Session 9
Algorithms for Model Checking CTL

Contents

1 Algorithm for Model Checking CTL

ACM Turing Award 2007

Edmund Clarke

E. Allen Emerson

Joseph Sifakis

> For their role in developing Model-Checking into a highly effective verification technology, widely adopted in the hardware and software industries (1980-1983)

- Use of Logic CTL (from Ben-Ari, Pnueli and Manna)
- Linear-time labelling algorithm for model checking CTL
- EMC model checker implemented in Franz-Lisp
- The goal was to extend Hoare logic to concurrent systems...
- but it is easier to check if a formula satisfies a model than to ensure the validity of a formula (post-condition)

Model checking for CTL

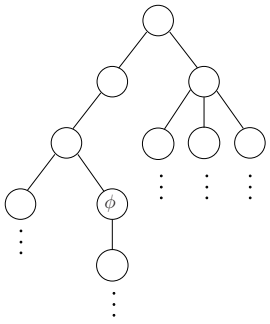
The semantics of CTL is state-based

$$s_0 \models \varphi$$

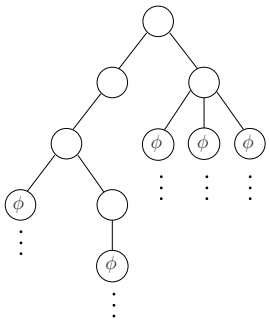
so it is not needed to reasoning over paths as in the case of LTL.

- Given $\mathcal{M}$ and $\varphi \in CTL$ the algorithm computes *the set of states* that satisfy φ, $Sat(\varphi)$.
- this is accomplished by computing $Sat(\psi)$ for all sub-formulae of φ
- then, one only needs to check that $s_0 \in Sat(\varphi)$

Semantics of CTL

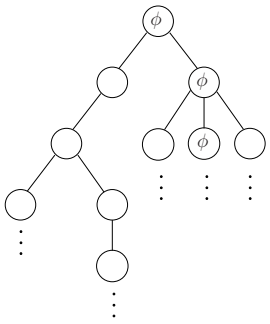


EFφ

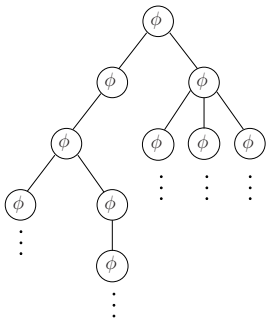


AFφ

Semantics of CTL



EGφ



AGφ

Model checking for CTL

We only consider a complete set of connectives

$$\{\mathsf{false}, \neg, \wedge, \mathrm{AF}, \mathrm{EU}, \mathrm{EX}\}$$

$$\begin{aligned}
\mathrm{EG}\varphi &\equiv \neg\mathrm{AF}\neg\varphi \\
\mathrm{EF}\varphi &\equiv \mathrm{E}[\neg\mathsf{false}\mathrm{U}\varphi] \\
\mathrm{AG}\varphi &\equiv \neg\mathrm{EF}\neg\varphi \equiv \neg\mathrm{E}[\neg\mathsf{false}\mathrm{U}\neg\varphi] \\
\mathrm{AX}\varphi &\equiv \neg\mathrm{EX}\neg\varphi \\
\mathrm{A}[\varphi\mathrm{U}\psi] &\equiv \neg(\mathrm{E}[\neg\psi\mathrm{U}(\neg\varphi \wedge \neg\psi)]) \wedge \mathrm{AF}\psi
\end{aligned}$$

because it LTL we have

$$\varphi\mathrm{U}\psi \equiv \neg(\neg\psi\mathrm{U}(\neg\varphi \wedge \neg\psi)) \wedge \mathrm{F}\psi$$

$\mathbf{A}[\varphi\mathbf{U}\psi] \equiv \neg\mathbf{E}[\neg\psi\mathbf{U}(\neg\varphi \wedge \neg\psi)] \wedge \neg\mathbf{EG}\neg\psi$

