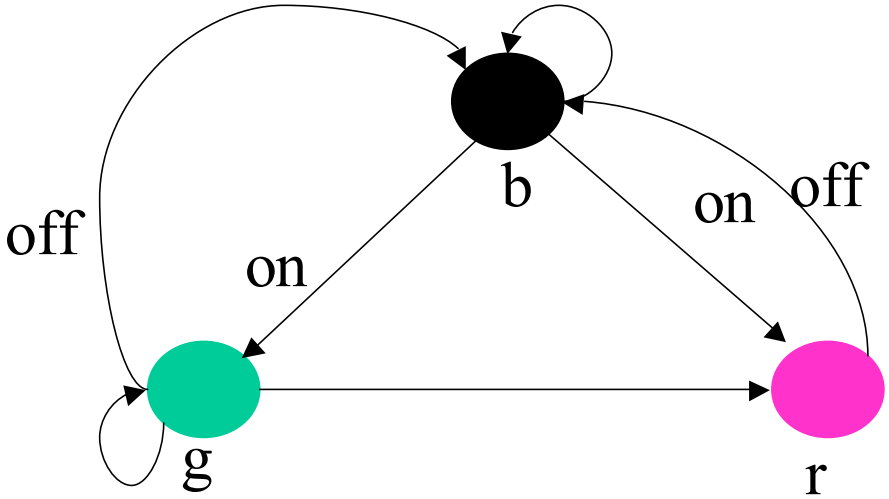
Unwinding

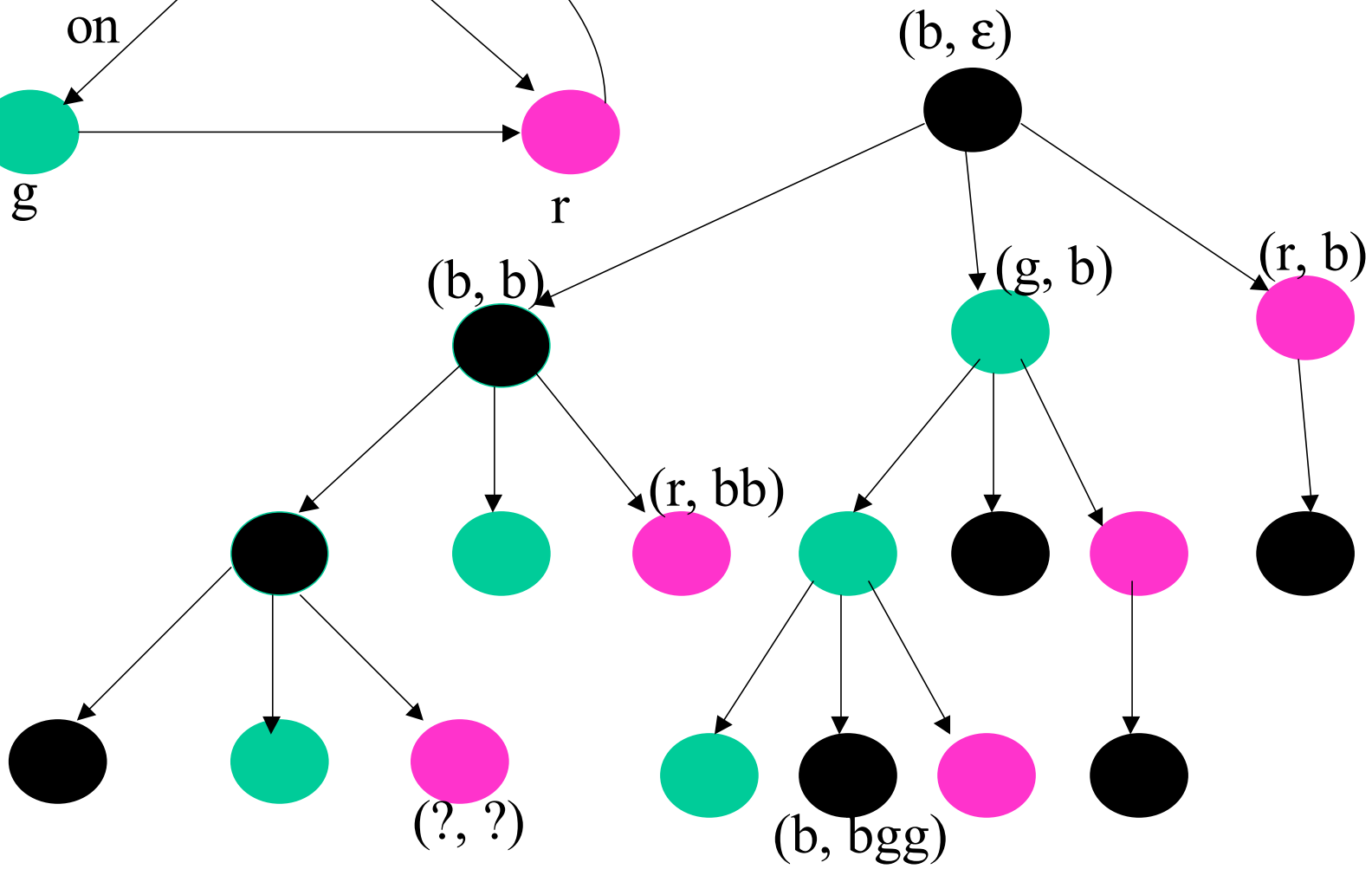
- $K = (S, S_0, R, AP, L)$ $s \in S$
- **TR(K, s)** --- The computation tree rooted at **s**.
- $TR(K, s) = (\mathcal{S}_s, (s,e), \mathcal{R}_s, AP, \mathcal{L}_s)$.
  - $(s, e) \in \mathcal{S}_s$;
  - For any $(s', s) \in \mathcal{S}_s$, $s' \in S$ and $s = s\, s_1 \ldots s_n$ is a path in **K** leading from **s** to **s'** (i.e. $s.s'$ is a path from **s** to **s'**);
  - If $(s_1, s) \in \mathcal{S}_s$ and $R(s_1, s_2)$ then $(s_2, s.s_1) \in \mathcal{S}_s$ and $\mathcal{R}_s((s_1, s), (s_2, s.s_1))$;
  - $\mathcal{L}((s_1, s)) = L(s_1)$.

Unwinding

- **TR(K, s)** is almost a Kripke structure.
  - $\mathcal{S}_s$ may be infinite
  - But $\mathcal{R}_s$ is ***tree-like***.
  - The "***graph***" of **TR(K, s)** is a tree rooted at **(s,** e**)**.
- **TR(K, s)** is the ***computation tree rooted*** at **s**.

A Modified Example





Linear time Vs Branching time

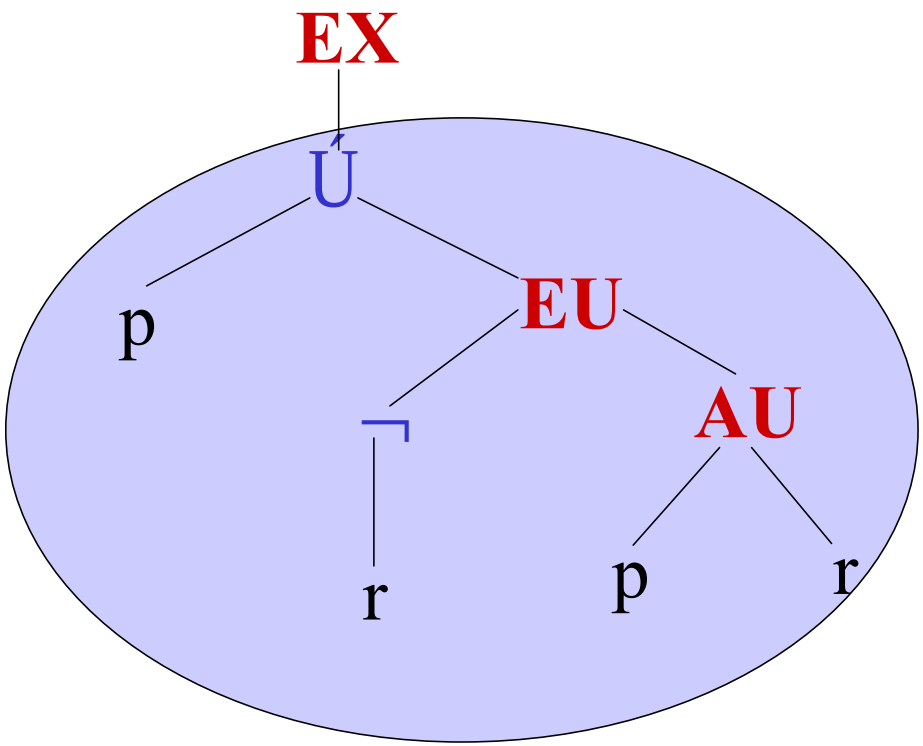
- There are ***properties*** that can be ***expressed in* LTL** but which ***can not be expressed in* CTL**. (sloppy statement!)
- There are ***properties*** that can be ***expressed in* CTL** but ***not in* LTL**.
- The **LTL *model checking problem*** can be ***converted*** into a ***restricted*** kind of a **CTL*** ***model checking problem***.

CTL

- **Syntax**
  - **AP** – a finite set of ***atomic propositions***.
  - **p Î AP** is a formula.
  - If y and y' are formulas then so are Øy and y Ú y'.
  - If y is a formula then so is **EX**y
  - If y_1 and y_2 are formulas then so are **EU**(y_1, y_2) and **AU**(y_1, y_2).

Formulas

- **EX**(p Ú **EU**(Ø**r**, **AU**(p, r)))



Semantics

- **K = (S, S_0, R, AP, L)**
  - **L : S $\longrightarrow$ 2^{AP}**
- **y** a **CTL** formula and **s Î S**
- **K, s ⊨ y**
- **y** (holds) ***is satisfied*** at **s**.
- **FACT**:

K, s ⊨ y iff **TR(K, s), (s, e) ⊨ y**.

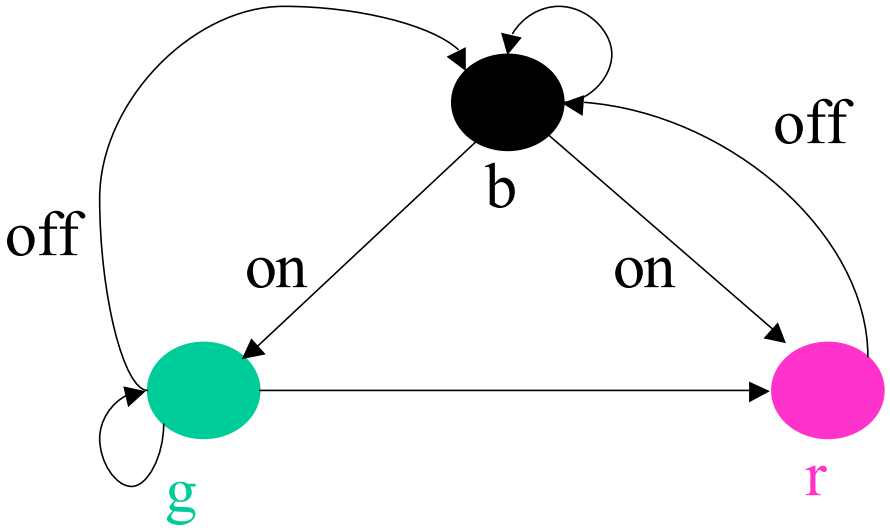
Semantics

- **CTL ::= p | Ø y | y_1 Ú y_2 | EX(y) |**
  | EU(y_1, y_2) | AU(y_1, y_2)
- **K = (S, S_0, R, AP, L) ; L: S ⟶ 2^{AP} ; s Î S**
- **K, s ⊨ p** iff **p Î L(s).**
- **K, s ⊨ Ø y** iff **not K, s ⊨ y**
- **K, s ⊨ y_1 Ú y_2** iff
  K, s ⊨ y_1 or **K, s ⊨ y_2.**

Semantics

- **CTL ::= p | Ø y | y_1 Ú y_2 | EX(y) |**
  | EU(y_1, y_2) | AU(y_1, y_2)
- **K = (S, S_0, R, AP, L) ; L: S ⟶ 2^{AP} ; s Î S**
- **K, s ⊨ EX(y)** iff there exists **s'** such that:
  - **s ⟶ s'** (i.e. **R(s, s')**) and **K, s' ⊨ y**

  s has a successor state **s'** at which **y** holds.



AP = {n, h, uh}

K, b ⊨ EX(uh) ? K, b ⊨ EX(∅uh) ?

K, g ⊨ EX(uh) ?

K, r ⊨ EX(h) ?

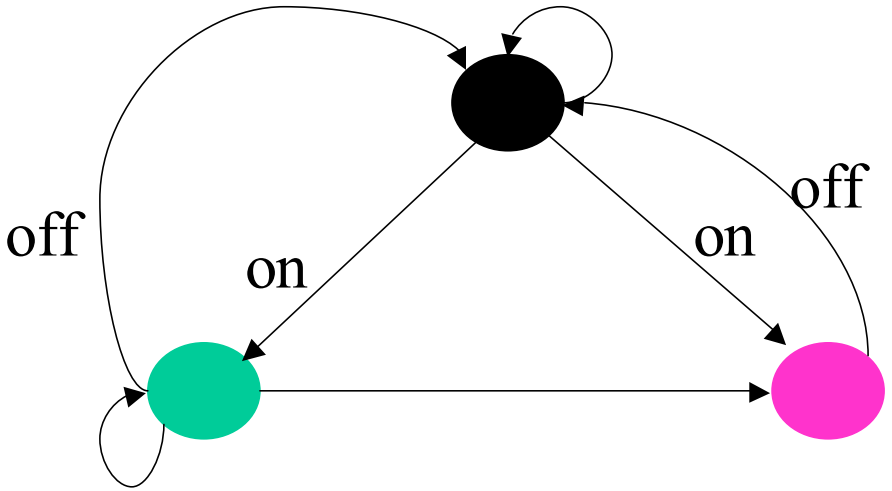
Semantics

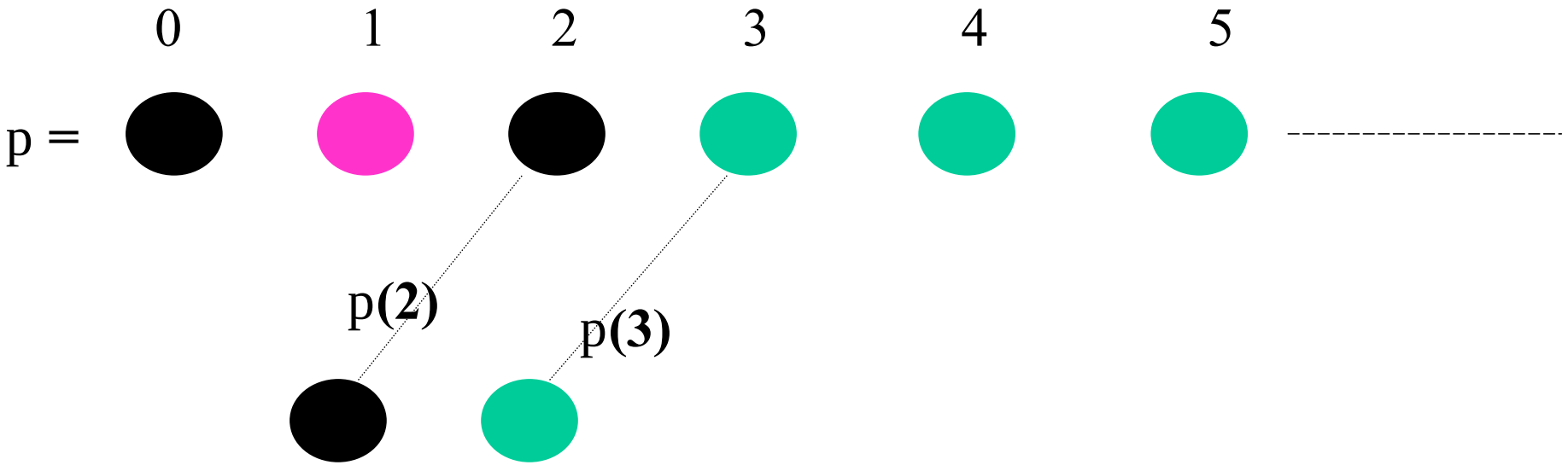
- $K = (S, S_0, R, AP, L)$; $L: S \longrightarrow 2^{AP}$; $s \hat{I} S$
- ***A path from*** **s** is a (infinite) sequence of states

  $p = s_0, s_1, s_2, \ldots, s_i, s_{i+1}, \ldots$ s.t:

  – $s = s_0$

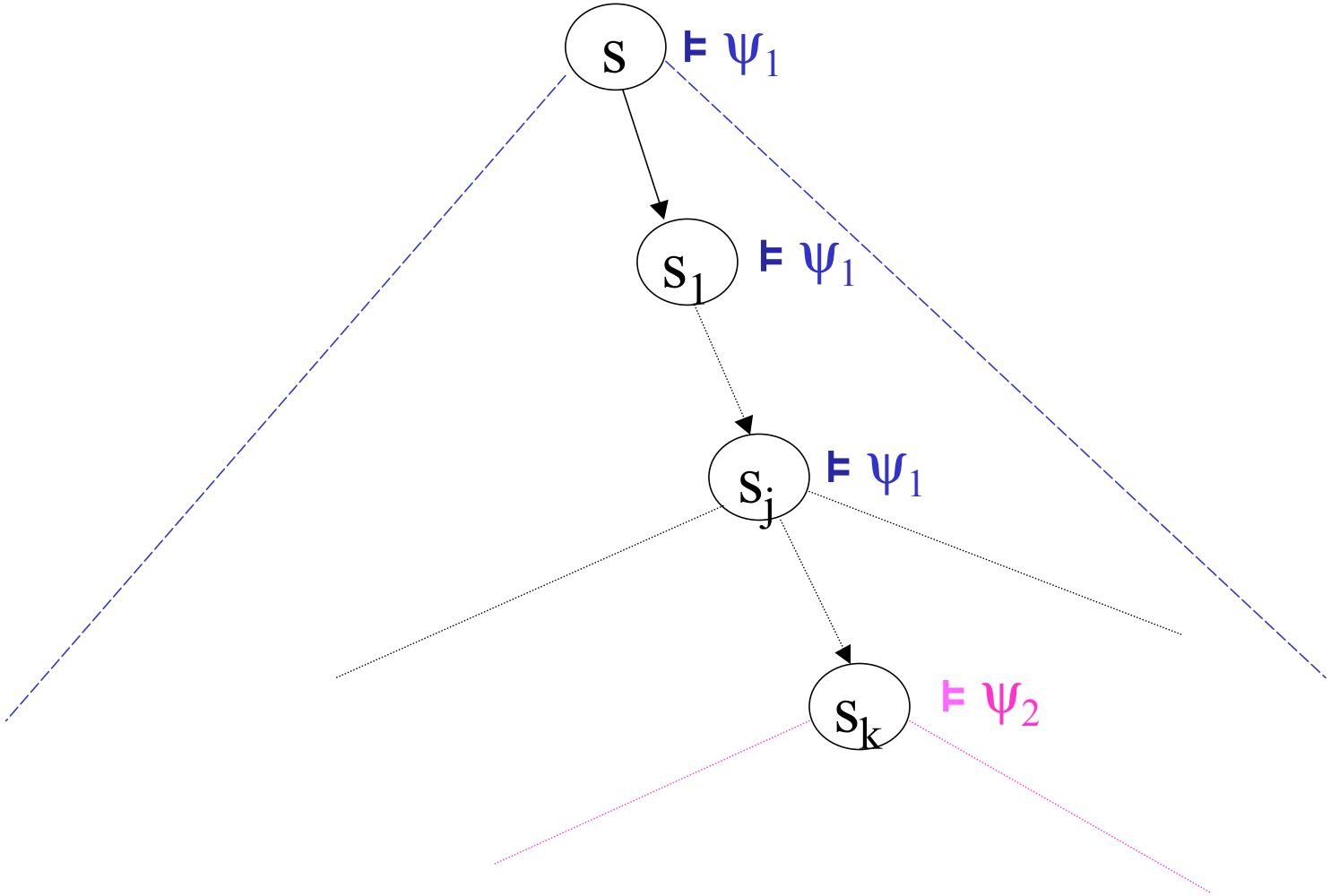
  – $s_i \longrightarrow s_{i+1}$ (i.e. $R(s_i, s_{i+1})$) for every **i**.
- $p(i) = s_i$ the **i**-th element of p.


off
on
on
off


0
1
2
3
4
5
p =
p(2)
p(3)

