

### Algoritmos de *model checking*

A semântica dada para o CTL e o LTL, permite decidir se

$$s_0 \models \varphi$$

Mas é conveniente ter algoritmos de decisão.

Normalmente, dado  $\mathcal{M}$  e  $\varphi$  os algoritmos determinam o conjunto de estados que satisfazem  $\varphi$

Depois basta ver se  $s_0$  está nesse conjunto.

### Model checking para o CTL

Vamos apenas considerar um conjunto completo de conectivas,

$$\{\text{false}, \neg, \wedge, \text{AF}, \text{EU}, \text{EX}\}$$

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

sendo que no LTL

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

### Model checking para o CTL – algoritmo de etiquetagem

[Clarke and Emerson 1981, 2007 Turing Award]

**Dados:** um sistema de transições  $T = (S, Act, \longrightarrow, AP, I, L)$  e uma fórmula CTL  $\varphi$

**Saída:**  $Sat(\varphi)$ , i.e. conjunto de estados de  $T$  que satisfazem  $\varphi$ ,

Temos que  $T \models \varphi$  sse  $I \subseteq Sat(\varphi)$ .

O algoritmo etiqueta cada estado de  $T$  com as subfórmulas de  $\varphi$  que são satisfeitas nesse estado, começando das mais pequenas até  $\varphi$ . O algoritmo procede por indução na estrutura de  $\varphi$ . Supondo que  $\psi$  é uma subfórmula de  $\varphi$  cujas subfórmulas imediatas já etiquetam todos os estados em que são satisfeitas, determina-se por análise de casos os estados que têm  $\psi$  na etiqueta.

### Etiquetagem

```
for  $i \leq |\varphi|$  do
  for  $\psi \in Sub(\varphi)$  com  $i = |\psi|$  do
```

Calcular  $Sat(\psi)$

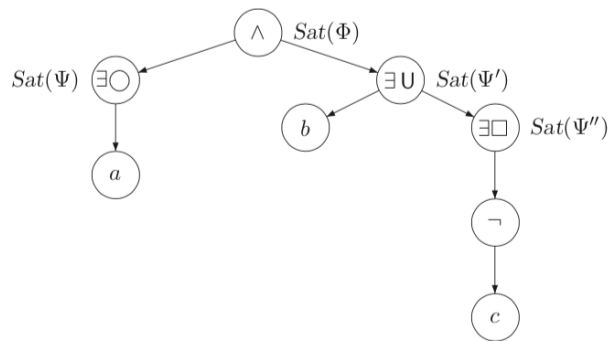
**if**  $I \subseteq Sat(\varphi)$  **then return true**  
**return false**

onde  $Sub(\varphi)$  é o conjunto de todas as subfórmulas de  $\varphi$ .

Bastará percorrer a árvore sintática de  $\varphi$  começando com as folhas. Para cada  $\psi$  podemos associar uma nova variável proposicional  $a_\psi$  e para cada estado  $s$  tal que  $s \in Sat(\psi)$  podemos supor que acrescentamos  $a_\psi$  a  $L(s)$ .

**Exemplo**

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



Por exemplo  $\Psi''$  pode ser substituído por  $a_1$ , tal que  $a_1 \in L(s)$  se só se  $s \in Sat(\Psi'')$  e depois seria necessário calcular  $Sat(E[bUa_1])$ .

**Model checking para o CTL – algoritmo de etiquetagem**

Se  $\psi$  é

false Não etiqueta nenhum estado

$p$  etiquetar os estados  $s$  tal que  $p \in L(s)$

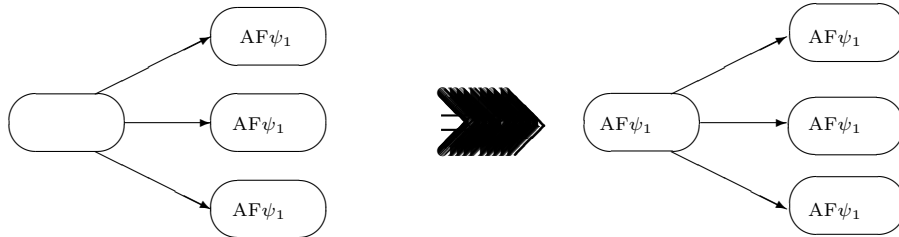
$\psi_1 \wedge \psi_2$  etiquetar os estados  $s$  que estejam etiquetados com  $\psi_1$  e  $\psi_2$