Using CTL^*,

$$\begin{aligned}
\mathrm{A}[\varphi\mathrm{U}\psi] &\equiv \mathrm{A}[\neg(\neg\psi\mathrm{U}(\neg\varphi \wedge \neg\psi)) \wedge \mathrm{F}\psi] \\
&\equiv \neg\mathrm{E}\neg[\neg(\neg\psi\mathrm{U}(\neg\varphi \wedge \neg\psi)) \wedge \mathrm{F}\psi] \\
&\equiv \neg\mathrm{E}[\neg\psi\mathrm{U}(\neg\varphi \wedge \neg\psi) \vee \mathrm{G}\neg\psi] \\
&\equiv \neg(\mathrm{E}[\neg\psi\mathrm{U}(\neg\varphi \wedge \neg\psi)] \vee \mathrm{EG}\neg\psi) \\
&\equiv \neg\mathrm{E}[(\neg\psi\mathrm{U}(\neg\varphi \wedge \neg\psi)]) \wedge \neg\mathrm{EG}\neg\psi)
\end{aligned}$$

1.1 Labelling Algorithm

Model checking for CTL – Labelling Algorithm

Input: a transition system $T = (S, Act, \longrightarrow, AP, I, L)$ and a CTL formula φ

Output: $Sat(\varphi)$, i.e., the set of states of T that satisfy φ,

- $T \models \varphi$ iff $I \subseteq Sat(\varphi)$.
- The algorithm labels each state of T with the sub-formulae of φ that are satisfied in that state,
- starting from the smaller ones: first, atomic propositions, boolean formulae and temporal formulae until φ.
- thus the algorithm proceeds by induction in the structure of φ
- If ψ is a sub-formula of φ which immediate sub-formulae already label the states were they are true, one can determine the states labelled by ψ.

Labelling

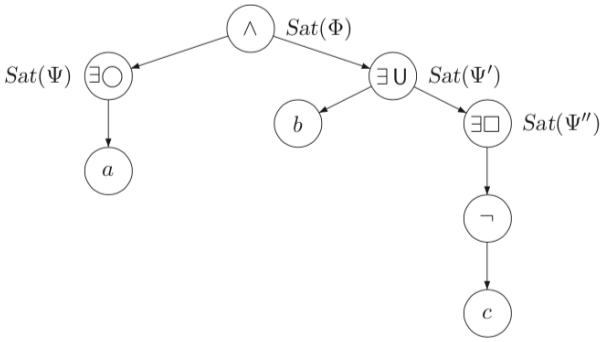
for $i \leq |\varphi|$ **do**
  for $\psi \in Sub(\varphi)$ with $i = |\psi|$ **do**
    Compute $Sat(\psi)$
if $I \subseteq Sat(\varphi)$ **then return** true
return false

where $Sub(\varphi)$ is the set of all sub-formulae of φ

- The recursive computation of $Sat(\varphi)$ consists in a bottom-up transversal of the parse tree of φ
- The nodes of the parse tree are subformulae of φ
- The leaves are the atomic propositions $a \in AP$ or a constant
- The inner nodes are labelled with an operator

Example

$$\Phi = \underbrace{\exists \bigcirc a}_{\Psi} \wedge \exists (b \,\mathsf{U}\, \underbrace{\underbrace{\exists \Box \neg c}_{\Psi''}}_{\Psi'}) \quad .$$



Model checking for CTL – Labelling Algorithm

If ψ is

false label no state

p label the states s such that $p \in L(s)$

$\psi_1 \wedge \psi_2$ label the states s that are already labelled by ψ_1 and ψ_2

$\neg\psi_1$ label the states s that are not labelled with ψ_1

General idea for temporal connectives

The labelling for the temporal connectives is based on the following equivalences