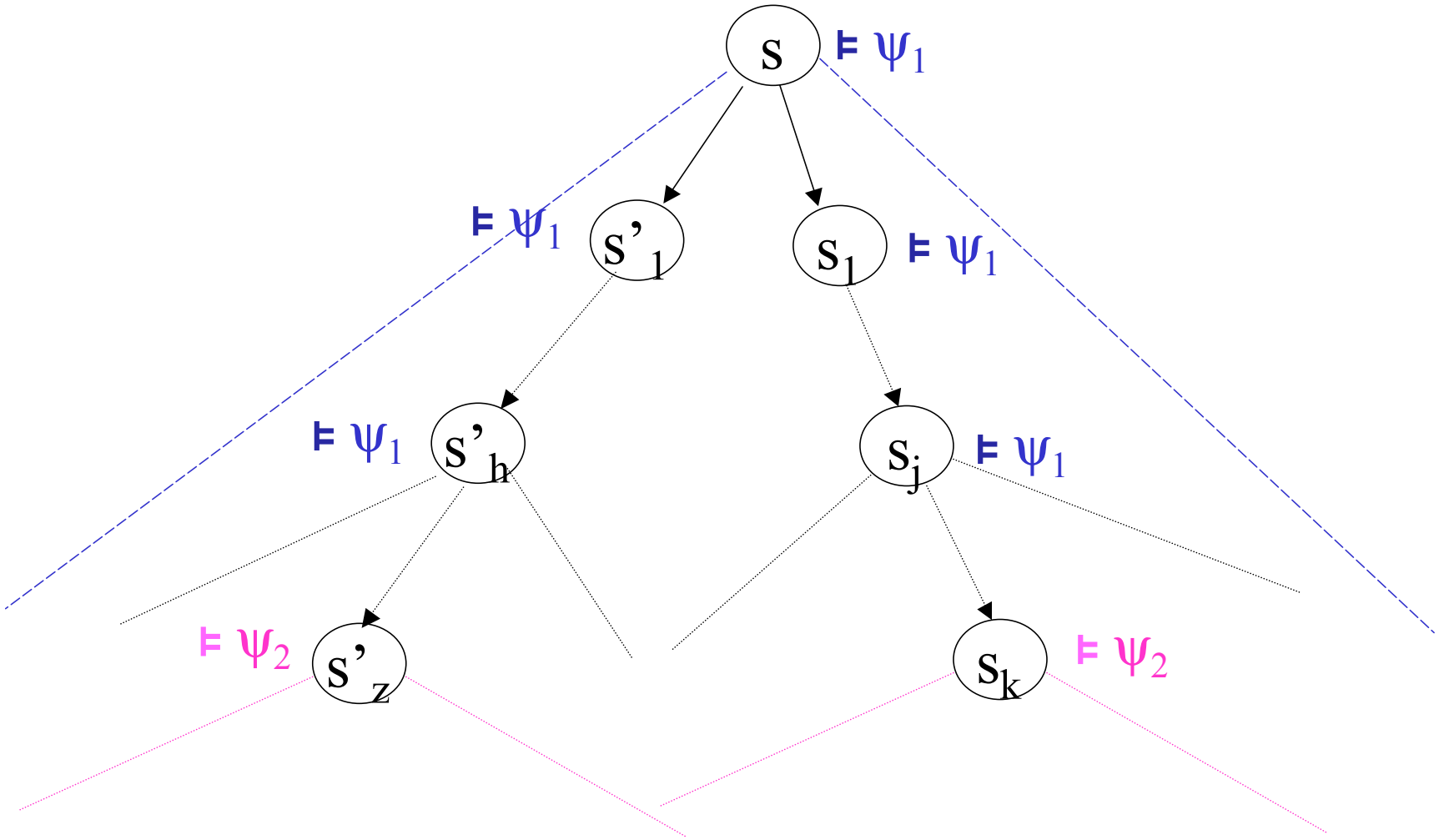
Semantics

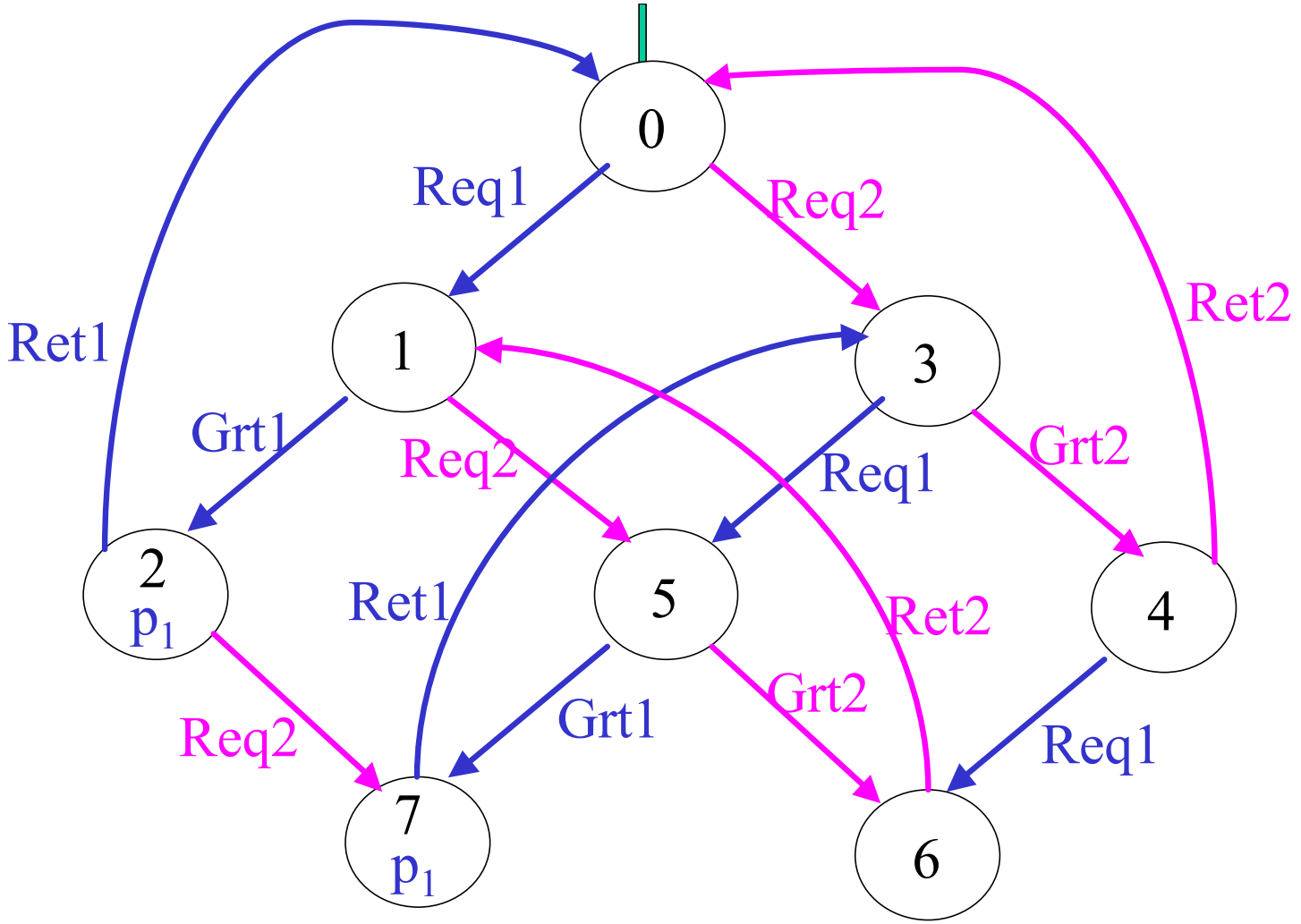
- **CTL ::= p | Ø y | y_1 Ú y_2 | EX(y) |**
  | EU(y_1, y_2) | AU(y_1, y_2)
- **K = (S, S_0, R, AP, L) ; L: S ⟶ 2^{AP} ; s Î S**
- **K, s ⊨ EU(y_1, y_2)** iff ***there exists a path*** p = **s_0, s_1, …** from **s** (i.e. **s_0=s**) and **k ³ 0** such that:
  - **K,** p**(k) ⊨** y_2
  - **K,** p**(j) ⊨** y_1 , for all **0 £ j < k**.


s
⊨ ψ1
s1
⊨ ψ1
sj
⊨ ψ1
sk
⊨ ψ2

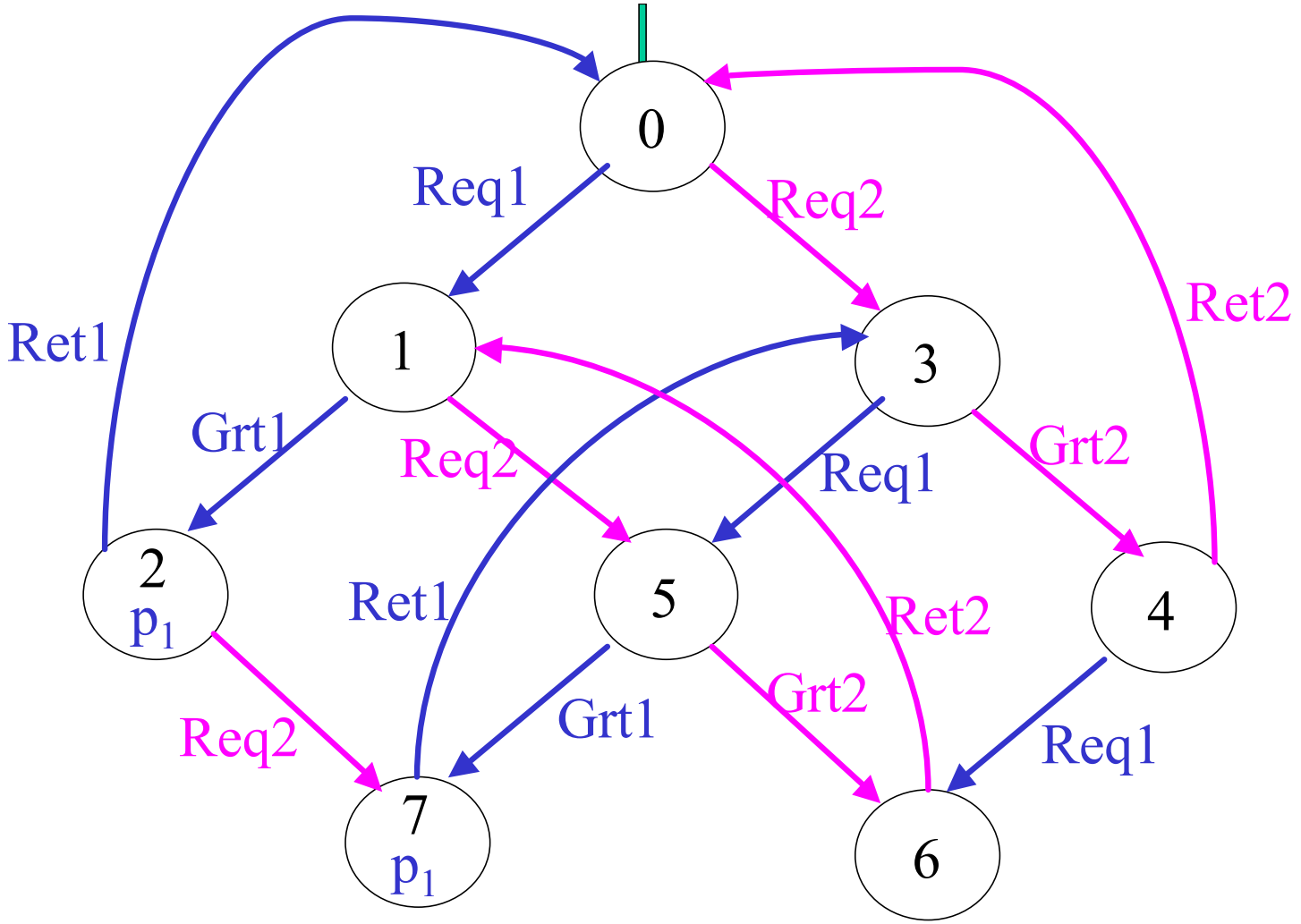
Semantics

- **CTL ::= p | Ø y | y_1 Ú y_2 | EX(y) |**
  | EU(y_1, y_2) | AU(y_1, y_2)
- **K = (S, S_0, R, AP, L) ; L: S ⟶ 2^{AP} ; s Î S**
- **K, s ⊨ AU(y_1, y_2)** iff ***for every path*** p = **s_0, s_1, …** from **s** there exists **k ³ 0** such that:
  - **K,** p**(k) ⊨** y_2
  - **K,** p**(j) ⊨** y_1, for all **0 £ j < k**.

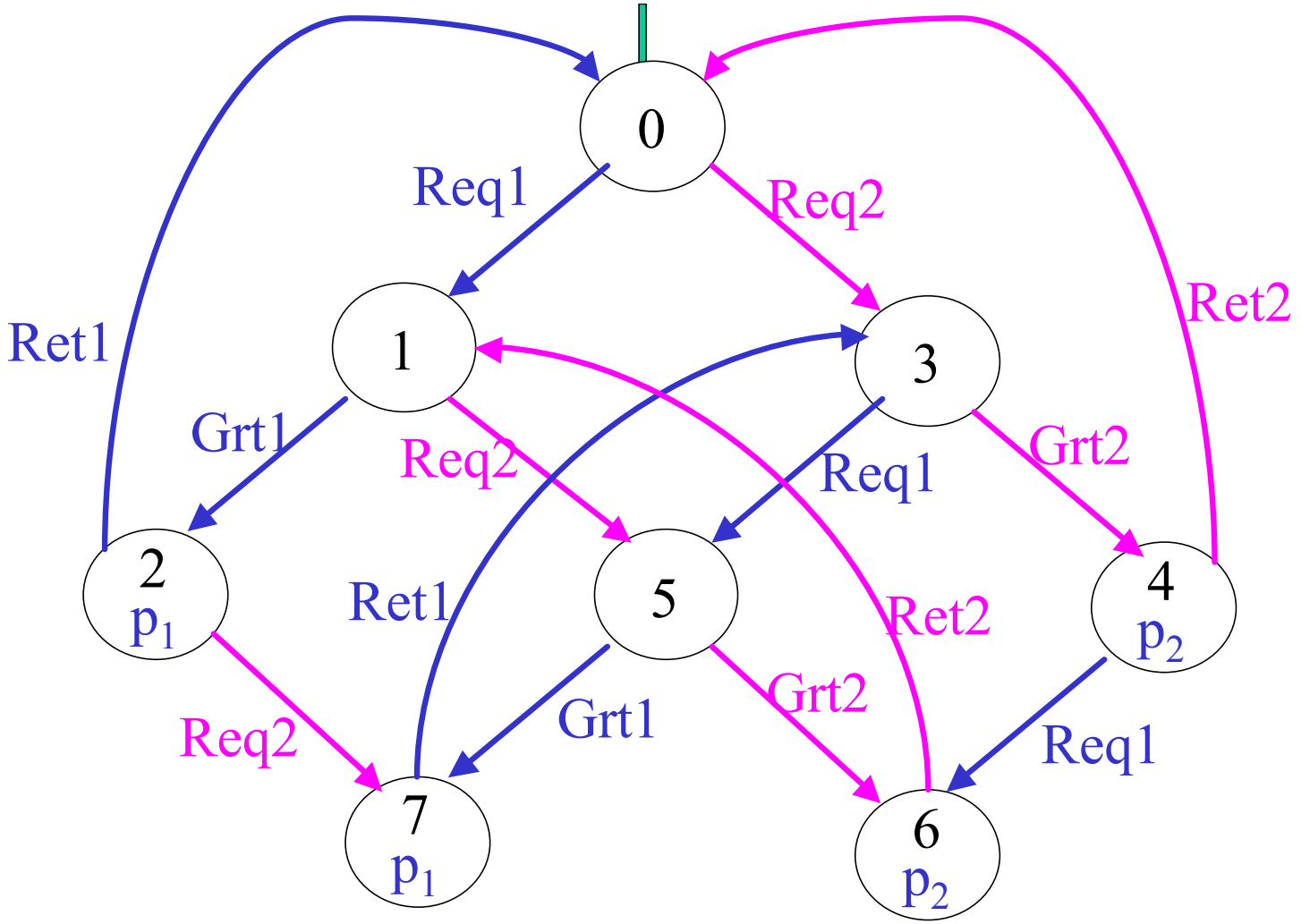

s
⊨ ψ1
⊨ ψ1
s'1
s1
⊨ ψ1
⊨ ψ1
s'h
sj
⊨ ψ1
⊨ ψ2
s'z
sk
⊨ ψ2


0
Req1
Req2
Ret1
Ret2
1
3
Grt1
Req2
Req1
Grt2
2
p1
Ret1
5
Ret2
4
Req2
Grt1
Grt2
Req1
7
p1
6

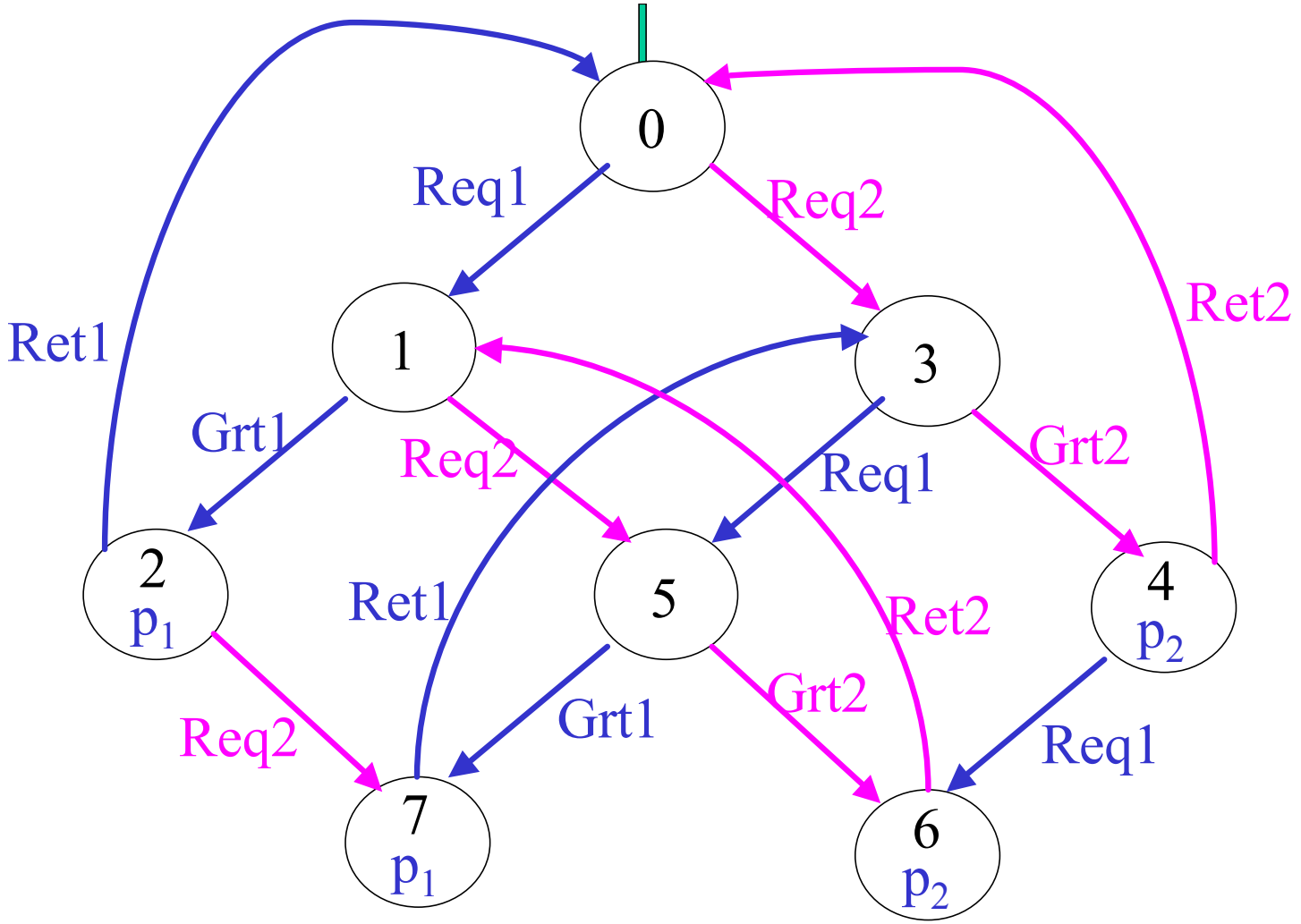
M, 0 ⊨ EU(⊤, p_1) ?


0
Req1
Req2
Ret1
Ret2
1
3
Grt1
Req2
Req1
Grt2
2
p1
5
4
Ret1
Ret2
Grt1
Grt2
Req2
Req1
7
p1
6

M, 0 ⊨ AU(⊤, p_1) ?