$\neg\psi_1$  etiquetar os estados  $s$  que não estejam etiquetados com  $\psi_1$

**Model checking para o CTL – algoritmo de etiquetagem**

**AF** $\psi_1$  • Se existe  $s$  que está etiquetado com  $\psi_1$  então etiquetá-lo com **AF** $\psi_1$ .

- Repetir: Se todos os sucessores de um estado estiverem etiquetados com  $AF\psi_1$ , etiquetar esse estado. Até não haver alteração.

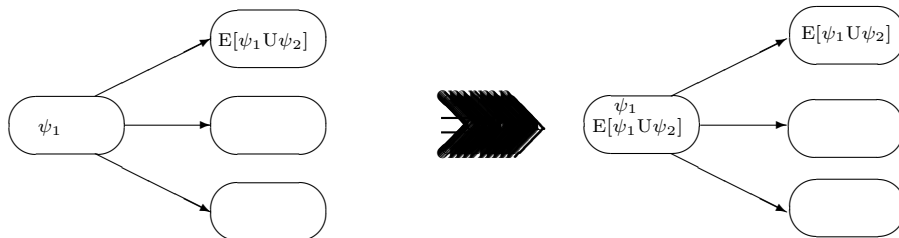


$$AF\varphi \equiv \varphi \vee AXAF\varphi$$

### Model checking para o CTL – algoritmo de etiquetagem

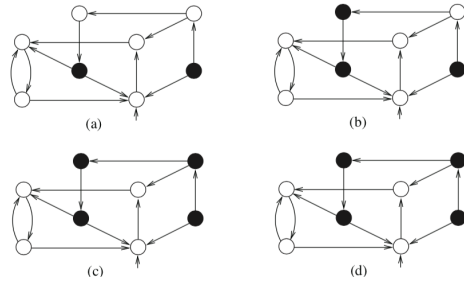
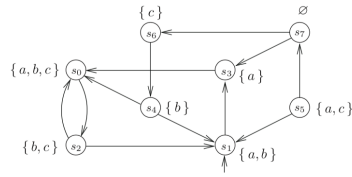
$E(\psi_1 U \psi_2)$  • Se existe  $s$  que está etiquetado com  $\psi_2$  então etiquetá-lo com  $E(\psi_1 U \psi_2)$ .

- Repetir: Etiquetar um estado com  $E(\psi_1 U \psi_2)$  se está etiquetado com  $\psi_1$  e se pelo menos um dos seus sucessores está etiquetado com  $E(\psi_1 U \psi_2)$ . Até não haver alteração.



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

$$E(\text{true}U(a \leftrightarrow c) \wedge \neg(a \leftrightarrow b))$$



**Model checking para o CTL – algoritmo de etiquetagem**

$EX\psi_1$  etiquetar com  $EX\psi_1$  um estado se pelo menos um dos seus sucessores estiver etiquetado com  $\psi_1$ .

**Model checking para o CTL – algoritmo de etiquetagem**

**Exercício 13.1.** Considere o modelo  $T = (\{q_0, q_1, q_2, q_3\}, \{q_0 \rightarrow q_1, q_0 \rightarrow q_3, q_1 \rightarrow q_1, q_1 \rightarrow q_2, q_2 \rightarrow q_0, q_2 \rightarrow q_3, q_3 \rightarrow q_0\}, L(q_0) = \{p, q\}, L(q_1) = \{r\}, L(q_2) = \{p, t\}, L(q_3) = \{q, r\})$ .

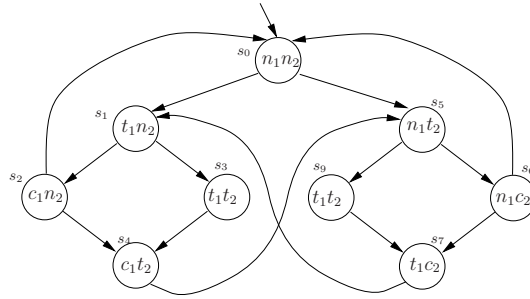
Determine os estados  $s$  tal que  $q_0 \models \varphi$  para

- a)  $\varphi = AFq$ ,
- b)  $\varphi = EXEXr$  e
- c)  $\varphi = AG(EF(p \vee r))$ .