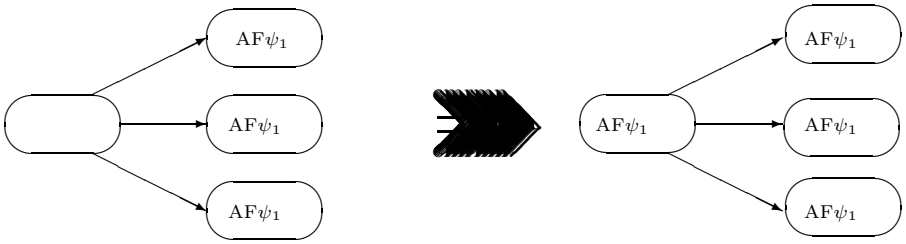
$$\begin{aligned}
\mathrm{AG}\varphi &\equiv \varphi \wedge \mathrm{AXAG}\varphi \\
\mathrm{EG}\varphi &\equiv \varphi \wedge \mathrm{EXEG}\varphi \\
\mathrm{AF}\varphi &\equiv \varphi \vee \mathrm{AXAF}\varphi \\
\mathrm{EF}\varphi &\equiv \varphi \vee \mathrm{EXEF}\varphi \\
\mathrm{A}[\varphi \mathrm{U} \psi] &\equiv \psi \vee (\varphi \wedge \mathrm{AXA}[\varphi \mathrm{U} \psi]) \\
\mathrm{E}[\varphi \mathrm{U} \psi] &\equiv \psi \vee (\varphi \wedge \mathrm{EXE}[\varphi \mathrm{U} \psi])
\end{aligned}$$

but only AF, EU and EX are needed.

Labelling Algorithm – AFψ_1

AFψ_1

- If a state s is labelled with ψ_1 then label it with AFψ_1.
- Repeat: If all successors of a state s are labelled with AFψ_1, label that state with AFψ_1. Until there is no change.

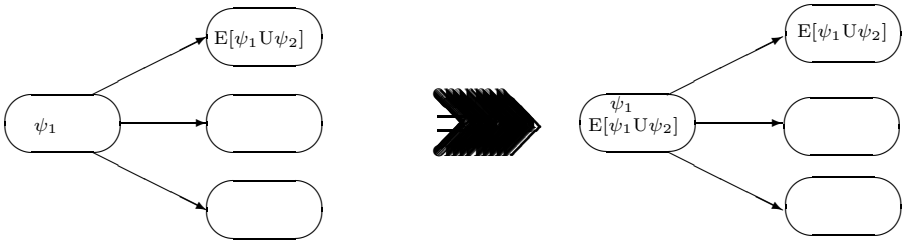


$$\mathrm{AF}\psi_1 \equiv \psi_1 \ \vee \ \mathrm{AXAF}\psi_1$$

Labelling Algorithm – $\mathbf{E}(\psi_1 \mathbf{U} \psi_2)$

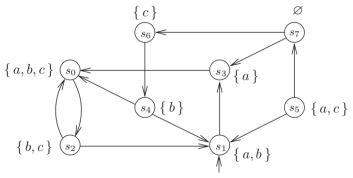
$\mathbf{E}(\psi_1 \mathbf{U} \psi_2)$

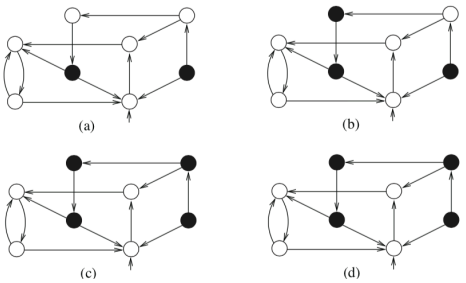
- If a state s is labelled with ψ_2 then label s with $\mathrm{E}(\psi_1 \mathrm{U} \psi_2)$.
- Repeat: Label a state s with $\mathrm{E}(\psi_1 \mathrm{U} \psi_2)$ if it is labelled with ψ_1 and if at least one of its successors is labelled with $\mathrm{E}(\psi_1 \mathrm{U} \psi_2)$. Until there is no change.



$$\mathrm{E}[\psi_1 \mathrm{U} \psi_2] \equiv \psi_2 \ \vee \ (\psi_1 \ \wedge \ \mathrm{EXE}[\psi_1 \mathrm{U} \psi_2])$$

$$\mathrm{EF}((a \leftrightarrow c) \wedge \neg(a \leftrightarrow b)) \equiv \mathrm{E}(\mathsf{true}\mathrm{U}(a \leftrightarrow c) \wedge \neg(a \leftrightarrow b))$$