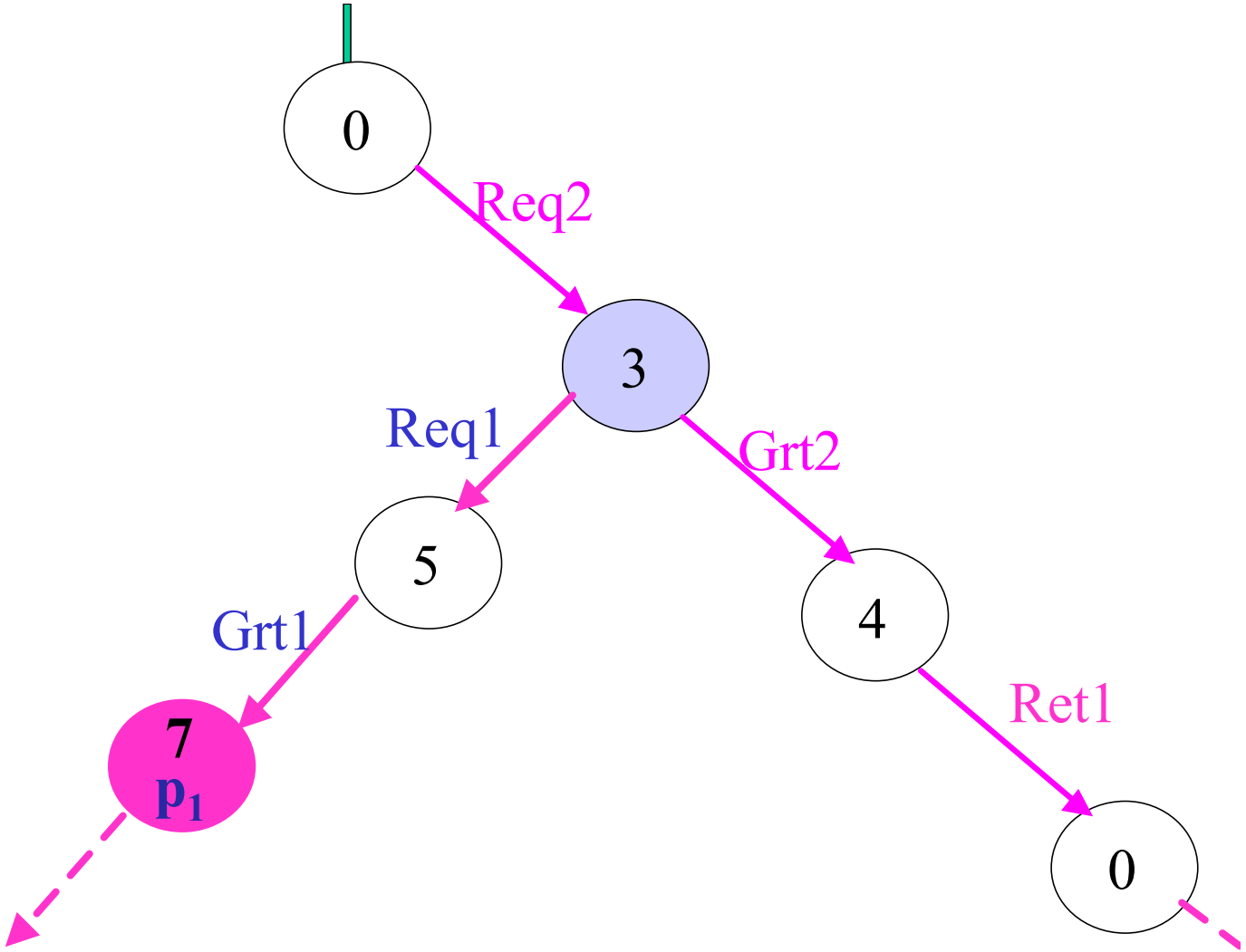

0
Req1
Req2
Ret1
Ret2
1
3
Grt1
Req2
Req1
Grt2
2
p1
Ret1
5
Ret2
4
p2
Req2
Grt1
Grt2
Req1
7
p1
6
p2

M, 0 ⊨ AU(⊤, p_1 Ú p_2) ?



M, 0 ⊨ AU(⊤, EU(⊤, p_1)) ?

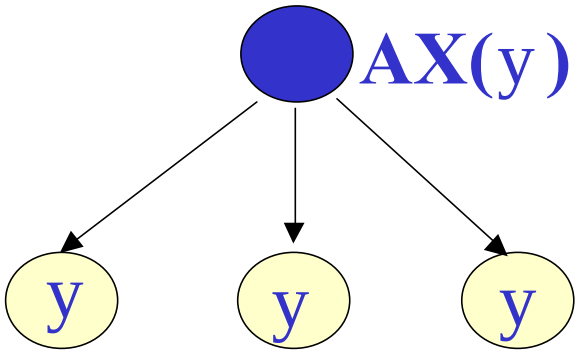
From s_0, ***all*** the computations will reach a point, where it is ***possible*** for 1 to print ***eventually.***



M, 0 ⊨ AU(⊤, EU(⊤, p_1)) ?

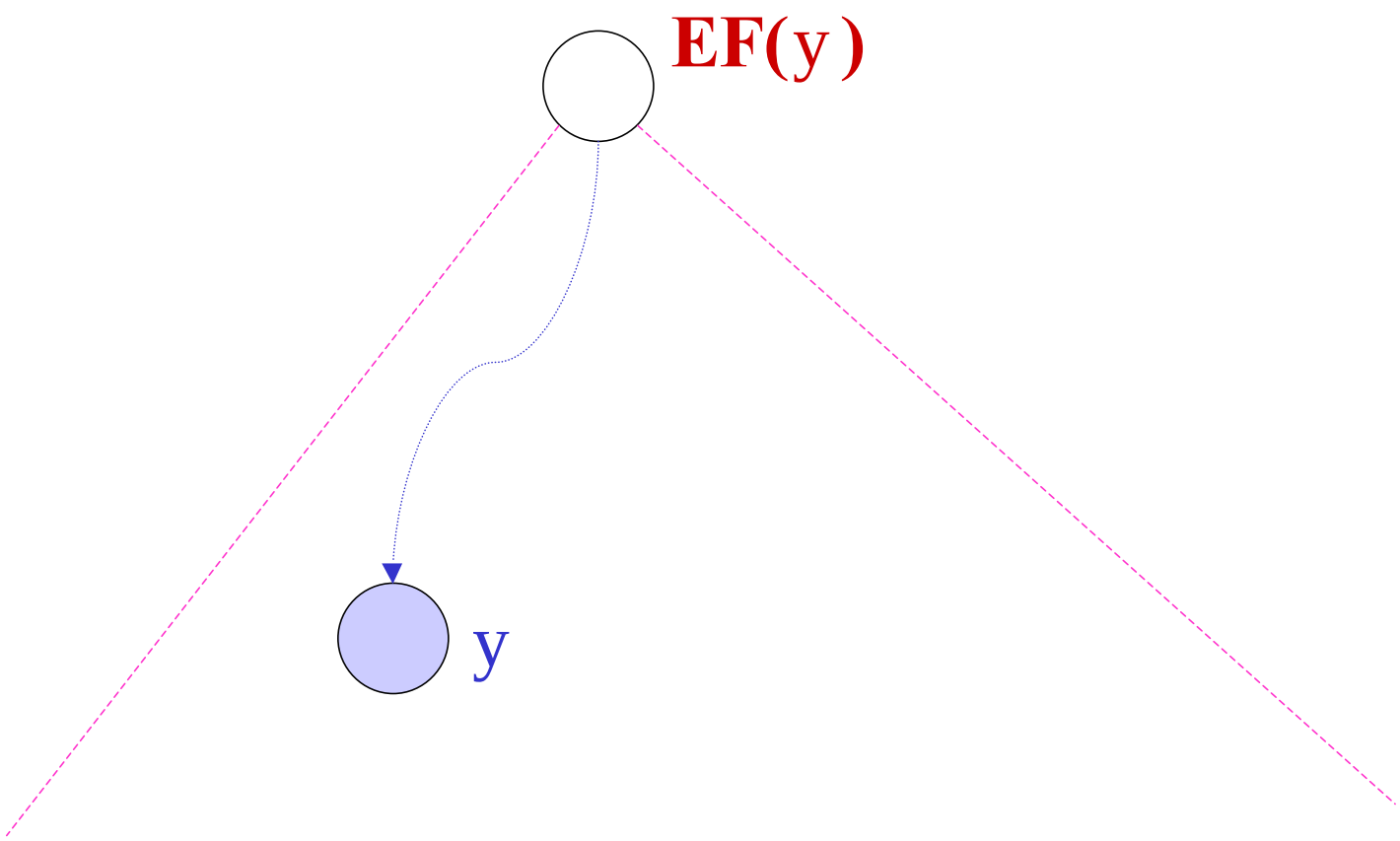
Derived Operators

- **AX(y) = ØEX(Øy)**
  - It is not the case there exists a next state at which y does not hold.
  - ***For every next state*** y holds.



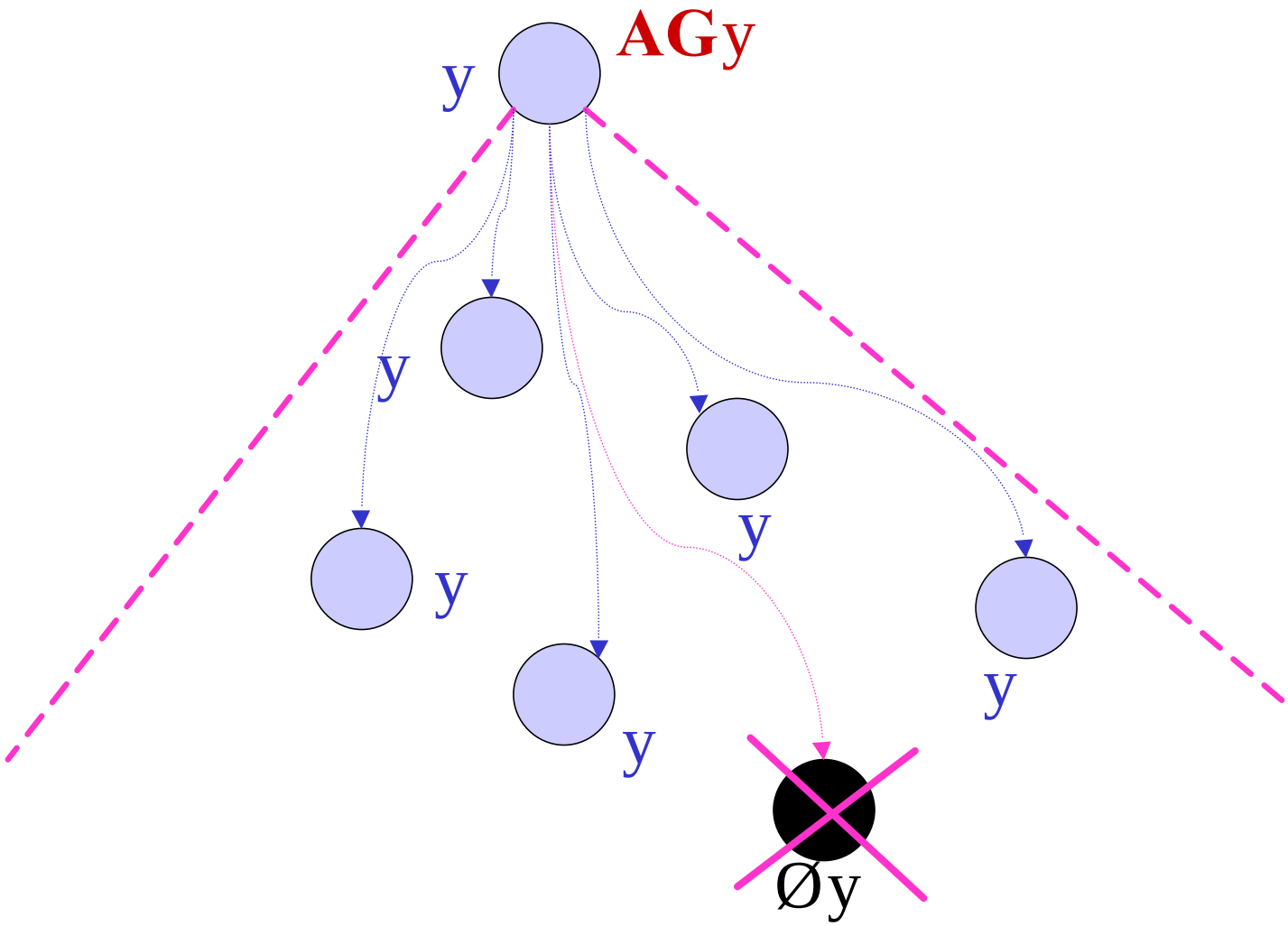
Derived Operators

- **K, s ⊨ EF(y)**
- **EF(y) = EU(⊤, y)**
  - There exists a path p (from **s**) and **k ³ 0** such that:
    - **K, p(k) ⊨ y.**


EF(y)
y

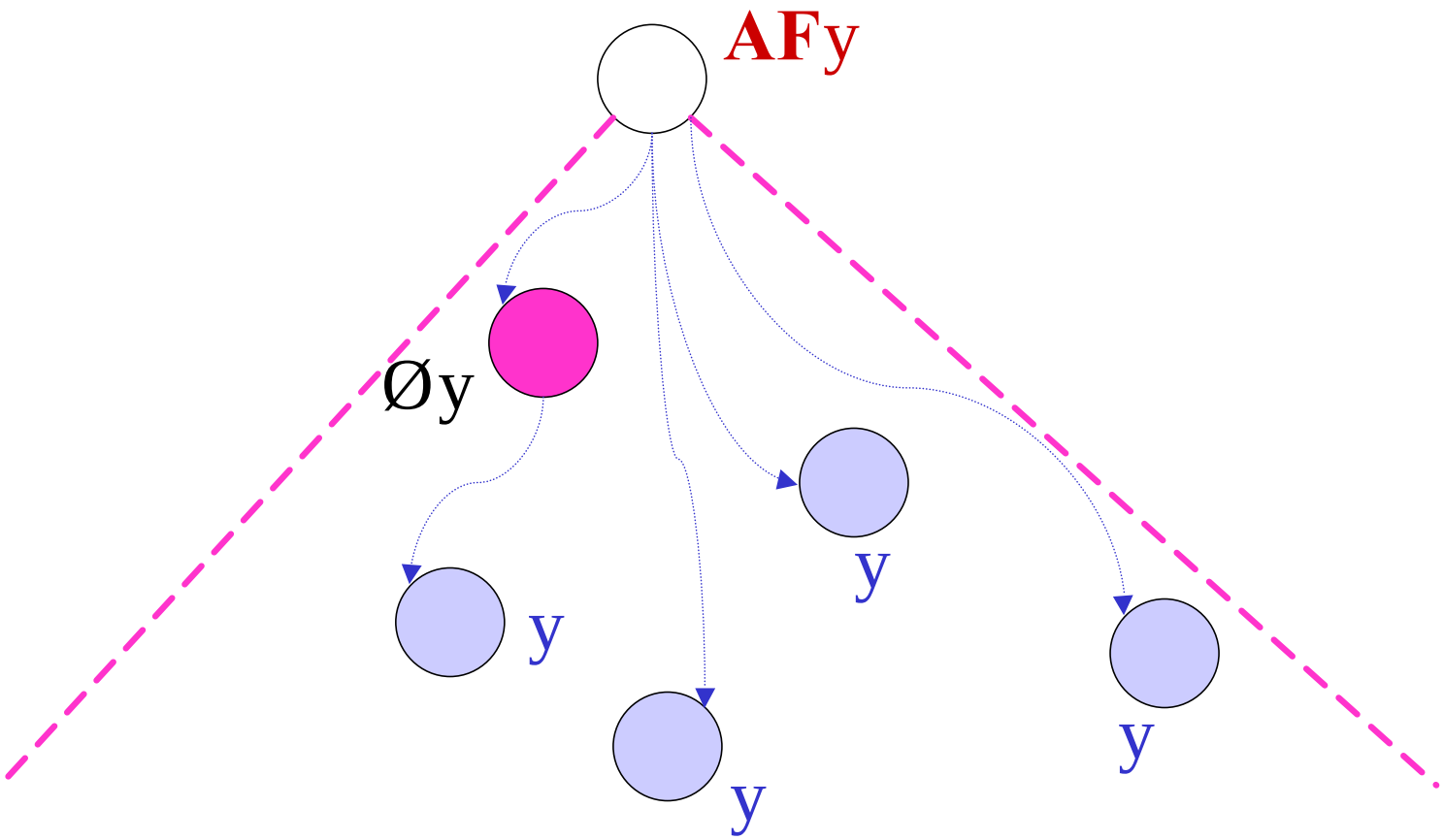
Derived Operators

- **K, s ⊨ AG(y)**
- **AG(y) = ØEF(Øy)**
  - It is ***not*** the case ***there exists a path*** p (from **s**) and **k ³ 0** such that:
    - **K,** p**(k) ⊨** y
  - ***For every path*** p (from **s**) and ***every*** **k ³ 0**:
    - **K,** p**(k) ⊨** y


AGy
y
y
y
y
y
y
Øy

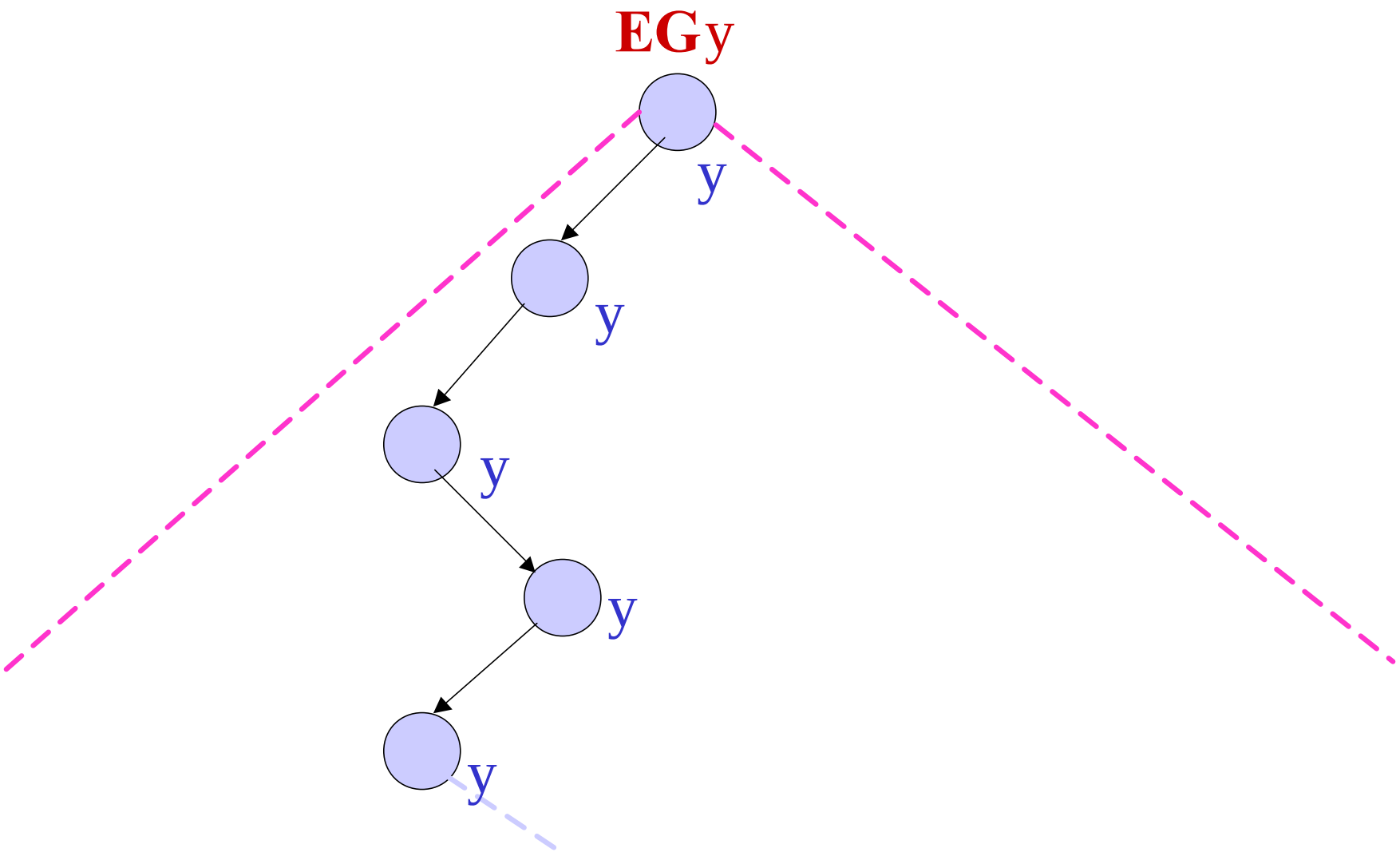
Derived Operators

- **K, s ⊨ AF(y)**
- **AF(y) = AU(⊤, y)**
  - ***For every path*** p from **s**, there ***exists*** **k ³ 0** such that:
    - **K,** p**(k) ⊨** y **.**


AFy
Øy
y
y
y
y

Derived Oparators

- **K, s ⊨ EG(y)**
- **EG(y) = ØAF(Øy)**
  - It is **not** the case that ***for every path*** p from **s** there is a **k ³ 0** such that **K,** p**(k) ⊨** y .
  - ***There exists a path*** p from **s** such that, for every **k ³ 0**:
    - **K,** p**(k) ⊨** y .


EGy
y
y
y
y
y

A more convenient CTL

- **NCTL ::= p** | Ø y | y_1 Ú y_2 | **EX(y)** |
  | **EU(y_1, y_2)** | **EG(y)**
- **CTL ::= p** | Ø y | y_1 Ú y_2 | **EX(y)** |
  | **EU(y_1, y_2)** | **AU(y_1, y_2)**
- **NCTL** is more convenient for model checking!
- Clearly **NCTL** can be defined in terms of **CTL**.

A more convenient CTL

- **NCTL ::= p** | Ø y | y_1 Ú y_2 | **EX(y)** |
  | **EU(y_1, y_2)** | **EG(y)**
- **CTL ::= p** | Ø y | y_1 Ú y_2 | **EX(y)** |
  | **EU(y_1, y_2)** | **AU(y_1, y_2)**
- **CTL** can be defined in terms of **NCTL**!
- The semantics of **NCTL** is given in the obvious way.