◇

**Model checking para o CTL – algoritmo de etiquetagem**

**Exemplo2:**  $s_0 \models E(\neg c_2 U c_1)$ ?



### Model checking para o CTL – algoritmo de etiquetagem

O algoritmo tem complexidade  $O(f \cdot V \cdot (V + E))$ , onde  $f$  é o número de conectivas da fórmula,  $V$  o número de estados e  $E$  ou número de transições.

Pode ser mais eficiente se se considerarem mais conectivas, p.e. EG.

- EG $\psi$**
- Etiquetar todos os estados com EG $\psi$ .
  - Se um estado  $s$  não tiver a etiqueta  $\psi$ , apagar a etiqueta EG $\psi$ .
  - Repetir: apagar a etiqueta EG $\psi$  se nenhum dos sucessores do estado tiver a etiqueta EG $\psi$ . Até não haver mudança.
  - EG $\psi \equiv \psi \wedge \text{EXEG}\psi$

Equivale ter  $\neg \text{AF}\neg\psi$

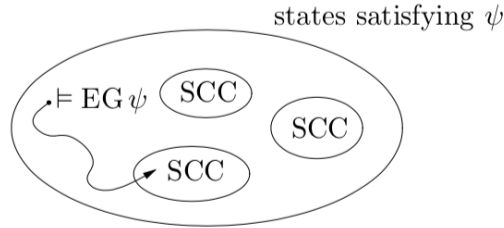
### Uma variante mais eficiente

Se considerarmos EG em vez de AF no conjunto completo de conectivas podemos obter um algoritmo linear,  $O(f \cdot (V + E))$ .

Para EX e EU o algoritmo é o mesmo, mas considerando uma pesquisa em largura.

- EG $\psi$**
- Restringir o digrafo a estados que satisfazem  $\psi$
  - Determinar as componentes fortemente conexas (maximais) (SCC). Nos estados dessas componentes verifica-se EG $\psi$
  - Utilizar pesquisa em largura (para trás) para encontrar um estado que possa estar ligado a uma SCC. Esse estado também verifica EG $\psi$  e pode-se continuar a pesquisa.

## Uma variante mais eficiente



## Pseudo-código para o algoritmo dado $\varphi$ e $T$

```

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

```

## Pseudo-código para o algoritmo

Dado um conjunto de estados  $Y$ , a função  $\text{pre}_{\exists}(Y)$  ( $\text{pre}_{\forall}(Y)$ ) determina os estados de que se pode (ou só se pode) chegar a  $Y$ :

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

```

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

```

Pseudo-código para o algoritmo

```

function SATAF ( $\varphi$ )
local var  $X, Y$ 
begin
   $X := S$ ;
   $Y := \text{SAT}(\varphi)$ ;
  repeat until  $X = Y$ 
  begin
     $X := Y$ ;
     $Y := Y \cup \text{pre}_{\forall}(Y)$ 
  end
  return  $Y$ 
end

```

Pseudo-código para o algoritmo

```

function SATEU ( $\varphi, \psi$ )
/* determina os estados que satisfazem  $E[\varphi U \psi]$  */
local var  $W, X, Y$ 
begin
   $W := \text{SAT}(\varphi)$ ;
   $X := S$ ;
   $Y := \text{SAT}(\psi)$ ;
  repeat until  $X = Y$ 
  begin
     $X := Y$ ;
     $Y := Y \cup (W \cap \text{pre}_{\exists}(Y))$ 
  end
  return  $Y$ 
end

```

Pseudo-código para o algoritmo

```

function SATEG ( $\varphi$ )
/* determines the set of states satisfying EG  $\varphi$  */
local var X, Y
begin
  Y := SAT( $\varphi$ );
  X :=  $\emptyset$ ;
  repeat until X = Y
  begin
    X := Y;
    Y := Y  $\cap$  pre $_{\exists}$ (Y)
  end
  return Y
end

```

## Funções monótonas

### Pontos Fixos

Seja  $S$  um conjunto de estados e  $F : \mathcal{P}(S) \rightarrow \mathcal{P}(S)$

1.  $F$  é monótona sse  $X \subseteq Y$  então  $F(X) \subseteq F(Y)$
2.  $X \subseteq S$  é ponto fixo sse  $F(X) = X$

**Exercício 13.2.** Sendo  $S = \{s_0, s_1\}$  e  $F(Y) = Y \cup \{s_0\}$  mostrar que  $F$  é monótona e determinar os pontos fixos.