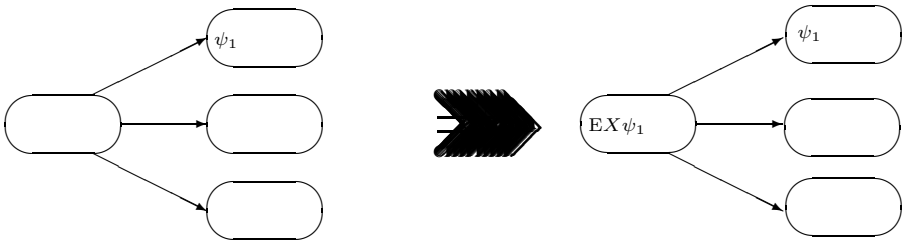




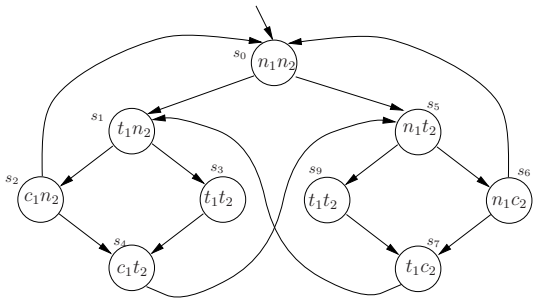
a) $Sat((a \leftrightarrow c) \wedge \neg(a \leftrightarrow b))$

Labelling Algorithm – EXψ_1

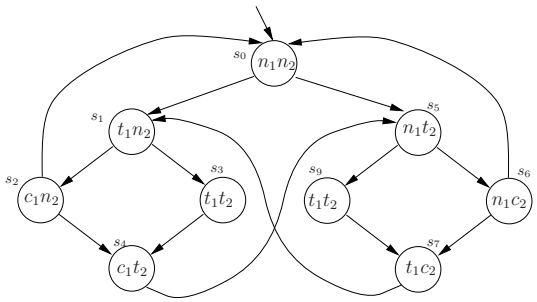
EXψ_1 label with EXψ_1 a state if at least one of its successors is labelled with ψ_1.



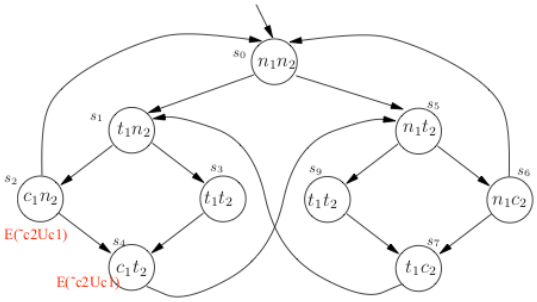
Example: Peterson algorithm $s_0 \models \mathbf{E}(\neg c_2 \mathbf{U} c_1)$?



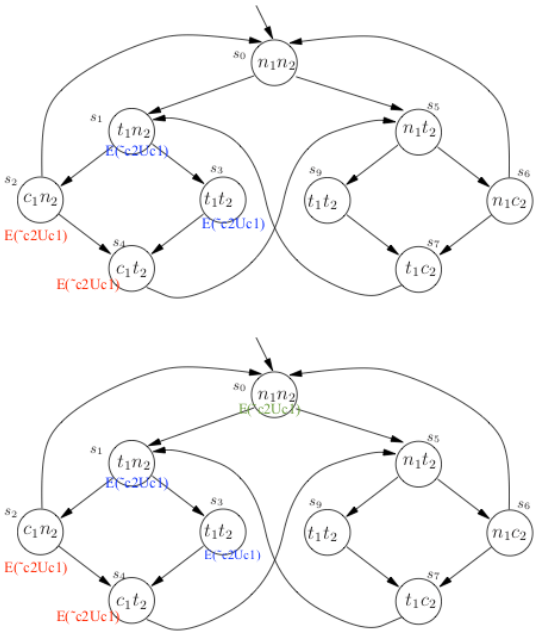
The subformulae are: c_1, c_2, $\neg c_2$ and E($\neg c_2$Uc_1). The states labeled by $\neg c_2$ are the ones not labeled by c_2.