A more convenient CTL

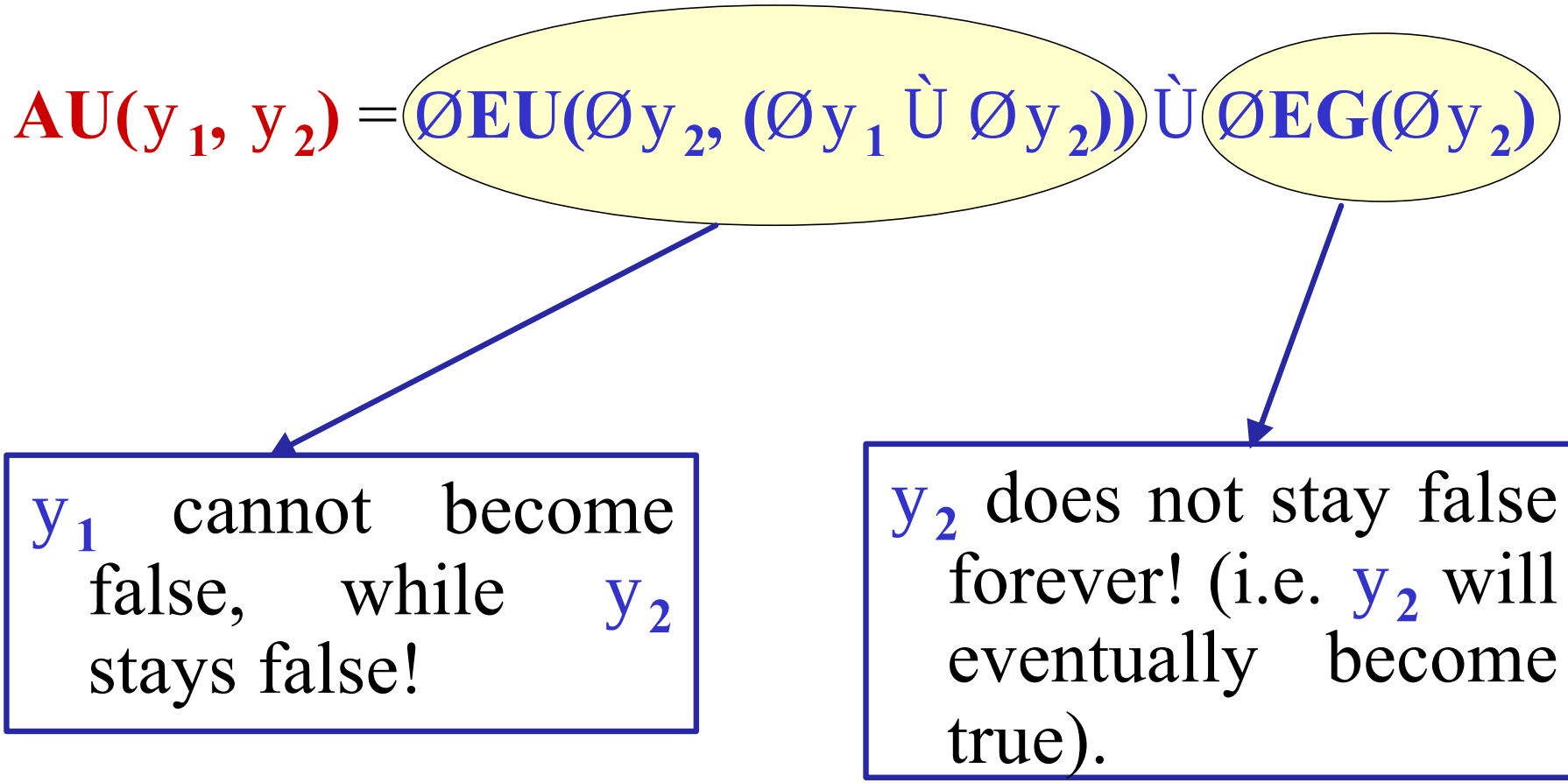
- **NCTL ::= p** | Ø y | y_1 Ú y_2 | **EX(y)** |
  | **EU(y_1, y_2)** | **EG(y)**
- **K, s ⊨ EG(y)** iff ***there exists a path*** p from **s** such that for every **k ³ 0**:
  - **K, p(k) ⊨** y

A more convenient CTL

- **NCTL ::= p | Ø y | y_1 Ú y_2 | EX(y) |**
  | EU(y_1, y_2) | EG(y)
- **CTL ::= p | Ø y | y_1 Ú y_2 | EX(y) |**
  | EU(y_1, y_2) | AU(y_1, y_2)
- **AU(y_1, y_2) = ØEU(Øy_2, (Øy_1 Ù Øy_2)) Ù ØEG(Øy_2)**

i.e., along any path: y_2 must hold eventually and (Øy_1 Ù Øy_2) can only happen after y_2 (recall the *before operator* of LTL)

A more convenient CTL



A more convenient CTL

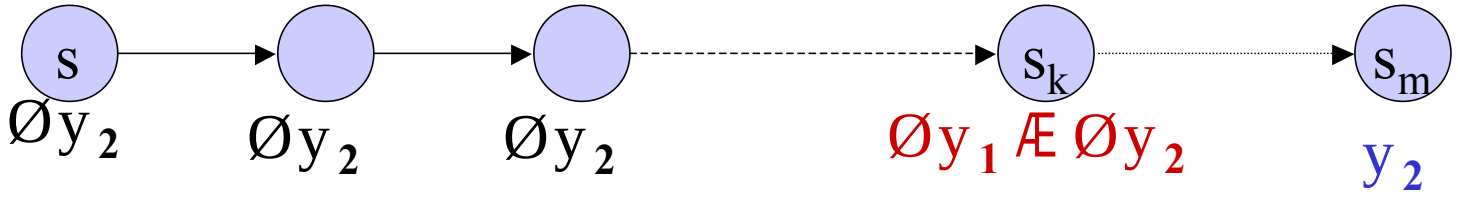
AU(y_1, y_2) = **ØEU(Øy_2, (Øy_1 Ù Øy_2))** Ù **ØEG(Øy_2)**

⇒ Assume **K, s ⊨ AU(y_1, y_2)**

- Let p be a path from **s**. Then there exists **k ³ 0** with:
  - **K, s ⊨ y_2**
- Hence, **not K, s ⊨ EG(Øy_2)**
- Hence, **K, s ⊨ ØEG(Øy_2)**

A more convenient CTL

- **AU(y_1, y_2) = NewAU(y_1, y_2) =**
  ØEU(Øy_2, (Øy_1 Ù Øy_2)) Ù ØEG(Øy_2)
- Clearly **K, s ⊨ AU(y_1, y_2)** implies **K, s ⊨ ØEG(Øy_2)**
- Let **K, s ⊨ AU(y_1, y_2)**
  - Suppose now **K, s ⊨ EU(Øy_2, Øy_1 Ù Øy_2)**
  - Let p be any path from **s** witnessing the above:
  - Let now **k *be the least integer*** such that:
    - **K, p(k) ⊨ Øy_1 Ù Øy_2**
    - **K, p(j) ⊨ Øy_2** for **0 £ j < k.**



- Suppose **K, p(m) ⊨ y 2**, required by **K, s ⊨ AU(y 1, y 2)**
- Take **m** to be the least such number.
- Then **m > k**, since **K, s ⊨ EU(Øy 2, Øy 1 Ù Øy 2)**
- But **0 £ k < m** and **K, p(k) ⊨ Øy 1**
- Hence **not K, s ⊨ AU(y 1, y 2)**. ***Contradiction!***
- Thus **K, s ⊨ AU(y 1, y 2)** also implies:
  - **K, s ⊨ ØEU(Øy 2, Øy 1 Ù Øy 2)**
- So **K, s ⊨ AU(y 1, y 2)** implies **K, s ⊨ NewAU(y 1, y 2)**

From CTL to NCTL

- In a similar way we can argue that:

> ***if*** **K, s ⊨ newAU(y_1, y_2)**
>
> ***then*** **K, s ⊨ AU(y_1, y_2).**

- Hence ***CTL*** can be expressed in terms of ***NCTL***.

A more convenient CTL

- **NCTL ::= p** $\mid \neg y \mid y_1 \vee y_2 \mid$ **EX(y)** $\mid$
  $\mid$ **EU(y_1, y_2)** $\mid$ **EG(y)**
- **CTL ::= p** $\mid \neg y \mid y_1 \vee y_2 \mid$ **EX(y)** $\mid$
  $\mid$ **EU(y_1, y_2)** $\mid$ **AU(y_1, y_2)**
- **AU(y_1, y_2) = NewAU(y_1, y_2) =**
  $\neg$(EU($\neg y_2$, ($\neg y_1 \wedge \neg y_2$)) $\wedge$ **AF(y_2)**
- **NewAU$_1$ = $\neg$ EU($\neg y_2$, ($\neg y_1 \wedge \neg y_2$)**
- **NewAU$_2$ = AF y_2**

$\neg$EG$\neg y_2$ = AF y_2

From CTL to NCTL

- Let **K = (S, S_0, R, AP, L)** and **s Î S**.
- We need to argue:
  - **K, s ⊨ AU(y_1, y_2)** iff

    K, s ⊨ NewAU$_1$ Ù NewAU$_2$
- We already argued that:
  - If **K, s ⊨ AU(y_1, y_2)** then

    K, s ⊨ NewAU$_1$ Ù NewAU$_2$

From CTL to NCTL

$\mathbf{AU}(y_1, y_2) = \neg\mathbf{EU}(\neg y_2, (\neg y_1 \wedge \neg y_2)) \wedge \neg\mathbf{EG}(\neg y_2)$

$\Leftarrow$ We need to argue that:

– If $\mathbf{K}, s \models \mathbf{NewAU}_1 \wedge \mathbf{NewAU}_2$ then

$\mathbf{K}, s \models \mathbf{AU}(y_1, y_2)$

- So assume $\mathbf{K}, s \models \mathbf{NewAU}_1 \wedge \mathbf{NewAU}_2$.
- $\mathbf{NewAU}_1 = \neg\mathbf{EU}(\neg y_2, (\neg y_1 \wedge \neg y_2))$.
- $\mathbf{NewAU}_2 = \neg\mathbf{EG}\neg y_2 = \mathbf{AF}\, y_2$

From CTL to NCTL

- Let p be some path from **s**.
- We need to show that there exists **k ³ 0** such that:
  - **K,** p**(k) ⊨** y_2
  - **K,** p**(j) ⊨** y_1 if **0 £ j < k**.
- But **K, s ⊨ AF** y_2 implies there along any path (and also along p) there exists **k ³ 0** such that:
  - **K,** p**(k) ⊨** y_2
- Assume **k** is the ***least*** such number along p.

From CTL to NCTL

Now consider an arbitrary **m** with **0 £ m < k**.

CLAIM: **K, s(m) ⊨ y_1**

- If the **CLAIM** is true then we are done.
- Suppose instead that **K, s(m) ⊨ Øy_1**.
  - Then **K, s(m) ⊨ Øy_1 Ù Øy_2 (m < k)** ***WHY???***
  - and **K, s(j) ⊨ Øy_2** if **0 £ j < m,** since **j < m < k**
  - Hence **K, s(0) ⊨ EU(Øy_2, Øy_1 Ù Øy_2)**
  - Therefore, **not K, s ⊨ NewAU$_1$** which is a ***contradiction***!

CTL Model Checking

- $\mathbf{K} \models \gamma$ *iff*

  $\mathbf{K}, \mathbf{s}_0 \models \gamma$ for every $\mathbf{s}_0 \in \mathbf{S}_0$.

- The ***CTL model checking problem***.
  - $\mathbf{K} = (\mathbf{S}, \mathbf{S}_0, \mathbf{R}, \mathbf{AP}, \mathbf{L})$ (system model)
  - γ a ***CTL*** formula (spec. of the property)

- Given $\mathbf{K}$ and γ ***determine whether or not*** $\mathbf{K} \models \gamma$

CTL Model Checking

- The actual model checking problem:
  - Given **$K = (S, S_0, R, AP, L)$**
  - Given **$s \in S$**
  - Given y, **an NCTL formula**.
  - Determine whether:

$K, s \models y$

The Sub-formulas of ψ

- **SF(y)** is the ***least set of formulas*** satisfying:
  - y Î **SF(y)**
  - If Øa Î **SF(y)** then a Î **SF(y)**.
  - If a Ú b Î **SF(y)** then a, b Î **SF(y)**
  - If **EX**a Î **SF(y)** then a Î **SF(y)**.
  - If **EU**(a, b) Î **SF(y)** then a, b Î **SF(y)**
  - If **EG**a Î **SF(y)** then a Î **SF(y)**.
- **SF(y)** ---- The ***set of sub-formulas*** of y.

The Labeling Procedure.

- **K = (S, S_0, R, AP, L)**
  - **s Î S**
  - y a ***NCTL*** formula (built out of **AP**).
- **Strategy**:
  - Construct **Labels: S ⟶ $2^{SF(y)}$**
  - **$2^{SF(y)}$**, the set of subsets of **SF(y)**.
  - Each state of **K** is assigned a subset of a **SF(y)** by the Labels function.
- **K, s ⊨ y *iff* y Î Labels(s)**.

The Labels function

- **Stage 1**:
  - For every **t Î S**:
  - **Labels(t) = L(t)** (**K =(S, S_0, R, AP, L)**)
- ….

Assume we have done up to stage **i**.

- **Stage i +1**:
  - For every **t Î S**:
  - If a = Øb then
    a î **Labels(t)** ***iff*** b Ï **Labels(t)**.

The Labels function

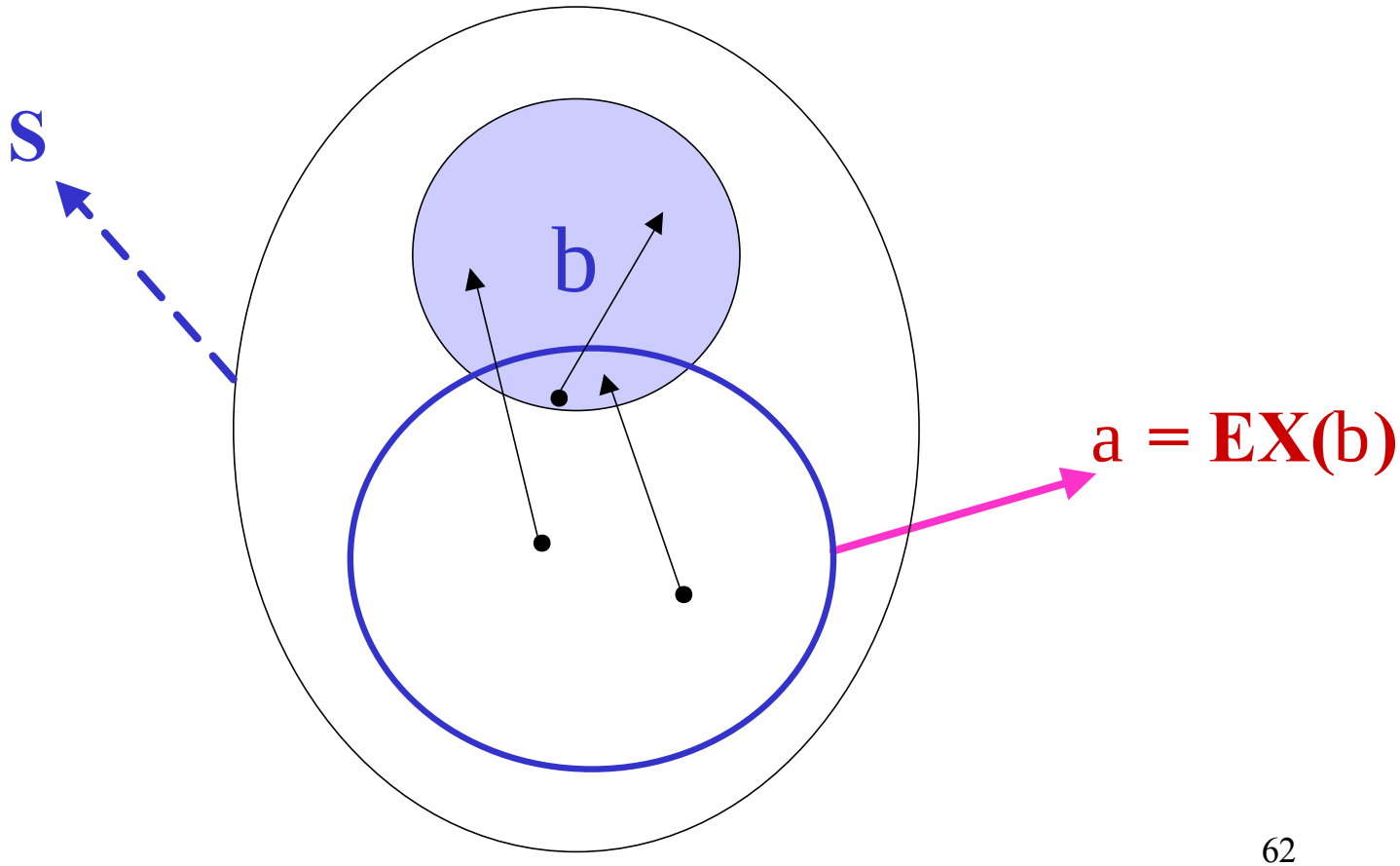
- **Stage i +1**:
  - For every **t** Î **S**:
  - If a = b_1 Ú b_2 then

    a Î **Labels(t)** ***iff*** b_1 Î **Labels(t)** **or** b_2 Î **Labels(t)**
  - If a = **EX**b then

    a Î **Labels(t)** ***iff*** **there exists s Î S such that**

    b Î **Labels(s)** and **R(t, s)**

The Labels Function



Computing the labeling for EX(β)

Complexity: O(|M|)

Algorithm Check_EX(β)

T := {**s** | b Î **Labels(s)**};

while **T** ¹ Æ do

choose **s** Î **T**;

T := **T** \{**s**};

forall **t** Î **S** such that (**t**,**s**) Î **R** do

Labels(t) := **Labels(t)** È {**EX** β};

The Labels Function

- **Stage i +1**:
  - For every **t** Î **S**:
  - If a = **EU**(b_1,b_2) then

    a Î **Labels(t)** *iff*

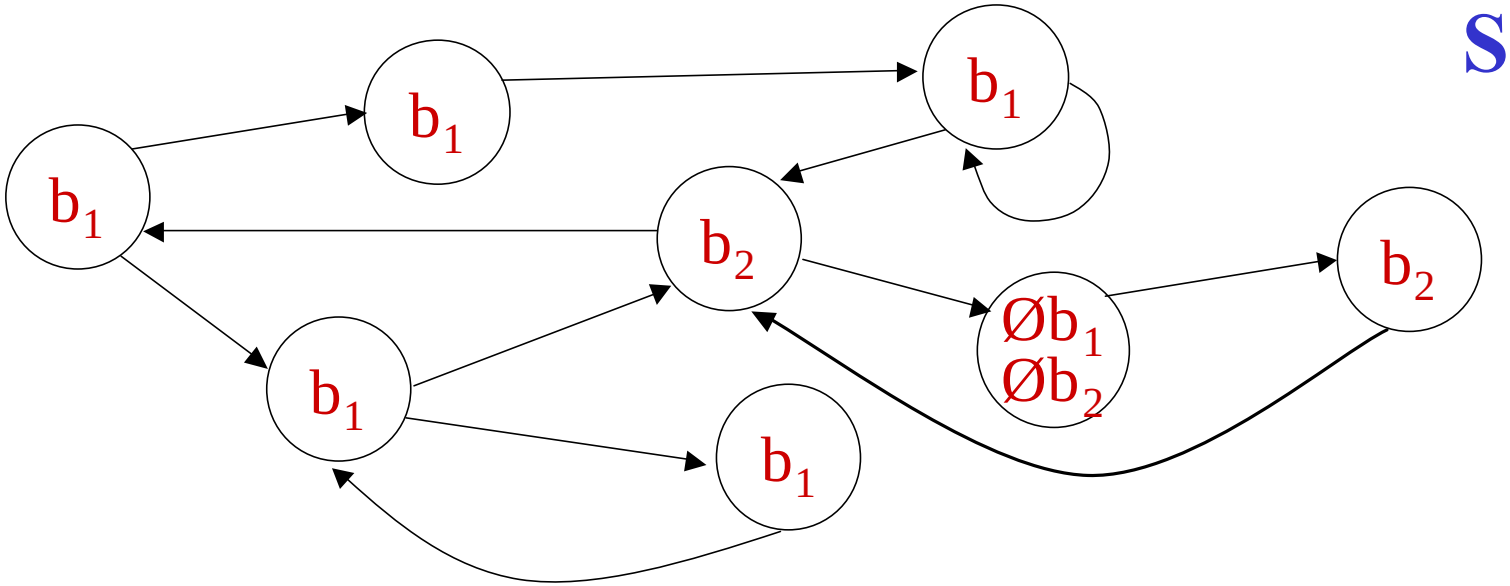
    - b_2 Î **Labels(t)** **or**
    - b_1 Î **Labels(t)** **and** **EU**(b_1,b_2) Î **Labels(s)** **for some s with R(t,s).**