### Correção e Terminação do Algoritmo

Para provar que o algoritmo de etiquetagem é correcto e termina:

- Provar que cada função recursiva é monótona em  $(\mathcal{P}(S), \subseteq)$ .
- O ponto fixo máximo (ou mínimo) é atingido e é a semântica pretendida.

**Teorema 13.1** (Knaster-Tarski). *Se  $S$  é um conjunto com  $n + 1$  elementos e  $F : \mathcal{P}(S) \rightarrow \mathcal{P}(S)$  é monótona, então  $F^{n+1}(\emptyset)$  é o ponto fixo mínimo de  $F$  e  $F^{n+1}(S)$  é o ponto fixo máximo de  $F$ .*

### Teorema de Knaster-Tarskiq

beginproof Como  $\emptyset \subseteq F(\emptyset)$  então também  $F(\emptyset) \subseteq F(F(\emptyset))$  e por indução podemos mostrar que

$$\emptyset \subseteq F(\emptyset) \subseteq F(F(\emptyset)) \subseteq \dots \subseteq F^i(\emptyset)$$

para qualquer  $i \geq 1$ . Para  $i = n + 1$  vamos mostrar que existe  $0 \leq k \leq n + 1$  tal que  $F^k(\emptyset)$  é um ponto fixo de  $F$ . Por contradição suponhamos que não. Então,



$F(\emptyset)$  tem pelo menos 1 elemento (senão seria igual a  $\emptyset$ ),  $F^2(\emptyset)$  2 elementos, etc, e  $F^{n+2}(\emptyset)$  teria  $n + 2$  elementos o que é absurdo. Logo concluímos que em particular  $F^{n+1}(\emptyset)$  é um ponto fixo. Agora suponhamos que  $F(X) = X$  temos de mostrar que  $F^{n+1}(\emptyset) \subseteq X$ . Como  $\emptyset \subseteq X$  temos  $F(\emptyset) \subseteq F(X) = X$ . Mas repetindo temos  $F^2(\emptyset) \subseteq F(X) = X, \dots, F^{n+1}(\emptyset) \subseteq X$ .

### Correção e Terminação do Algoritmo

Para demonstrar a correção de SAT temos:

1. A semântica de EG, AF e EU pode ser expressa em termos de pontos fixos máximos ou mínimos de funções em  $P(S)$ ,
2. Estes pontos fixos são calculáveis
3.  $SAT_{EG}$ ,  $SAT_{AF}$  e  $SAT_{EU}$  codificam o cálculo destes pontos fixos (o primeiro máximo, e os outros mínimos).

### Correção e Terminação do Algoritmo

Seja  $Sat(\varphi) \subseteq S$  o conjunto de estados que satisfaz  $\varphi$ .

Os caso base ( $false, true, p$ ) são calculados directamente.

Para a negação e disjunção é calculado o conjunto usando o complemento e a união respectivamente:

$$Sat(\neg\varphi) = S \setminus Sat(\varphi)$$

$$Sat(\varphi_1 \vee \varphi_2) = Sat(\varphi_1) \cup Sat(\varphi_2).$$

E para EX vem:

$$Sat(EX\varphi) = pre_{\exists}(Sat(\varphi))$$

### Correção e Terminação do Algoritmo

Para as restantes conectivas temporais temos:

- $Sat(EG\varphi) = Sat(\varphi) \cap pre_{\exists}(Sat(EG\varphi))$
- $Sat(E[\varphi U \psi]) = Sat(\psi) \cup (Sat(\varphi) \cap pre_{\exists}Sat(E[\varphi U \psi]))$
- $Sat(AF\varphi) = Sat(\varphi) \cup pre_{\forall}(Sat(AF\varphi))$

Para  $SAT_{EG}$ ,  $SAT_{EU}$ , e  $SAT_{AF}$  basta associar a respectiva função, provar a sua monotonia e mostrar que o ponto fixo máximo (mínimos) é a semântica pretendida.