Check the states labelled with c_1 and labelled them with $\mathrm{E}(\neg c_2 \mathrm{U} c_1)$.



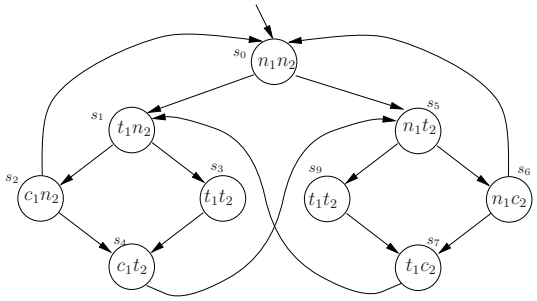
If a state is labelled with $\neg c_2$ and has a successor labelled with $\mathrm{E}(\neg c_2 \mathrm{U} c_1)$ label that state with $\mathrm{E}(\neg c_2 \mathrm{U} c_1)$.



Yes, $s_0 \models \mathrm{E}(\neg c_2 \mathrm{U} c_1)$

Example: Mutual Exclusion $s_0 \models \mathbf{AG}(\neg(c_2 \wedge c_1))$?

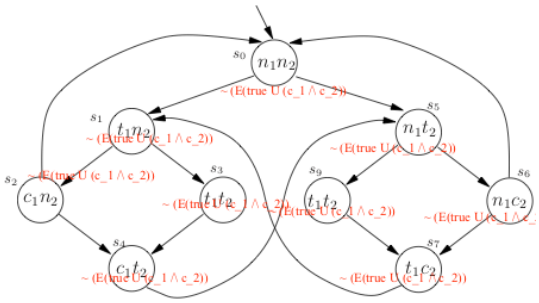
$$\mathrm{AG}(\neg(c_2 \wedge c_1)) \equiv \neg\mathrm{EF}(c_2 \wedge c_1)) \equiv \neg\mathrm{E}(\mathsf{true}\mathrm{U}(c_2 \wedge c_1))$$



It is no possible to label any state with $c_2 \wedge c_1$, so no state can be labeled $\mathrm{E}(\mathsf{true}\mathrm{U}(c_2 \wedge c_1))$

Example: Mutual Exclusion $s_0 \models \mathbf{AG}(\neg(c_2 \wedge c_1))$?

$$\mathrm{AG}(\neg(c_2 \wedge c_1)) \equiv \neg\mathrm{EF}(c_2 \wedge c_1)) \equiv \neg\mathrm{E}(\mathsf{true}\mathrm{U}(c_2 \wedge c_1))$$



Thus, all states are labeled with $\neg\mathrm{E}(\mathsf{true}\mathrm{U}(c_2 \wedge c_1))$ and thus in particular s_0.

Pseudo-code for the labelling algorithm given φ e T

function SAT (φ)
begin
  case
    φ is true : **return** S
    φ is false : **return** $\emptyset$

φ is atomic: **return** $\{s \in S \mid \varphi \in L(s)\}$
φ is $\neg\varphi_1$: **return** $S - \mathtt{SAT}(\varphi_1)$
φ is $\varphi_1 \wedge \varphi_2$: **return** $\mathtt{SAT}(\varphi_1) \cap \mathtt{SAT}(\varphi_2)$
φ is $\varphi_1 \vee \varphi_2$: **return** $\mathtt{SAT}(\varphi_1) \cup \mathtt{SAT}(\varphi_2)$
φ is $\varphi_1 \rightarrow \varphi_2$: **return** $\mathtt{SAT}(\neg\varphi \vee \varphi_2)$
φ is $\mathrm{AX}\varphi_1$: **return** $\mathtt{SAT}(\neg\mathrm{EX}\neg\varphi_1)$
φ is $\mathrm{EX}\varphi_1$: **return** $\mathtt{SAT}_{\mathtt{EX}}(\varphi_1)$
φ is $\mathrm{A}[\varphi_1 \mathrm{U}\, \varphi_2]$: **return** $\mathtt{SAT}(\neg(\mathrm{E}[\neg\varphi_2 \mathrm{U}(\neg\varphi_1 \wedge \neg\varphi_2)] \vee \mathrm{EG}\neg\varphi_2))$
φ is $\mathrm{E}[\varphi_1 \mathrm{U} \varphi_2]$: **return** $\mathtt{SAT}_{\mathtt{EU}}(\varphi_1, \varphi_2)$
φ is $\mathrm{EF}\varphi_1$: **return** $\mathtt{SAT}(\mathrm{E}(\mathsf{true}\mathrm{U}\varphi_1))$
φ is $\mathrm{EG}\varphi_1$: **return** $\mathtt{SAT}(\neg\mathrm{AF}\neg\varphi_1)$
φ is $\mathrm{AF}\varphi_1$: **return** $\mathtt{SAT}_{\mathtt{AF}}(\varphi_1)$
φ is $\mathrm{AG}\varphi_1$: **return** $\mathtt{SAT}(\neg\mathrm{EF}\neg\varphi_1)$
end case
end function