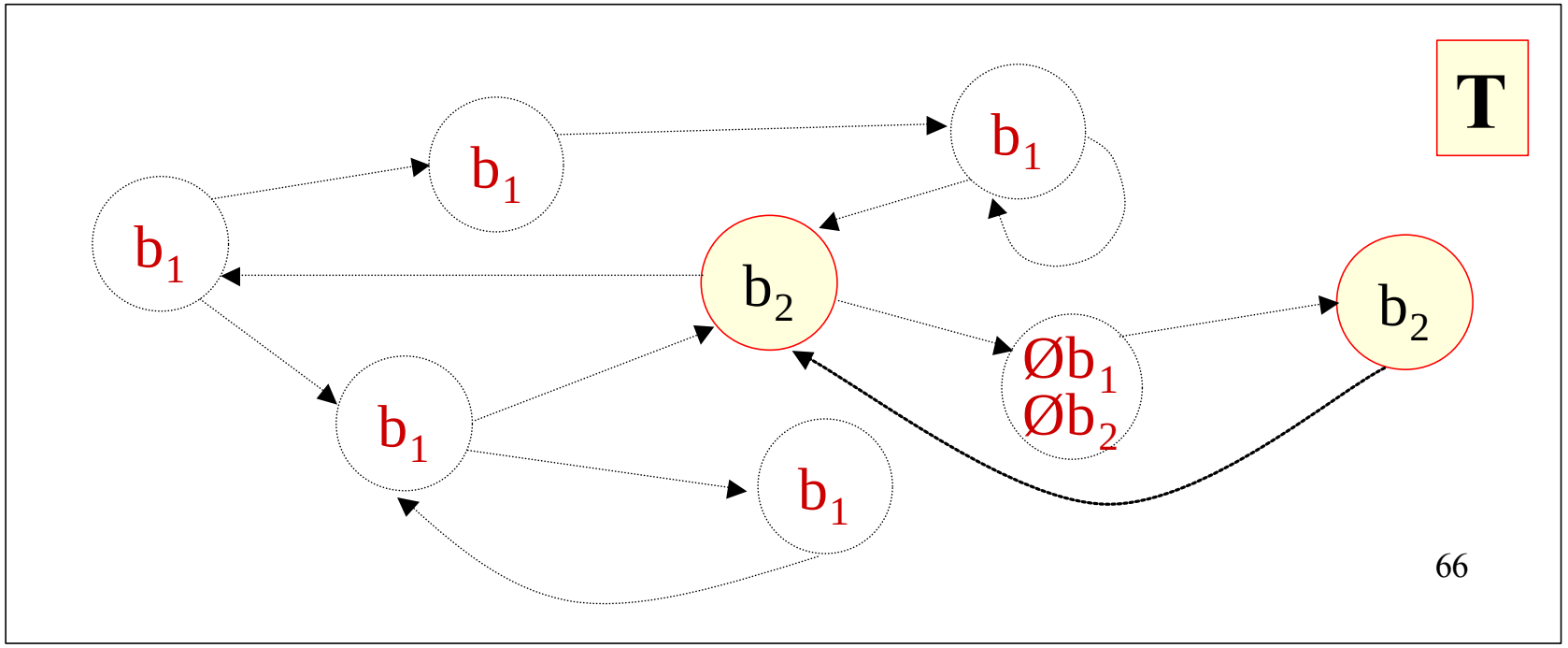
The Labels Function

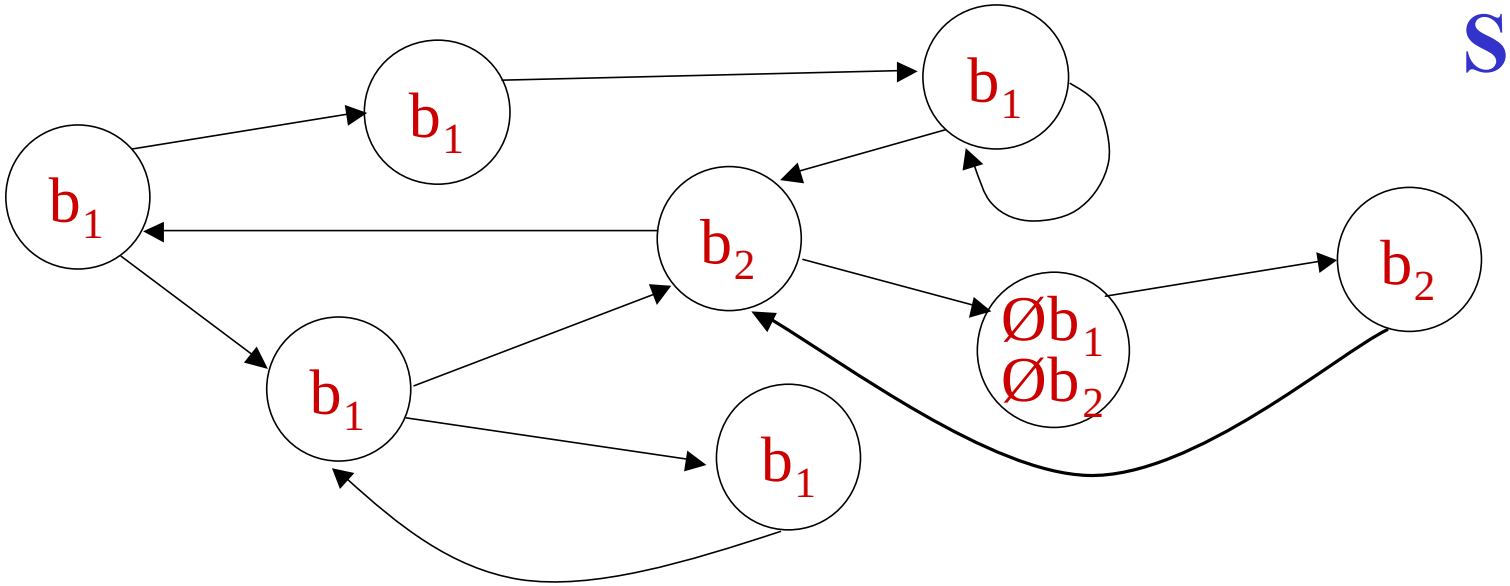
- Collect in **T** all the states satisfying b_2
  - **all these states do also satisfy EU(b_1,b_2)**.
- Traverse backward **R** from states in **T** and label with **EU(b_1,b_2)** all the states **t** satisfying b_1 and reaching at least a state **s** labeled with **EU(b_1,b_2)**.

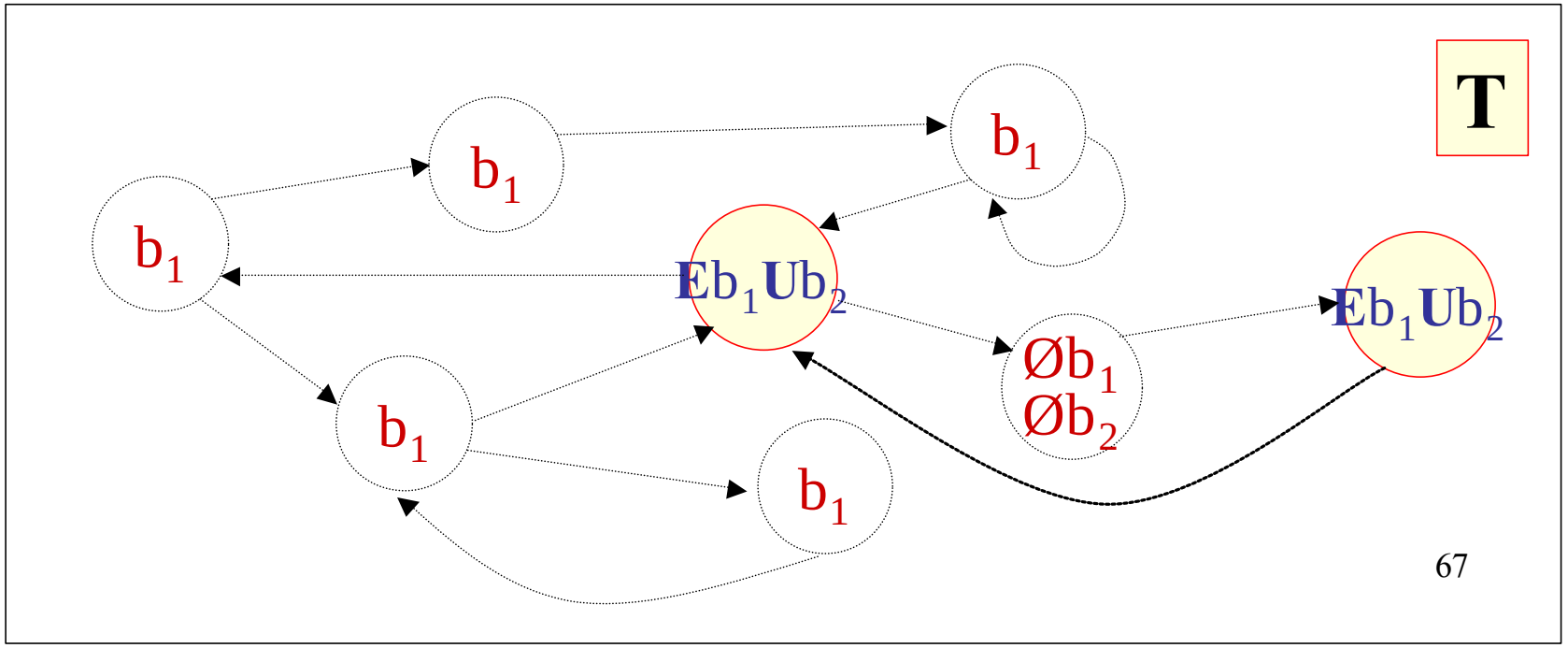
If s Î T, t with R(t,s) and b_1 Î Labels(t)

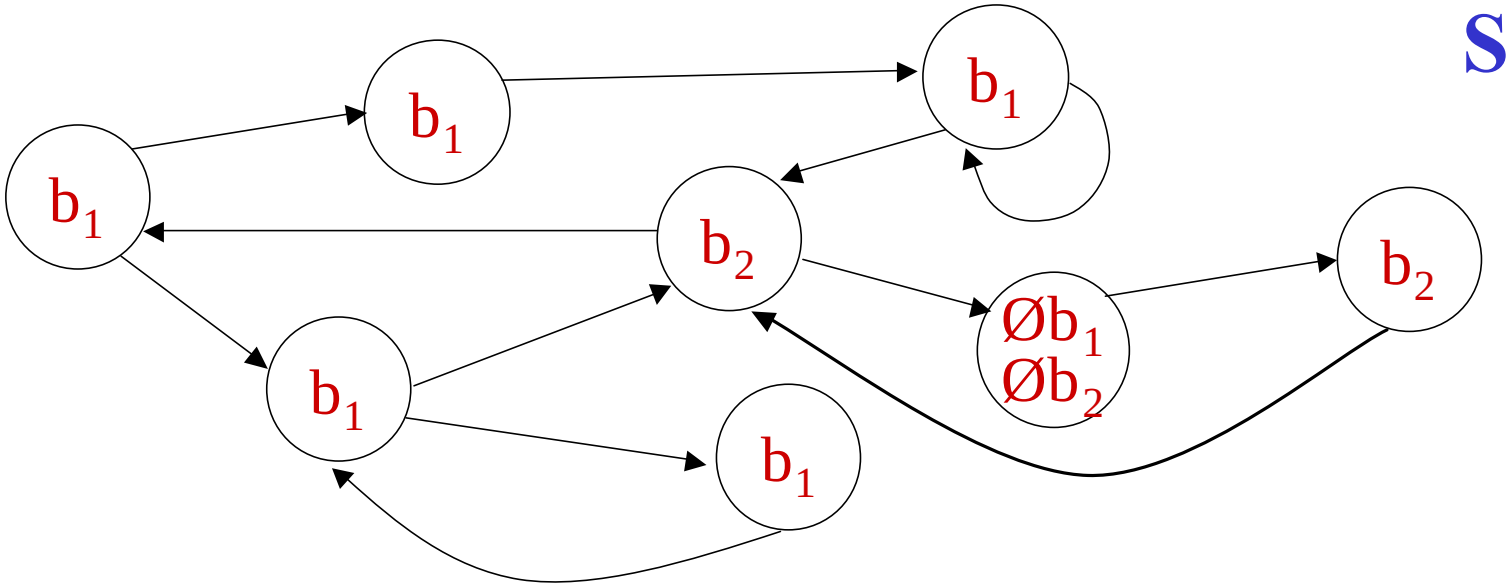
then EU(b_1,b_2) Î Labels(t)

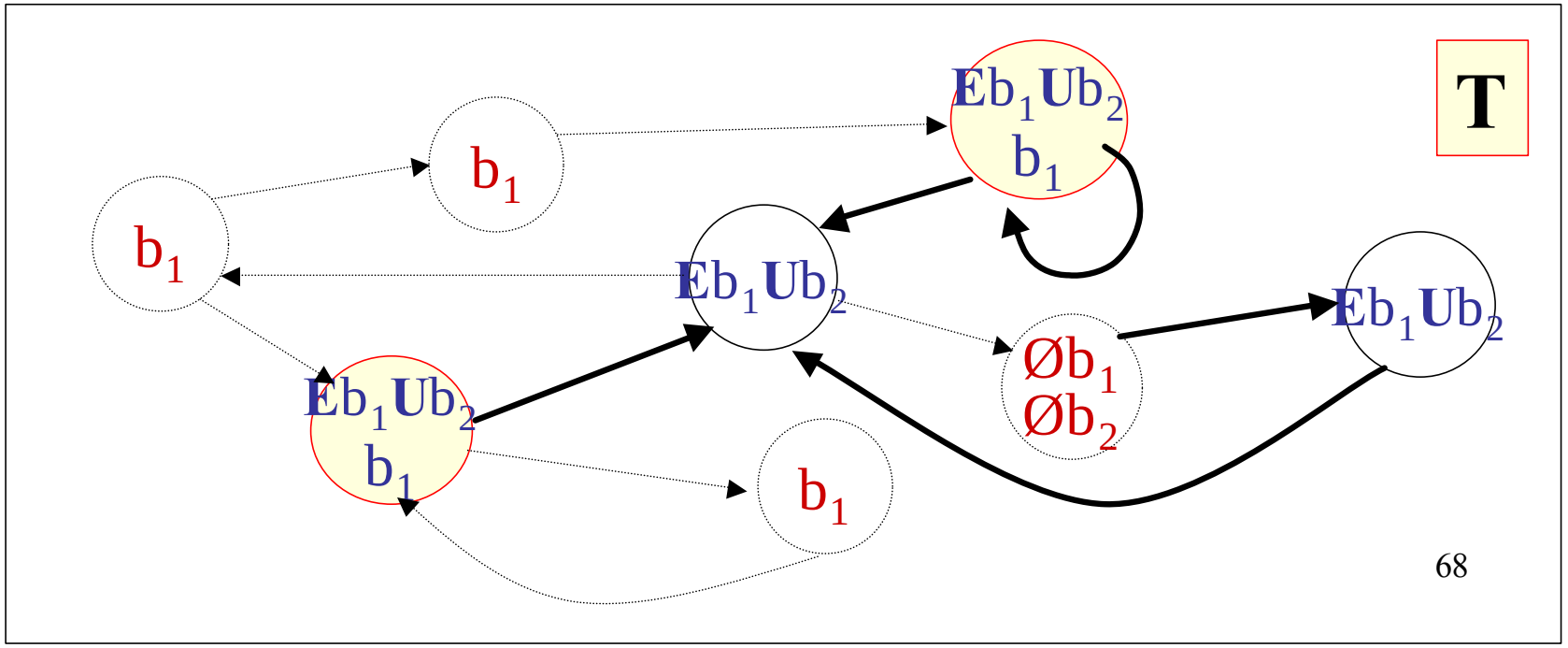

S
b1
b1
b1
b2
b2
Øb1
Øb2
b1
b1

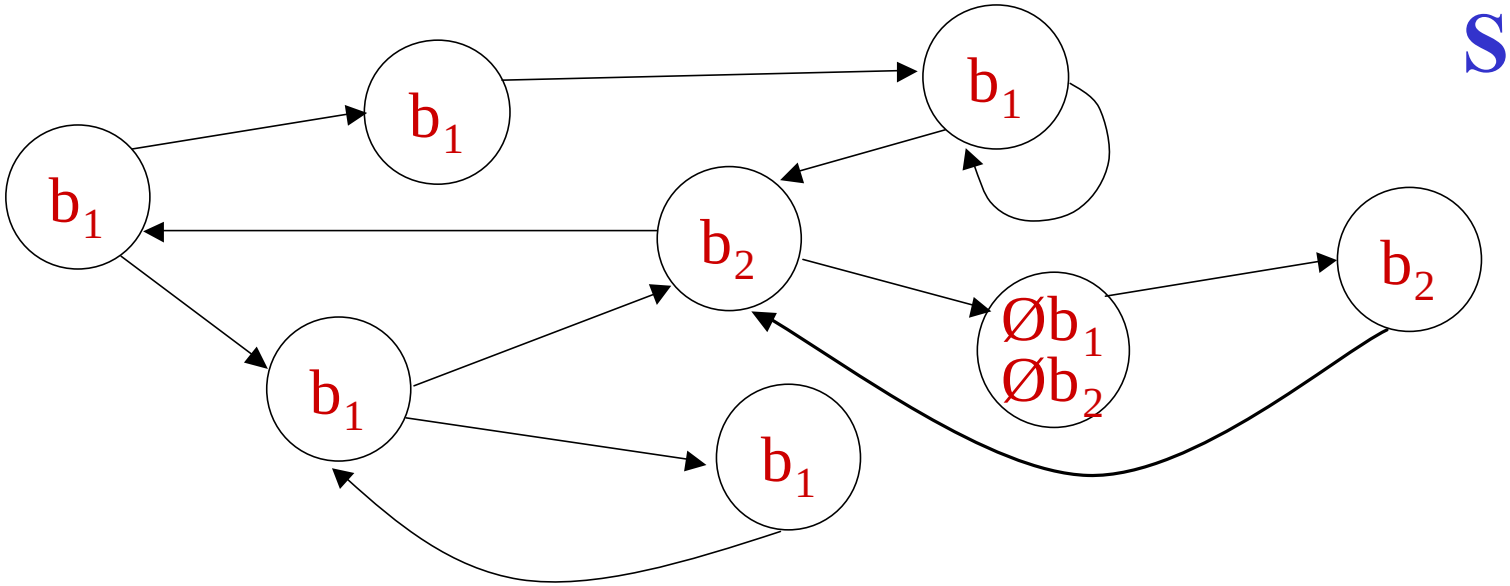

T
b1
b1
b1
b2
b2
Øb1
Øb2
b1
b1

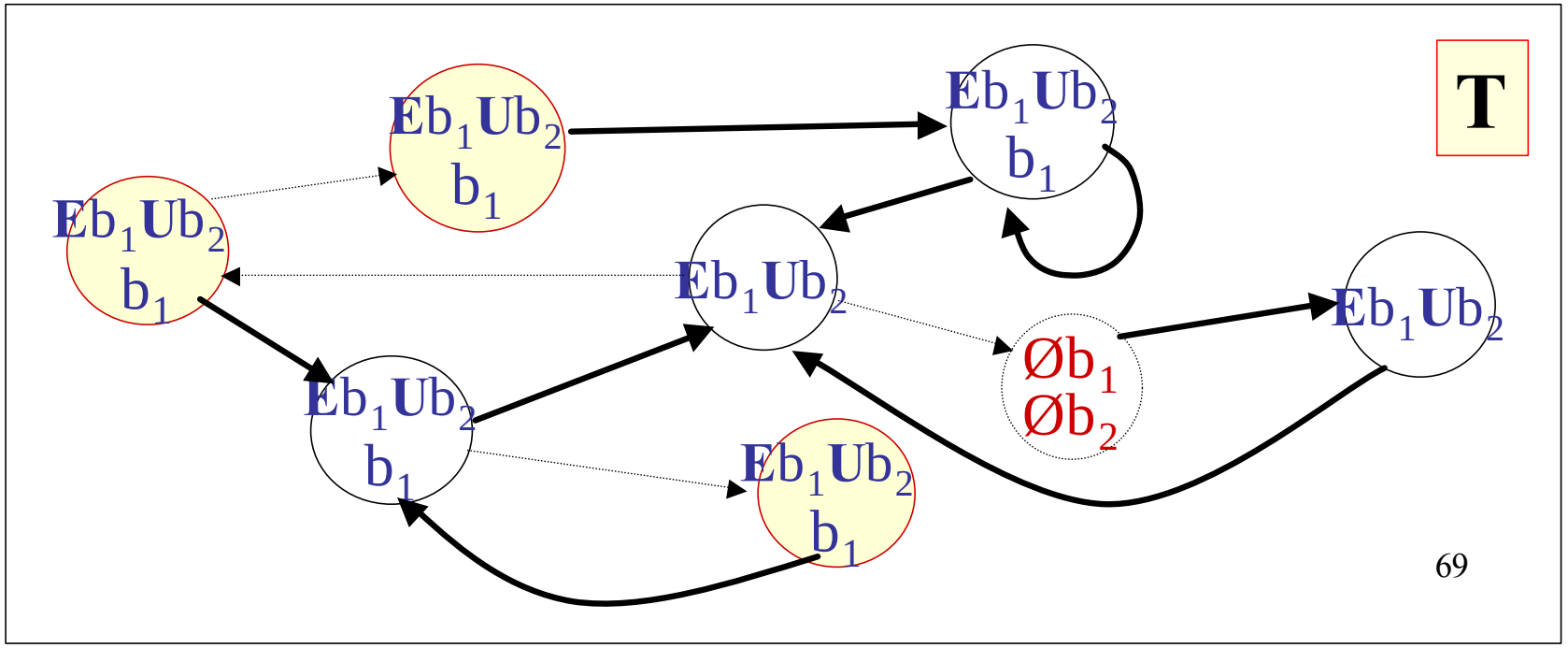

S
b1
b1
b1
b2
b2
Øb1
Øb2
b1
b1

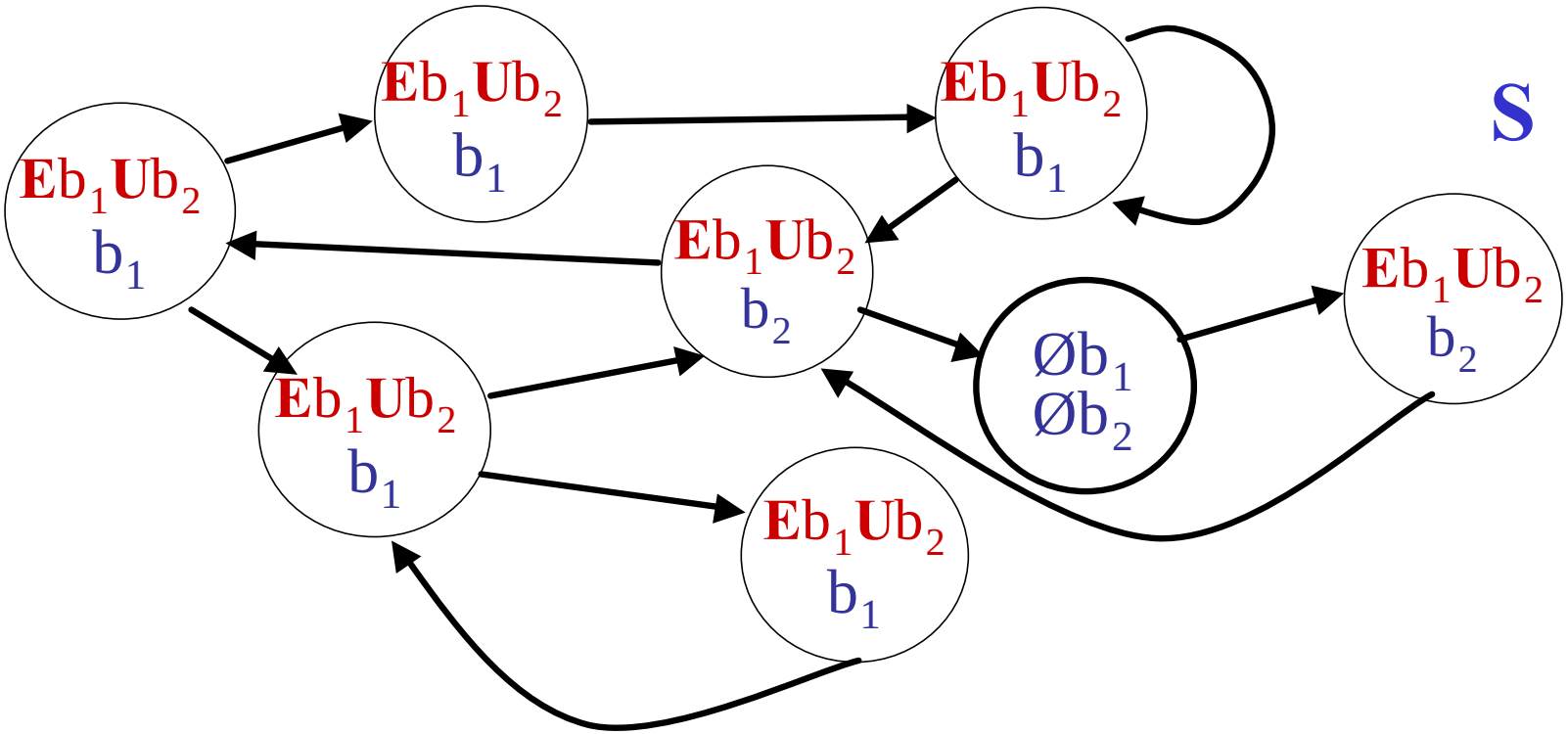

T
b1
b1
b1
Eb1Ub2
Eb1Ub2
Øb1
Øb2
b1
b1


S
b1
b1
b1
b2
b2
Øb1
Øb2
b1
b1


T
Eb1Ub2
b1
b1
b1
Eb1Ub2
Eb1Ub2
Øb1
Øb2
Eb1Ub2
b1
b1


S
b1
b1
b1
b2
b2
Øb1
Øb2
b1
b1


T
Eb1Ub2
b1
Eb1Ub2
b1
Eb1Ub2
b1
Eb1Ub2
Eb1Ub2
Øb1
Øb2
Eb1Ub2
b1
Eb1Ub2
b1


Eb1Ub2
b1
Eb1Ub2
b1
Eb1Ub2
b1
S
Eb1Ub2
b1
Eb1Ub2
b2
Øb1
Øb2
Eb1Ub2
b2
Eb1Ub2
b1
Eb1Ub2
b1