### Correção de $EG\varphi$

**Teorema 13.2.** *Seja  $F(X) = Sat(\varphi) \cap pre_{\exists}(X)$  e  $S$  com  $n + 1$  elementos. Então  $F^{n+1}(S) = Sat(EG\varphi)$ .*

1.  $F$  é monótona: basta ver que se  $X \subseteq Y$  então  $pre_{\exists}(X) \subseteq pre_{\exists}(Y)$
2.  $Sat(EG\varphi)$  é um ponto fixo de  $F$
3.  $Sat(EG\varphi)$  ponto fixo máximo de  $F$
4. Se  $F(X) = X$  então  $X \subseteq Sat(EG\varphi)$
5.  $F^{n+1}(S) = Sat(EG\varphi)$  pelo teorema de Knaster-Tarski.

### Correção de $EU\varphi$

$$Sat(E[\varphi U\psi]) = Sat(\psi) \cup (Sat(\varphi) \cap pre_{\exists}Sat(E[\varphi U\psi]))$$

**Teorema 13.3.** *Seja  $G(X) = Sat(\psi) \cup (Sat(\varphi) \cup pre_{\exists}(X))$  e  $S$  com  $n + 1$  elementos. Então  $G^{n+1}(\emptyset) = Sat(EG\varphi)$ .*

1.  $G$  é monótona: basta ver que se  $X \subseteq Y$  então  $pre_{\exists}(X) \subseteq pre_{\exists}(Y)$
2.  $Sat(E[\varphi U\psi])$  ponto fixo mínimo de  $G$
3. Se  $G(X) = X$  então  $X \supseteq Sat(E[\varphi U\psi])$

### Model checking with Fairness

- Verificar uma propriedade  $\varphi$  somente ao longo de caminhos onde se verifica um número infinito de vezes uma condição  $\psi$ :
  - em LTL?  $GF\psi \rightarrow \varphi$
  - e em CTL? Isso não é possível porque os quantificadores não podem ser aplicados em implicações, p.e.  $A(GF\psi \rightarrow \varphi)$  ou  $E(GF\psi \rightarrow \varphi)$  não são fórmulas válidas.

Seja  $C = \{\psi_1, \dots, \psi_n\}$  um conjunto de condições. Um caminho  $s_0 \rightarrow s_1 \rightarrow \dots$  diz-se **fair** w.r.t.  $C$  se e só se para cada  $i \geq 0$  existe um número infinito de  $j$ 's tal que  $s_j \models \psi_i$ .

Sejam  $A_C$  e  $E_C$  os operadores restritos a caminhos **fair**, i.e. temos  $s \models A_C G\varphi$  sse  $\varphi$  for verdadeiro em todos os estados ao longo de todos os caminhos **fair**, etc.

Como é que podemos adaptar os algoritmos para verificar  $E_C U$ ,  $E_C X$  e  $E_C G$ ?

### Exemplo

Considera a fórmula  $\varphi = \neg a \wedge AG(a \rightarrow b) \wedge AF(a \wedge EX \neg b) \wedge EG b$ .

- a) Indica um sistema de transições  $T$  e um estado  $s$  tal que  $s \models \varphi$ .
- b) Aplica o algoritmo de etiquetagem a  $T$  e  $\varphi$ , indicando todos os passos intermédios.

Usando a função  $F(X) = Sat(\varphi) \cup pre_{\forall}(X)$  demonstra que  $Sat(AF\varphi)$  é o ponto fixo mínimo de  $F$ .

### Model checking with Fairness

1.  $E_C[\varphi U \psi] \equiv E[\varphi U(\psi \wedge E_C G \text{true})]$
2.  $E_C X \varphi \equiv EX(\varphi \wedge E_C G \text{true})$
3. para  $E_C G \varphi$ :
  - Restringir o digrafo a estados onde se verifica  $\varphi$ .
  - Determinar as componentes fortemente conexas (maximais) (SCC).
  - Remover uma SSC se, para algum  $\psi_i$ , não tiver nenhum estado onde se verifica  $\psi_i$ . As restantes SCC's são **fair**.
  - Utilizar pesquisa em largura (para trás) para encontrar um estado que possa estar ligado a uma SCC que seja **fair**.

### Model checking with Fairness