Pseudo-code for the labelling algorithm

Given a set of states Y, the function $\mathtt{pre}_\exists(Y)$ ($\mathtt{pre}_\forall(Y)$) determines the set of states from which it is possible (only it is possible) to make a transition for states in Y:

$$\begin{aligned}
\mathtt{pre}_\exists(Y) &= \{s \in S \mid \exists s',\ s \longrightarrow s' \wedge s' \in Y\} \\
&= \{s \in S \mid Post(s) \cap Y \neq \emptyset\} \\
\mathtt{pre}_\forall(Y) &= \{s \in S \mid \forall s'(s \longrightarrow s') \Rightarrow s' \in Y)\} \\
&= \{s \in S \mid Post(s) \subseteq Y\}
\end{aligned}$$

Then

function $\mathtt{SAT}_{\mathtt{EX}}(\varphi)$
local var X, Y
begin
$X := \mathtt{SAT}(\varphi)$;
$Y := \mathtt{pre}_\exists(X)$;
return Y
end

function $\mathtt{SAT}_{\mathtt{AF}}(\varphi)$
/* determines the set of states satisfying $\mathrm{AF}\varphi$ */
local var X, Y
begin
$X := S$;
$Y := \mathtt{SAT}(\varphi)$;
/* least fixed point */
repeat until $X = Y$

```
    begin
        X := Y;
        Y := Y ∪ pre_∀(Y)
    end
    return Y
end
```

```
function SAT_EU (φ, ψ)
/* determines the set of states satisfying E[φU ψ] */
local var W, X, Y
begin
    W := SAT (φ);
    X := S;
    Y := SAT (ψ);
/* least fixed point */
    repeat until X = Y
    begin
        X := Y;
        Y := Y ∪ (W ∩ pre_∃(Y))
    end
    return Y
end
```

Labelling Algorithm for CTL

Consider $AP = \{p, q, t, r\}$ and the model $T = (S = \{q_0, q_1, q_2, q_3\}, \{q_0 \to q_1, q_0 \to q_3, q_1 \to q_1, q_1 \to q_2, q_2 \to q_0, q_2 \to q_3, q_3 \to q_0\}, L(q_0) = \{p, q\}, L(q_1) = \{r\}, L(q_2) = \{p, t\}, L(q_3) = \{q, r\})$.

Determine $Sat(\varphi)$ where

a) $\varphi = AFq$,

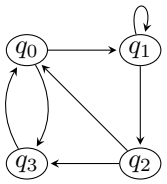
b) $\varphi = \text{EXEX}r$

c) $\varphi = \text{AG}(\text{EF}(p \vee r))$.