Computing the labeling for $EU(\beta_1,\beta_2)$

Algorithm Check_$EU(\beta_1,\beta_2)$

T := {**s** | β_2 Î Labels(**s**)};

forall **s** Î **T** do

Labels(**s**) := Labels(**s**) È {**EU**(β_1,β_2)};

while **T** ¹ Æ do

chose **s** Î **T**;

T := **T** \{**s**};

forall **t** Î **S** with (**t**,**s**) Î **R** do

if **EU**(b_1,b_2) Ï Labels(**t**) and β_1 Î Labels(**t**) then

Labels(**t**) := Labels(**t**) È {**EU**(β_1,β_2)};

T := **T** È {**t**};

Complexity: O(|M|)

The Labels Function

- **Stage i +1**:
  - For every **t** Î **S**:
  - If a = **EG**(b) then

    a Î **Labels(t)** ***iff***

    - b Î **Labels(t)** **and** **EG**(b) Î **Labels(s)** **for some s with R(t,s).**

Property of EG(b)

Let **M' = (S',R',L')** be the sub-graph of **M** where

- **S' = { s | M ,s ⊨ b }**
- **R' = $R|_{S' \times S'}$** (the restriction of **R** to **S'**)
- **L' = $L|_{S'}$** (the restriction of **L** to **S'**)

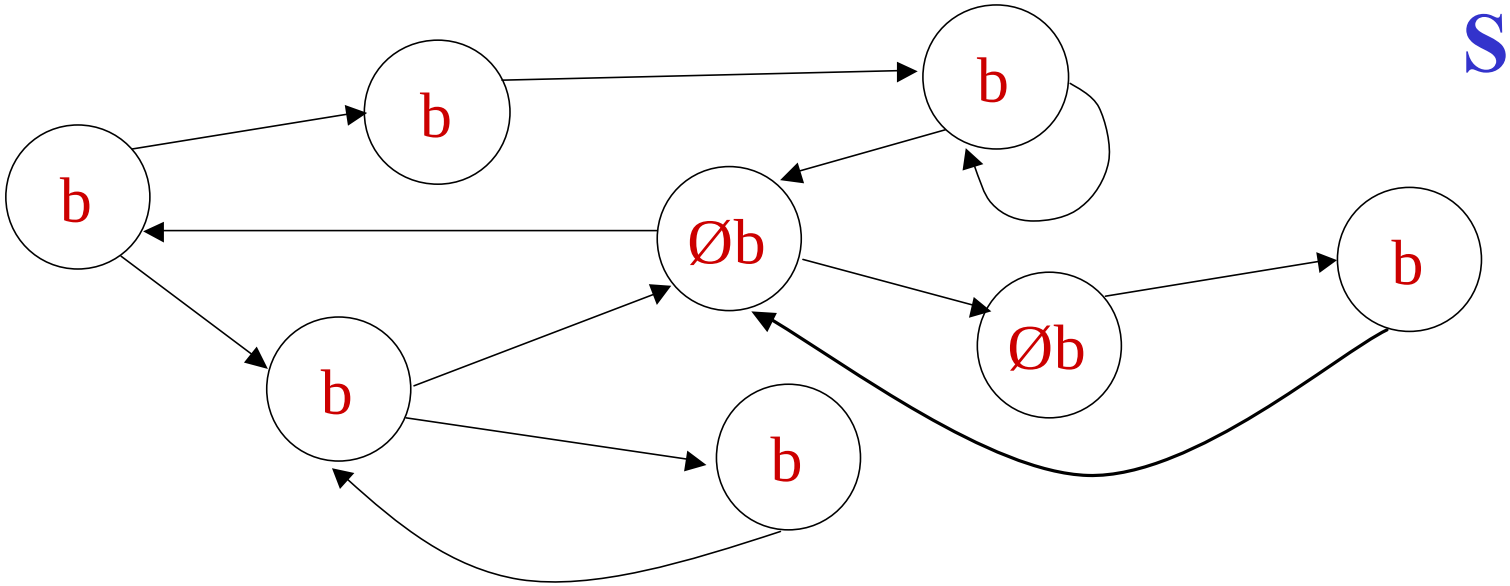
Lemma: M,s ⊨ EG(b) *iff*

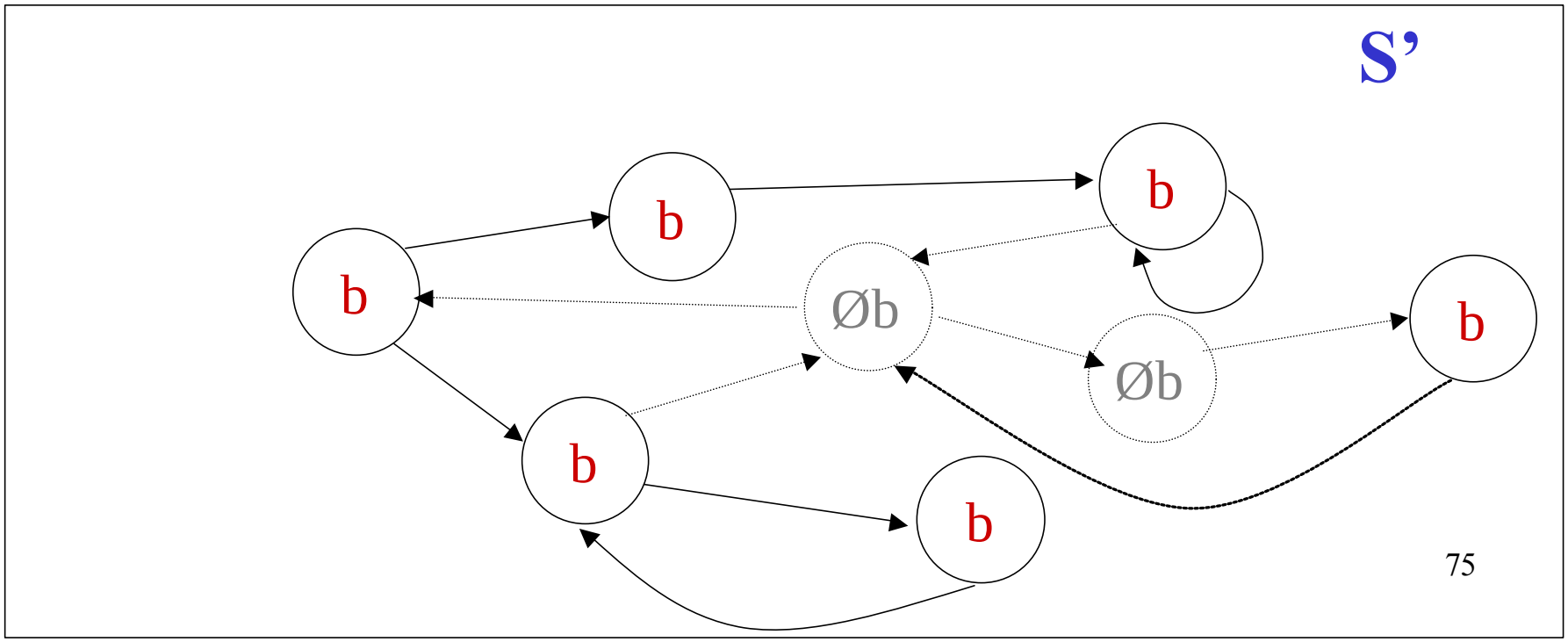
1. **s Î S' and**
2. ***there exists a path*** in **M'** leading from **s** to a ***non-trivial strongly connected component*** **C** of the graph **(S',R').**

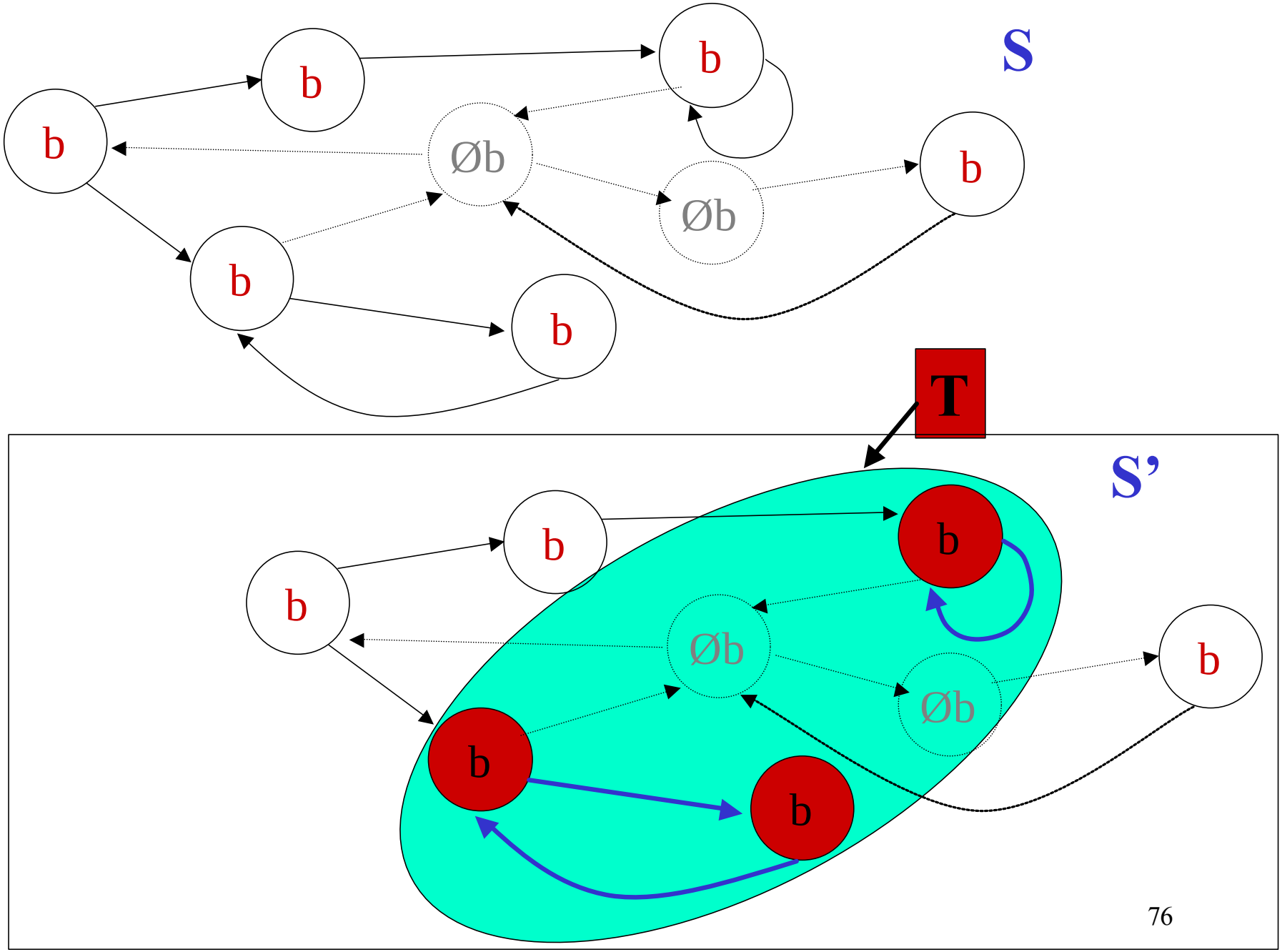
The Labels Function

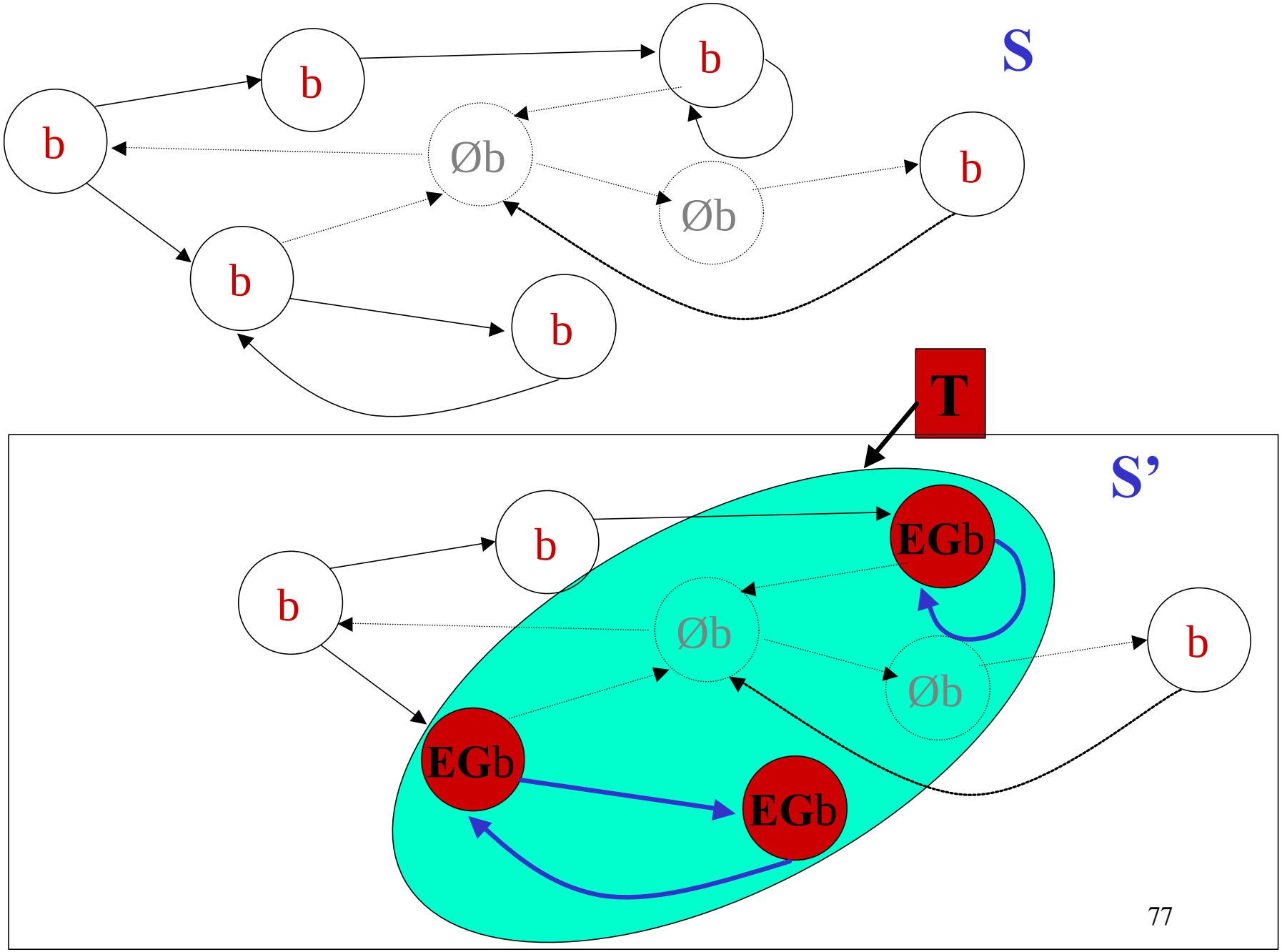
- Compute the ***non-trivial strongly connected components*** of the subgraph **S'** whose states all satisfy b
  - **all the states in these components do satisfy EG(b).**
- Traverse backward **R** and label with **EG(b)** the ***states t reaching at least a state s*** labeled with **EG(b)** (note that both **t** and **s** must belong to **S'**).

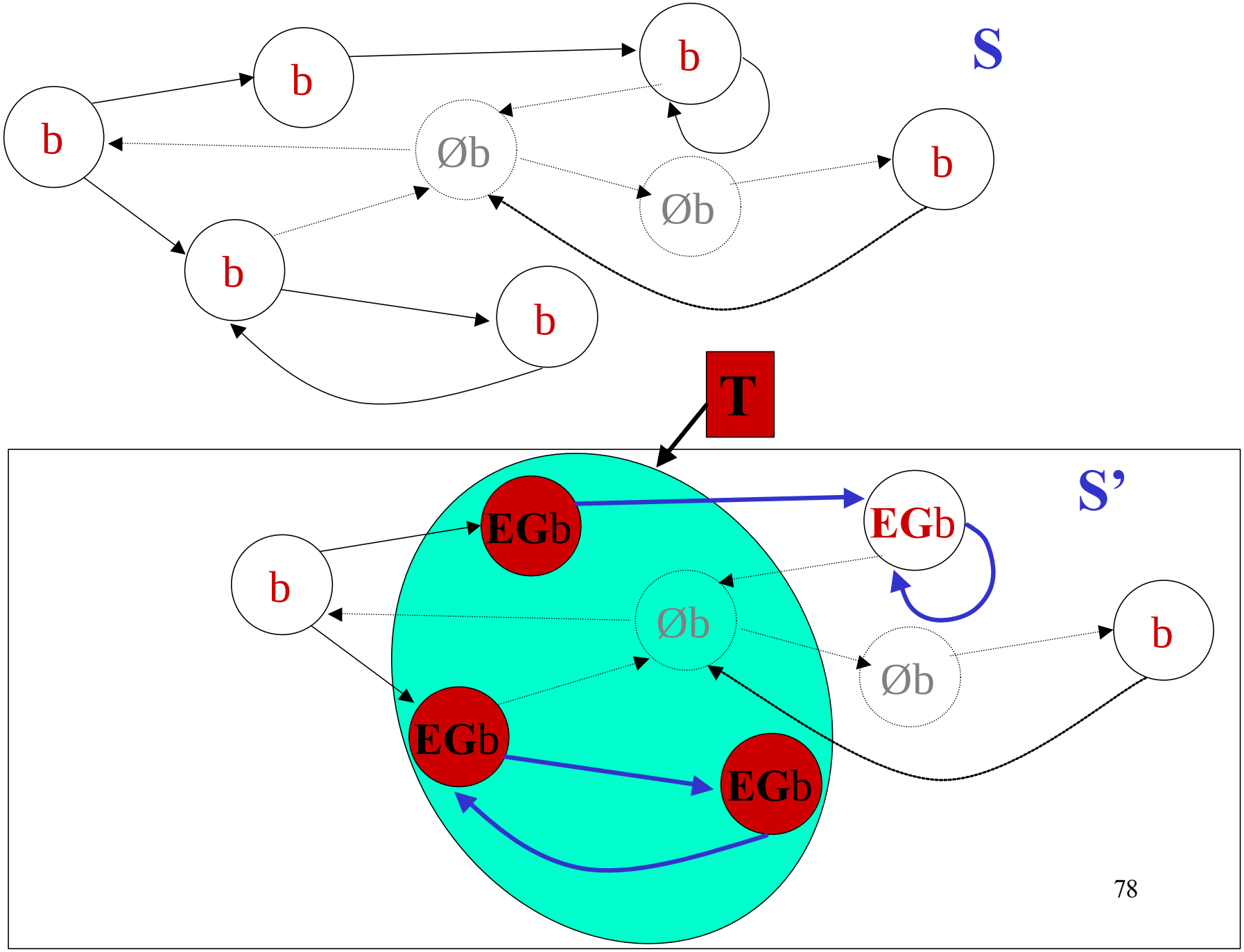
If t Î S' and R(t,s) then EG(b) Î Labels(t)

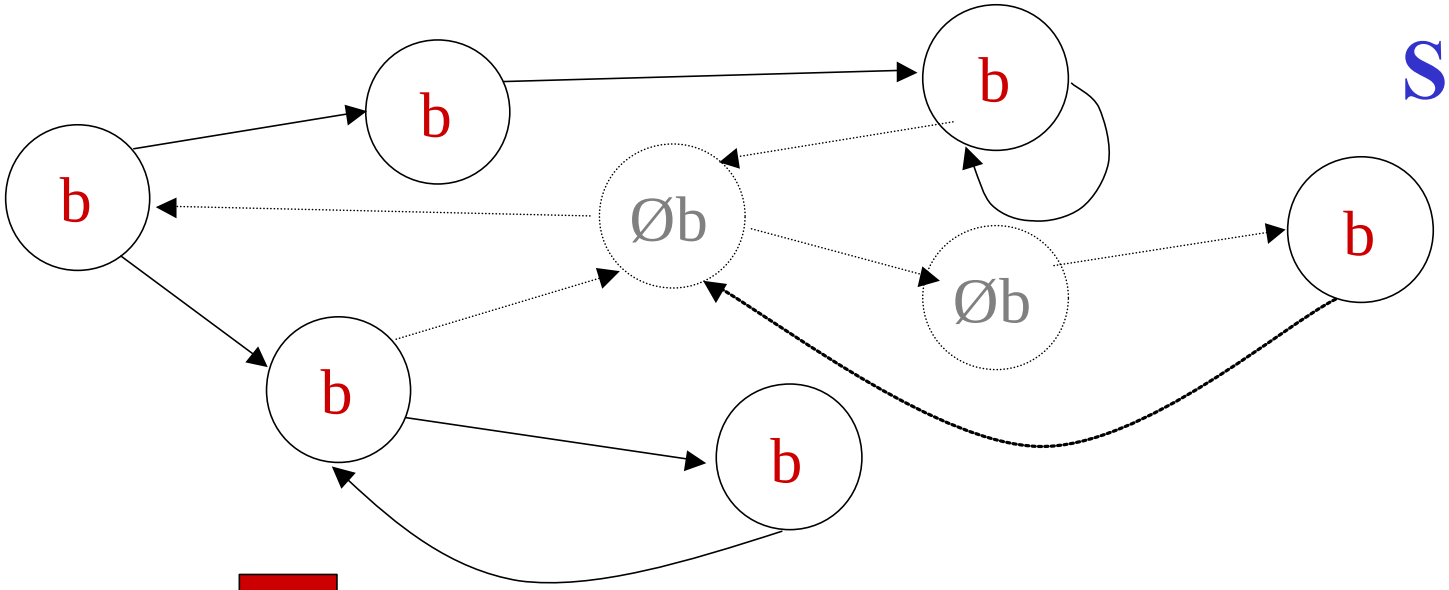

S
b
b
b
Øb
b
Øb
b
b
b

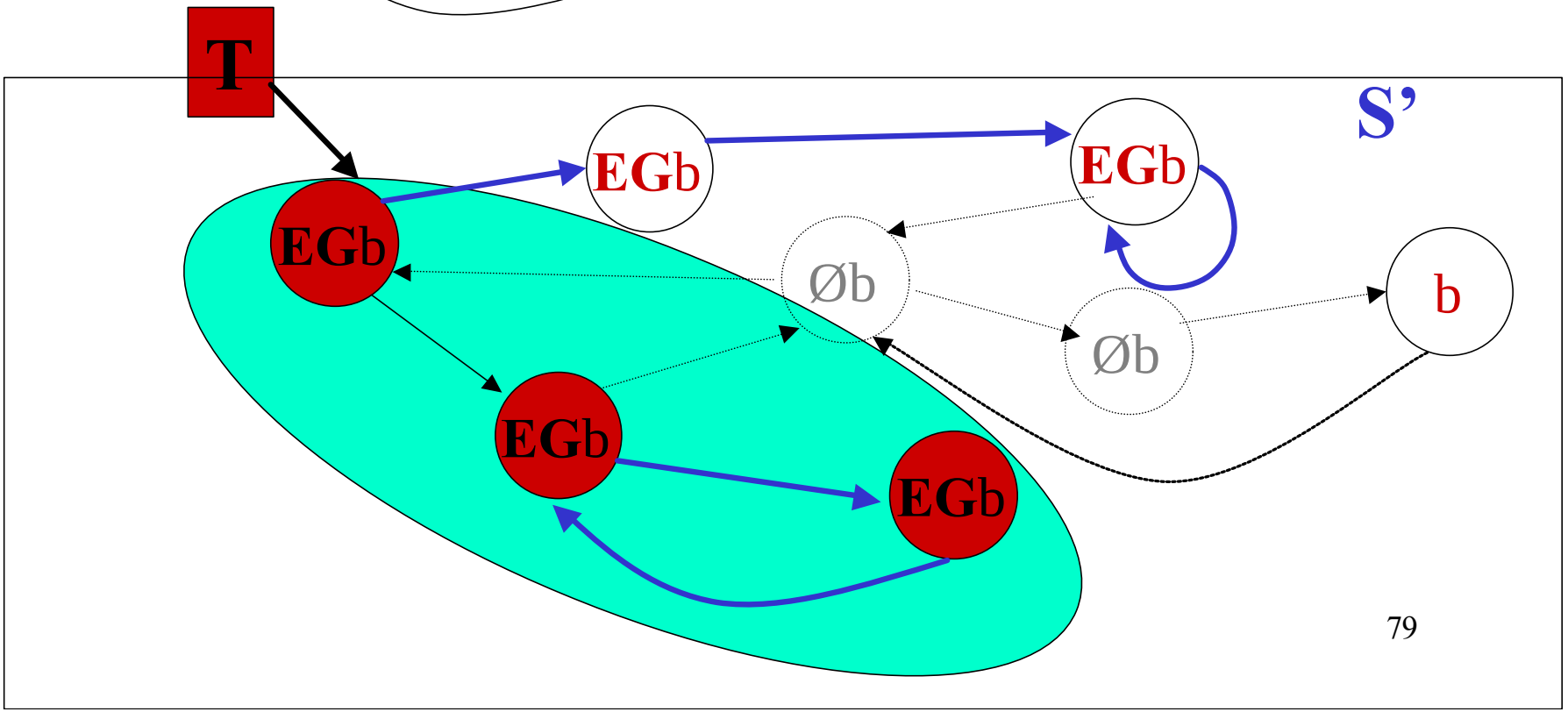

S'
b
b
b
Øb
b
Øb
b
b
b

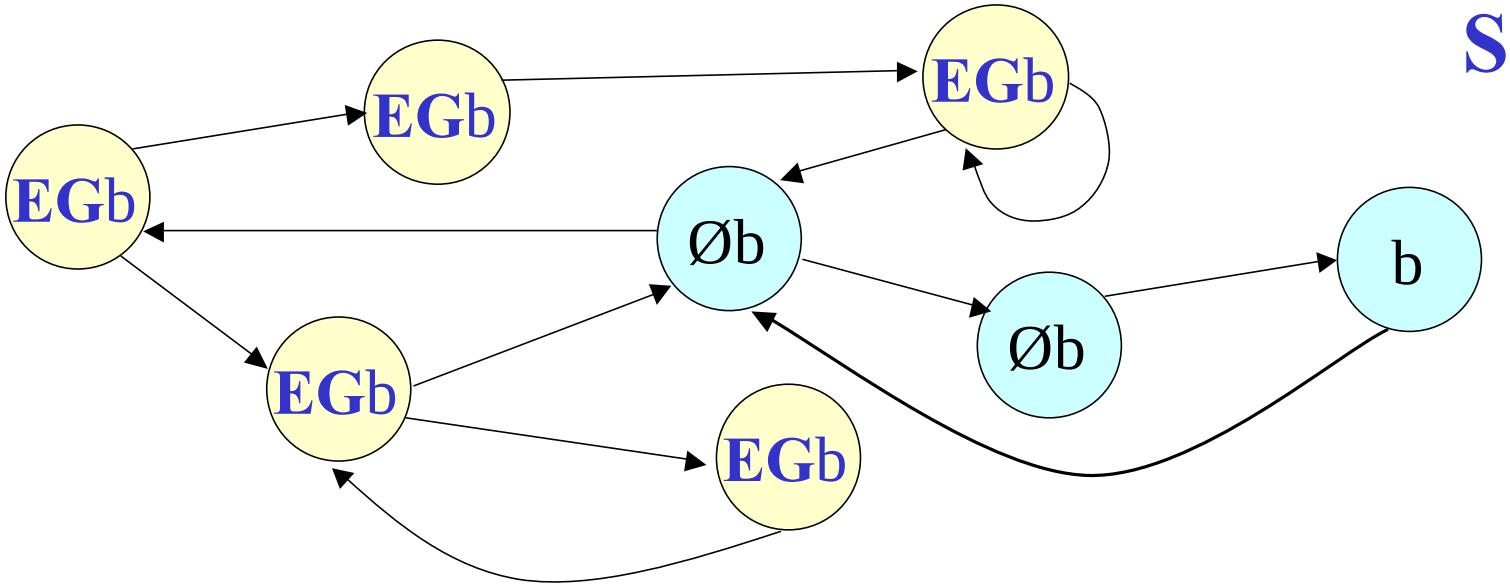

S
b
b
b
Øb
b
Øb
b
b
b
T
S'
b
b
b
Øb
b
Øb
b
b
b

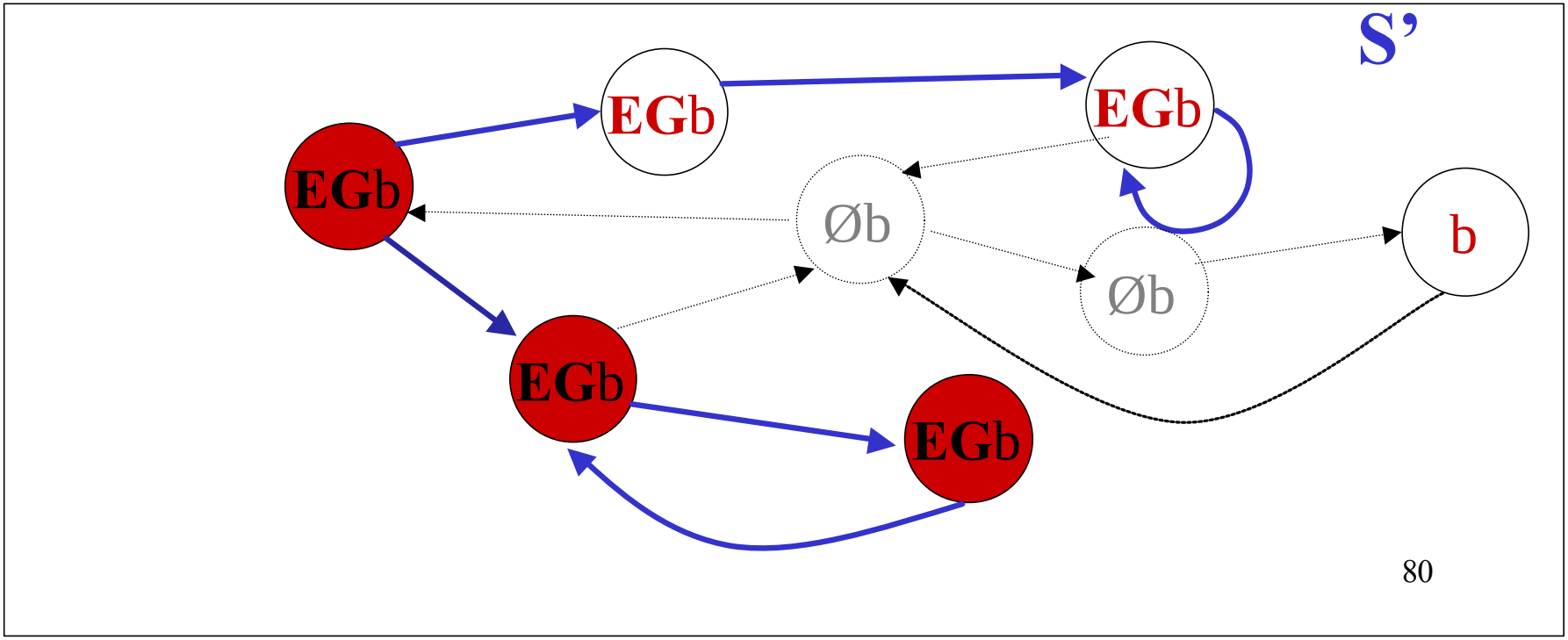

S
b
b
b
Øb
b
Øb
b
b
b
T
S'
b
b
EGb
Øb
Øb
b
EGb
EGb


S
b
b
b
Øb
b
Øb
b
b
b
T
S'
EGb
EGb
b
Øb
b
Øb
EGb
EGb


S
b
b
b
b
Øb
Øb
b
b
b


T
S'
EGb
EGb
EGb
EGb
EGb
Øb
Øb
b


S
EGb
EGb
EGb
Øb
b
Øb
EGb
EGb


S'
EGb
EGb
EGb
Øb
b
Øb
EGb
EGb

Computing the labeling for EG(β)

Algorithm Check_EG(β)

Complexity: O(\|M\|)

```
S' := {s | β Î Labels(s)};
SCC := {C | C is a non trivial SCC of S'};
T := È_{C Î SCC} {s | s Î C};
forall s Î T do Labels(s) := Labels(s) È {EG(β)};
while T ¹ Æ do
        chose s Î T;
        T := T \{s};
        forall t Î S' with (t,s) Î R do
            if EG(β) Ï Lables(t) then
                Labels(t) := Labels(t) È {EG(β)};
                T := T È {t};
```

CTL model checking

- The algorithms just presented show that the ***model checking problem*** for ***CTL*** can be solved in ***time linear*** in the size of System **M** and the size of the Property f, namely:

in time **O(|M|×|f|)**

where **|M|** is the ***size of the graph*** underlying **M** and **|f|** is the ***number of subformulae*** of f.

Fixed point characterization

- We will redefine the labeling function in terms of ***fixed point computation***.
- This is a ***nice*** and ***elegant*** algorithmic account.
- It will be used when ***efficient symbolic approach*** will be introduced.

Partial Orders

- A binary relation $\sqsubseteq$ on a set **A** is a ***partial order*** iff $\sqsubseteq$ is ***reflexive***, ***anti-symmetric*** and ***transitive***.
- The pair $\langle A, \sqsubseteq \rangle$ is called a ***partially ordered set*** (or ***poset***).
- **Example:** If **S** is any set and $\subseteq$ is the ordinary subset relation, then $\langle 2^S, \subseteq \rangle$ is a *partially ordered set*.

Upper Bounds

Given **<A, ⊑>** and **A' Í A**

- **a Î A** is an ***upper bound*** of **A'** iff **" a' Î A', a' ⊑ a**
- **a Î A** is a ***least upper bound*** (***lub***) of **A'**, written **⊔A'**, iff
  - **a** is an ***upper bound*** of **A'** and
  - **" a' Î A,** if **a'** is an ***upper bound*** of **A'**, then **a ⊑ a'**

Lower Bounds

Given **<A, ⊑>** and **A' Í A**

- **a Î A** is a ***lower bound*** of **A'** iff **" a' Î A', a ⊑ a'**
- **a Î A** is a ***greatest lower bound*** (***glb***) of **A'**, written **⊓A'**, iff
  - **a** is a ***lower bound*** of **A'** and
  - **" a' Î A,** if **a'** is a ***lower bound*** of **A'**, then **a' ⊑ a**

Complete Lattice

A ***poset*** **<A, ⊑>** is a ***complete lattice*** if, for each **A'** Í **A**, the ***greatest lower bound*** **⊓A'** and the ***least upper bound*** **⊔A'** do exist.

A ***complete lattice*** **<A, ⊑>** has a unique ***greatest element*** **⊔A=⊤** and also a unique ***least element*** **⊓A = ^**.

Complete Lattice

The ***poset*** $<2^S, \subseteq>$ is a ***complete lattice*** where ***intersection*** $\cap$ and ***union*** $\cup$ correspond to $\sqcap$ and $\sqcup$, respectively.

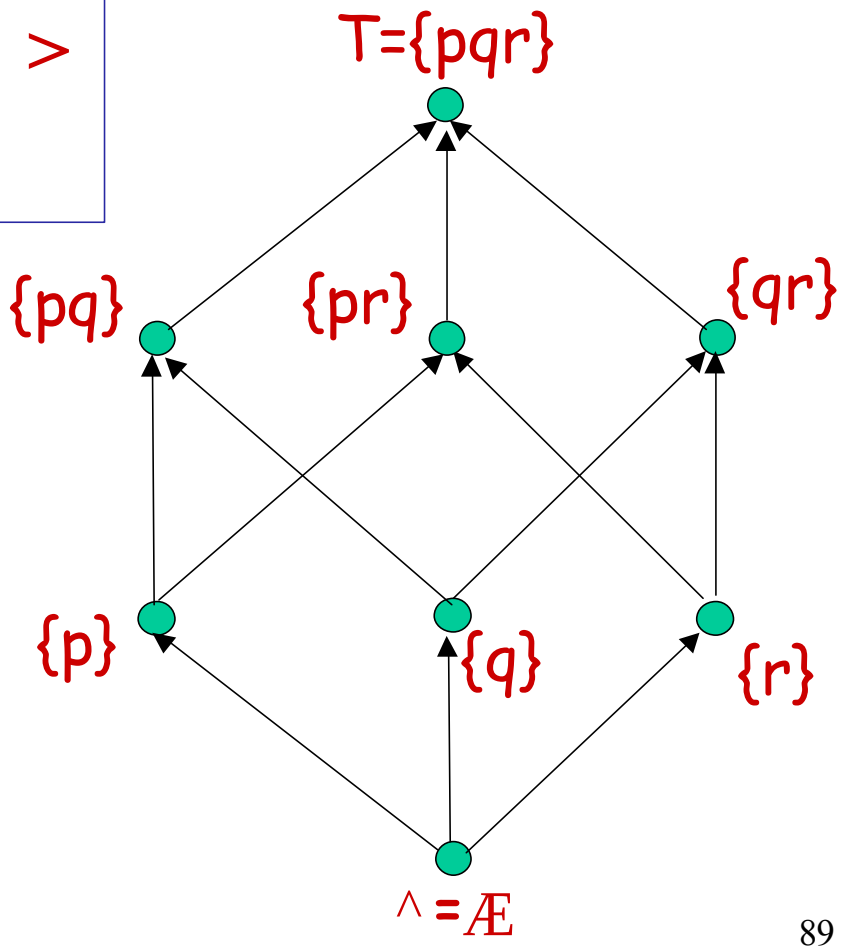
Any two subset of **S** have a ***least upper*** and a ***greatest lower bound***.

Example: **S={a,b,c,d}**. For **{a,c}** and **{b,c}** the ***lub*** is **{c}**, while the ***glb*** is **{a,b,c}**.

There is a unique ***greatest element*** $\cup 2^S = S$ and a unique ***least element*** $\cap 2^S = \emptyset$.

Example of a complete lattice

The complete lattice $\langle 2^S, \subseteq \rangle$ when S is the set {p,q,r}.



Monotonic functions

- A function **F: A ⟶ A** is ***monotonic*** if for each **a,b Î A**, **a ⊑ b** implies **F(a) ⊑ F(b)**.
- In other words, a function **F** is monotonic if it ***preserves the ordering*** ⊑.

Fixed points

- Given a function **F: A ⟶ A**, an element **a Î A** is a ***fixed point*** of **F** if **F(a) = a**.
- **a Î A** is called the ***least fixed point*** of **F** (**mx.F(x)**), if for all **a' Î A** such that **F(a') = a'**, then **a ⊑ a'**.
- **a Î A** is called the ***greatest fixed point*** of **F** (**nx.F(x)**), if for all **a' Î A** such that **F(a') = a'**, then **a' ⊑ a**.

Tarski's Fixed Point theorem

THEOREM: Let **<A, ⊑>** be a ***complete lattice***, and **F: A ⟶ A** a monotonic function. Then **F** has a ***least*** and a ***greatest fixed point*** given, respectively, by:

- **μx.F(x) = ⊓{x Î A | F(x) ⊑ x}**
- **νx.F(x) = ⊔{x Î A| x ⊑ F(x)}**

Fixed point in finite lattices

Let $\langle A, \sqsubseteq \rangle$ be a ***finite complete lattice***, and $F: A \to A$ be a monotonic function.

The ***least element*** of **A**

Then the ***least fixed point*** for **F** is obtained as

$$\mu x.F(x) = F^m(\bot)$$

for some **m**, where $F^0(\bot) = \bot$, and $F^{n+1}(\bot) = F(F^n(\bot))$.

Moreover, the ***greatest fixed point*** for **F** is obtained as

$$\nu x.F(x) = F^k(\top)$$

for some **k**, where $F^0(\top) = \top$, and $F^{n+1}(\top) = F(F^n(\top))$.