Labelling Algorithm

Consider $AP = \{p, q, t, r\}$ and the model $T = (\{q_0, q_1, q_2, q_3\}, \{q_0 \to q_1, q_0 \to q_3, q_1 \to q_1, q_1 \to q_2, q_2 \to q_0, q_2 \to q_3, q_3 \to q_0\}, L(q_0) = \{p, q\}, L(q_1) =$

$\{r\}, L(q_2) = \{p, t\}, L(q_3) = \{q, r\})$.



$$\begin{aligned} Post(q_0) &= \{q_1, q_3\} \\ Post(q_1) &= \{q_1, q_2\} \\ Post(q_2) &= \{q_0, q_3\} \\ Post(q_3) &= \{q_0\} \end{aligned}$$

For $Sat(\mathrm{AF}q) = Y$:

- $Sat(q) = \{q_0, q_3\}$
- Let $Y = Sat(q) = \{q_0, q_3\}$
- As $pre_\forall(Y) = \{s \mid Post(s) \subseteq \{q_0, q_3\}\} = \{q_2, q_3\}$
- we have $Y = Y \cup \{q_2, q_3\} = \{q_0, q_2, q_3\}$
- Repeating we have again $pre_\forall(Y) = \{q_2, q_3\}$, thus $Sat(\mathrm{AF}q) = Y = \{q_0, q_2, q_3\}$

For $Sat(\mathrm{EXEX}r)$:

- $Sat(r) = \{q_1, q_3\}$
- Let $X = Sat(r) = \{q_1, q_3\}$
- $X = pre_\exists(X) = \{s \mid Post(s) \cap \{q_1, q_3\} \neq \emptyset\} = \{q_0, q_1, q_2\}$
- Repeating, $pre_\exists(X) = pre_\exists(\{q_0, q_1, q_2\}) = \{q_0, q_1, q_2, q_3\}$
- Then $Sat(\mathrm{EXEX}r) = \{q_0, q_1, q_2, q_3\} = S$

$\mathrm{AG}(\mathrm{EF}(p \vee r)) = \neg\mathrm{E}[\mathsf{true}\mathrm{U}\neg(\mathrm{E}[\mathsf{true}\mathrm{U}(p \vee r)])]$
For $Sat(\mathrm{E}[\mathsf{true}\mathrm{U}(p \vee r)])$

- $Y = Sat(p \vee r) = \{q_0, q_1, q_2, q_3\} = S$ and $W = S$
- Then $Y = Y \cup (W \cap pre_\exists(Y)) = S$ and $Sat(\mathrm{E}[\mathsf{true}\mathrm{U}(p \vee r)]) = S$

Thus $Sat(\neg\mathrm{E}[\mathsf{true}\mathrm{U}(p \vee r)]) = \emptyset$

- $Y = \emptyset$ and $W = S$
- $pre_\exists(Y) = \emptyset$
- $Y = Y \cup (W \cap pre_\exists(Y)) = \emptyset = Sat(\mathrm{E}[\mathsf{true}\mathrm{U}\neg(\mathrm{E}[\mathsf{true}\mathrm{U}(p \vee r)])])$

Thus $Sat(\mathrm{AG}(\mathrm{EF}(p \vee r))) = S$

Complexity

Given a model $T = (S, Act, \longrightarrow, AP, I, L)$ and a CTL formula φ, the labelling algorithm has time complexity

$$O(f \cdot V \cdot (V + E))$$

where

- f is the number of connectives in φ
- $V = |S|$ is the number of states in T
- $E = |\ \longrightarrow\ |$ is the number of transitions in T

It can be more efficient if one consider explicitly the case for EG. Then the complexity can be $O(f.(V + E))$, thus linear both in the size of the model and in the size of the formula.

LTL versus CTL

Aspect	*Linear time*	*Branching time*
"behavior" in a state s	path-based: $trace(s)$	state-based: computation tree of s
temporal logic	LTL: path formulae φ $s \models \varphi$ iff $\forall \pi \in Paths(s).\, \pi \models \varphi$	CTL: state formulae existential path quantification $\exists\varphi$ universal path quantification: $\forall\varphi$
complexity of the model checking problems	PSPACE–complete $\mathcal{O}\left(\lvert TS\rvert \cdot \exp(\lvert\varphi\rvert)\right)$	*PTIME* $\mathcal{O}(\lvert TS\rvert \cdot \lvert\Phi\rvert)$
implementation-relation	trace inclusion and the like (proof is PSPACE-complete)	simulation and bisimulation (proof in polynomial time)
fairness	no special techniques needed	special techniques needed