The ***greatest element*** of **A**

Generic fixed point algorithm

Algorithm Compute_lfp(**F**:function)

$\mathbf{X_0}$:= ^;

$\mathbf{X_1}$:= $\mathbf{F(X_0)}$;

j=1;

while $\mathbf{X_j}$ ^{1} $\mathbf{X_{j-1}}$

$\mathbf{j := j+1}$;

$\mathbf{X_j := F(X_{j-1})}$;

return $\mathbf{X_j}$

CTL and complete lattices

- Given a Kripke structure $\mathbf{M=<S,S_0,R,L,AP>}$. We will then consider the ***poset*** $\mathbf{<2^S, \sqsubseteq>}$.
- $\mathbf{<2^S, \sqsubseteq>}$ is clearly a ***complete lattice*** (with respect to intersection and union).
- We will identify a ***CTL formula*** with the ***set of states*** which ***satisfy it***.
- In this way we can define ***temporal operators*** as ***functions*** on the ***complete lattice*** $\mathbf{<2^S, \sqsubseteq>}$.

Denotation of a CTL formula

- Given a formula f, let us define its ***denotation*** (in **M**), in symbols |[f]|, as the set of states satisfying the formula:

$$|[f]| = \{ s \mid M,s \vDash f \}$$

- We could then define the cpo **<*CTL*, ⊑ >** by:

$$f \sqsubseteq y \quad \textit{iff} \quad |[f]| \subseteq |[y]|$$

Denotation of a CTL formula

- Given the ***denotation*** of a formula

$$|[f]| = \{ s \mid M,s \models f \}$$

- We could then define the cpo $\langle CTL, \sqsubseteq \rangle$ by:

$$f \sqsubseteq y \quad \textit{iff} \quad |[f]| \subseteq |[y]|$$

- Then $|[\wedge]| = \mathcal{P}$; $|[\top]| = S$;
- $|[p]| = \{ s \mid p \in L(s) \}$;
- $|[\neg f]| = S \setminus |[f]|$;
- $|[f \vee y]| = |[f]| \cup |[y]|$;
- $|[f \wedge y]| = |[f]| \cap |[y]|$;

CTL is closed under ***conjunction*** and ***disjunction***, therefore for any pair of formulae the ***upper*** and ***lower bound*** do exist.

Denotation of a CTL formula

- Given a formula f, let us define its ***denotation*** (in **M**), in symbols **|[f]|**, as the set of states satisfying the formula:

$$|[f]| = \{ s \mid M,s \vDash f \}$$

- ….
- **|[EX f]| = { s | $t. (t Î |[f]| Ç R(s)) }**
- for the other **temporal operators** we would need to use **fixed points….**

Fixed point characterization of $\mathbf{EU(b_1,b_2)}$

- $\mathbf{EU(b_1,b_2)} \circ b_2 \acute{U} (b_1 \grave{U} \mathbf{EX\ EU(b_1, b_2)})$
- $|[\mathbf{EU(b_1,b_2)}]| = \mu\mathbf{Z.}(|[b_2]| \grave{E} (|[b_1]| \c{C} |[\mathbf{EX\ Z}]|))$
- $|[\mathbf{EU(b_1,b_2)}]| =$

  $\mu\mathbf{Z.}(|[b_2]| \grave{E} (|[b_1]| \c{C} \{ \mathbf{s} \mid \$\mathbf{t}\ \hat{I}\ \mathbf{Z} \c{C} \mathbf{R(s)} \}))$

Fixed point characterization of $\mathbf{EU(b_1,b_2)}$

Lemma: Let

$$\mathbf{F(Z) = (|[b_2]| \cup (|[b_1]| \cap \{ s \mid \$t \in Z \cap R(s) \}))}$$

then **F** is a ***monotonic function***, i.e.

$$\mathbf{Z_1 \subseteq Z_2 \text{ implies } F(Z_1) \subseteq F(Z_2)}$$

Fixed point characterization of $\mathbf{EU(b_1,b_2)}$

Theorem:

$$|[\mathbf{EU}(b_1,b_2)]| = \mu Z.(|[b_2]| \cup (|[b_1]| \cap \{ s \mid \exists t \in Z \cap R(s) \}))$$

in other words:

$$\mu Z.(|[b_2]| \cup (|[b_1]| \cap \{ s \mid \exists t \in Z \cap R(s)\})) \subseteq |[\mathbf{EU}(b_1,b_2)]|$$

and

$$|[\mathbf{EU}(b_1,b_2)]| \subseteq \mu Z.(|[b_2]| \cup (|[b_1]| \cap \{ s \mid \exists t \in Z \cap R(s)\}))$$

Computing fixed point for $EU(\beta_1,\beta_2)$

Algorithm Compute_EU(β_1,β_2)

X_0 := |[^]|; /* i.e. X_0 := Æ */

X_1 := |[β_2]| È (|[β_1]| Ç X_0); /* i.e. X_1 := |[β_2]| */

j=1;

while X_j ¹ X_{j-1}

$j := j+1$; T := X_{j-1}; X := Æ

while T ¹ Æ do

chose s Î T;

T := T \{s};

forall t such that s Î R(t) do

X := X È {t};

X_j := |[β_2]| È (|[β_1]| Ç X)

This computes
$X = EX\ X_{j-1}$

Computing fixed point for $\mathbf{EU(b_1,b_2)}$

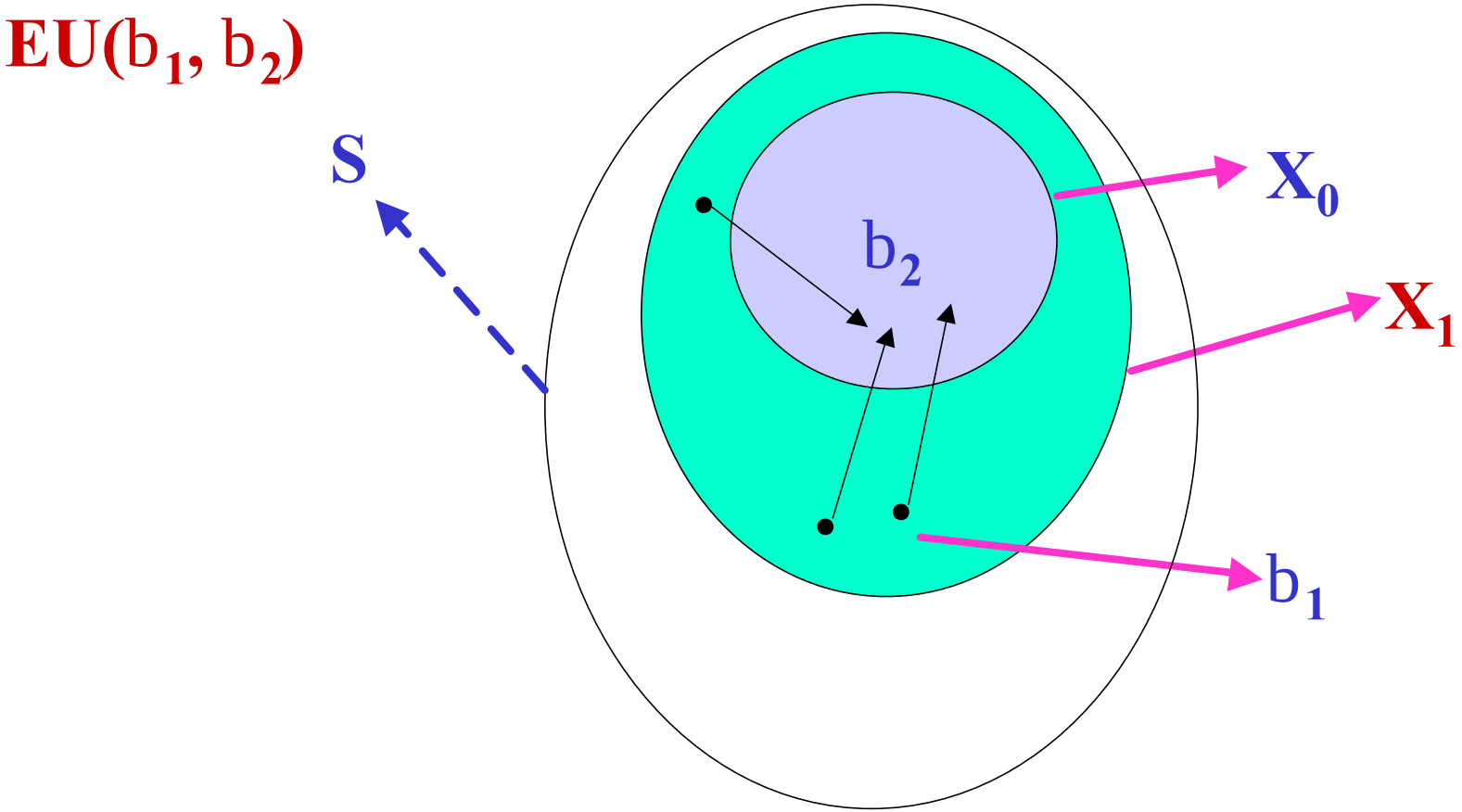
To compute $\mathbf{|[EU(b_1,b_2)]|}$ we can ***construct inductively the set*** of states $\mathbf{X_j}$ as follows:

- $\mathbf{X_1 = \{s \mid s \in |[b_2]|\}}$.
- $\mathbf{X_{j+1} = X_j \cup \{ s \mid s \in |[b_1]|}$ and $\mathbf{R(s,t)}$ for some $\mathbf{t \in X_j\}}$

$\mathbf{|[EU(b_1,b_2)]|}$ is then the set $\mathbf{X}$ such that $\mathbf{X = X_n}$ for $\mathbf{n}$ such that $\mathbf{X_{n+1} = X_n}$.

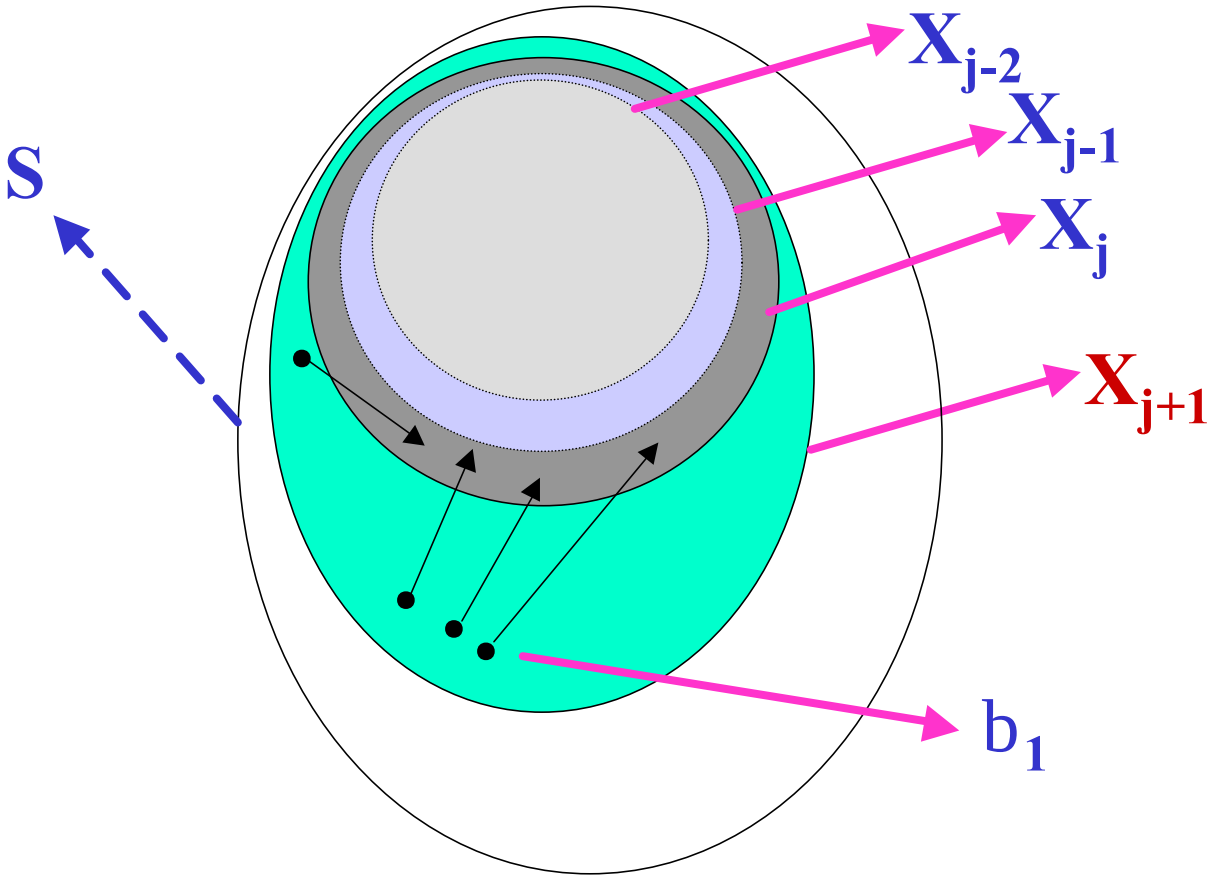
Notice that $\mathbf{n}$ ***must exist*** by ***Tarski's Theorem*** since $\mathbf{X_j \subseteq X_{j+1} \subseteq S}$ (and $\mathbf{S}$ is finite!)

From X_0 to X_1



From X_j to X_{j+1}

EU(b_1, b_2)



Computing fixed point for $EU(\beta_1,\beta_2)$

```
Algorithm Compute_EU(β1,β2)
    X1 := |[β2]|;
    j=1;
    repeat
        j := j+1; T := X := Xj-1;
        while T ¹ Æ do
            chose s Î T;
            T := T \{s};
            forall t such that s Î R(t) do
                if t Î |[β1]| then      /* t Î |[β1]| Ç EX Xj-1 */
                    X := X È {t};
        Xj = X ;
    until Xj-1 = Xj
```

Fixed point characterization of EG(β)

- **EG**(b) º b Ù **EX EG**(b)
- |[**EG**(b)]| = n**Z.**(|[b]| Ç |[**EX Z**]|))
- |[**EG**(b)]| =

  n**Z.**(|[b]| Ç { **s** | $**t** Î **Z** Ç **R**(**s**) })

Computing the fixed point for EG(β)

Algorithm Compute_EG(β)

X_0 := |[⊤]|; /* i.e. X_0 := S */

X_1 := |[β]| Ç X_0; /* i.e. X_1 := |[β]| */

j=1;

while X_j ¹ X_{j-1}

j := j+1; T := X_{j-1}; X := Æ;

while T ¹ Æ do

chose s Î T;

T := T \{s};

forall t such that s Î R(t)

X_j := X_j È {t};

X_j := |[β]| Ç X_j

X = EX X_{j-1}

The Labels function

- To compute $|[\mathbf{EGb}]|$ we can ***construct inductively the set*** of states $\mathbf{X_j}$ as follows:

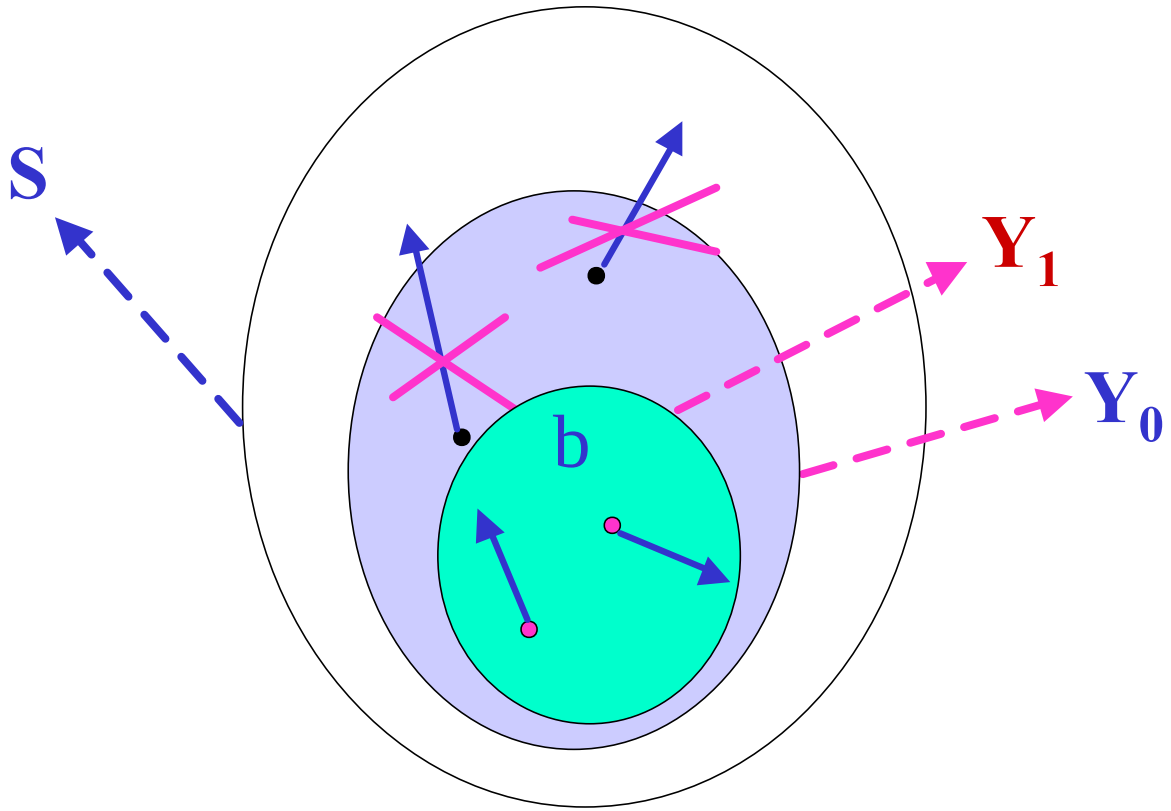
  - $\mathbf{X_1} = |[b]|$.
  - $\mathbf{X_{j+1}} = \mathbf{X_j} - \{ s \mid s \in \mathbf{X_j}$ and
    there does not exist $t \in \mathbf{X_j}$
    such that $\mathbf{R(s, t)}\}$

$|[\mathbf{EGb}]|$ is then the set $\mathbf{X}$ such that $\mathbf{X} = \mathbf{X_n}$ for $\mathbf{m}$ such that $\mathbf{X_{n+1}} = \mathbf{X_n}$.

- Notice that $\mathbf{m}$ ***must exists*** by ***Tarski's Theorem*** since $\emptyset \subseteq \mathbf{X_{j+1}} \subseteq \mathbf{X_j}$

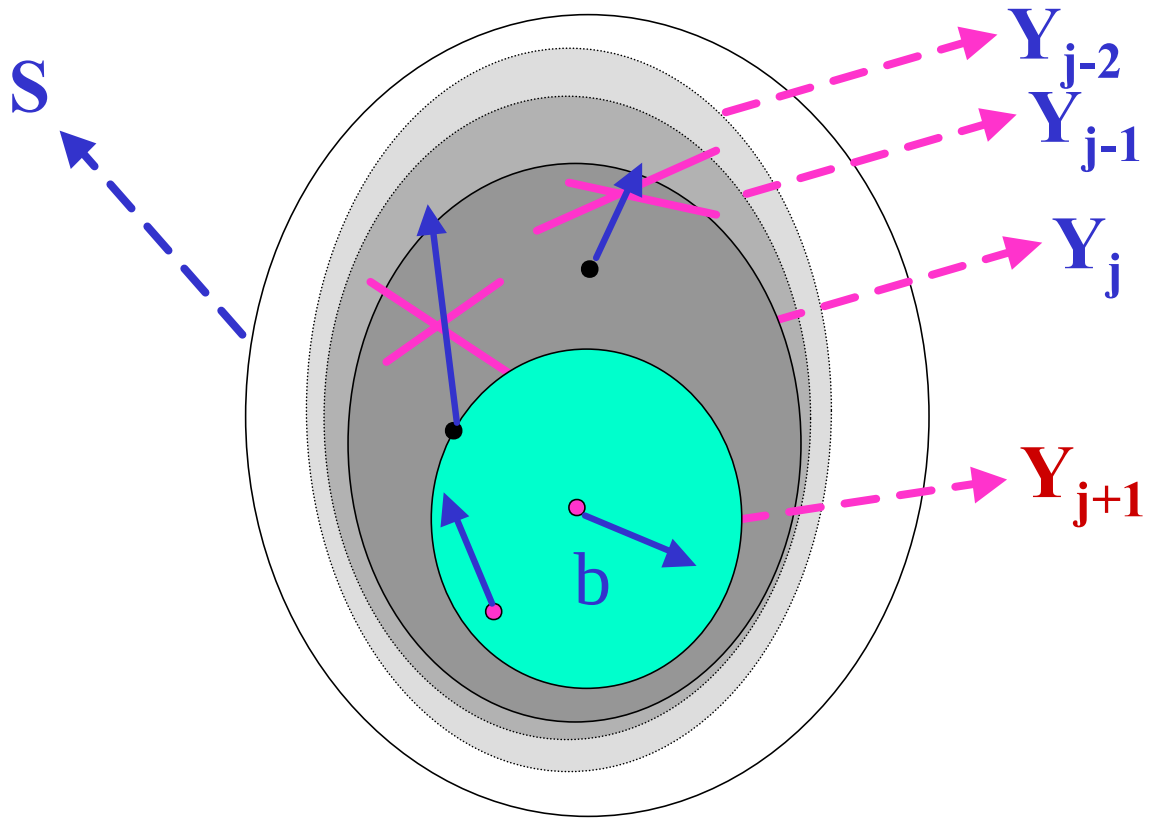
From Y_0 to Y_1

EGb



From Y_j to Y_{j+1}

EGb



Computing the fixed point for EG(β)

Algorithm Compute_EG(β)

X_1 := |[β]|;

j=1;

repeat

j := j+1; T := X_j := X_{j-1};

while T ¹ Æ do

chose s Î T;

T := T \{s};

if for no t Î R(s), t Î X_{j-1} then

X_j := X_j - {s};

until X_j = X_{j-1};