Pseudo-código para o algoritmo dado ϕ e $\mathcal{M} = (S, \rightarrow, L)$

function $\texttt{SAT}(\phi)$
begin
  case
    ϕ is $\top$: **return** S
    ϕ is $\bot$: **return** $\emptyset$
    ϕ is atomic: **return** $\{s \in S \mid \phi \in L(s)\}$
    ϕ is $\neg\phi_1$: **return** $S - \texttt{SAT}(\phi_1)$
    ϕ is $\phi_1 \wedge \phi_2$: **return** $\texttt{SAT}(\phi_1) \cap \texttt{SAT}(\phi_2)$
    ϕ is $\phi_1 \vee \phi_2$: **return** $\texttt{SAT}(\phi_1) \cup \texttt{SAT}(\phi_2)$
    ϕ is $\phi_1 \rightarrow \phi_2$: **return** $\texttt{SAT}(\neg\phi \vee \phi_2)$
    ϕ is $\mathrm{AX}\phi_1$: **return** $\texttt{SAT}(\neg\mathrm{EX}\neg\phi_1)$
    ϕ is $\mathrm{EX}\phi_1$: **return** $\texttt{SAT}_{\texttt{EX}}(\phi_1)$
    ϕ is $\mathrm{A}[\phi_1 \mathrm{U}\, \phi_2]$: **return** $\texttt{SAT}(\neg(\mathrm{E}[\neg\phi_2 \mathrm{U}(\neg\phi_1 \wedge \neg\phi_2)] \vee \mathrm{EG}\neg\phi_2))$
    ϕ is $\mathrm{E}[\phi_1 \mathrm{U} \phi_2]$: **return** $\texttt{SAT}_{\texttt{EU}}(\phi_1, \phi_2)$
    ϕ is $\mathrm{EF}\phi_1$: **return** $\texttt{SAT}(\mathrm{E}(\top \mathrm{U} \phi_1))$
    ϕ is $\mathrm{EG}\phi_1$: **return** $\texttt{SAT}(\neg\mathrm{AF}\neg\phi_1)$
    ϕ is $\mathrm{AF}\phi_1$: **return** $\texttt{SAT}_{\texttt{AF}}(\phi_1)$
    ϕ is $\mathrm{AG}\phi_1$: **return** $\texttt{SAT}(\neg\mathrm{EF}\neg\phi_1)$
  end case
end function

Pseudo-código para o algoritmo

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

$$\begin{aligned} \texttt{pre}_\exists(Y) &= \{s \in S \mid \text{ existe } s', s \rightarrow s', \text{ e } s' \in Y\} \\ \texttt{pre}_\forall(Y) &= \{s \in S \mid \text{para todo } s', (s \rightarrow s', \text{ implica } s' \in Y)\} \end{aligned}$$

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

Pseudo-código para o algoritmo

function $\texttt{SAT}_{\texttt{AF}}(\phi)$
local var X, Y
begin

```
    X := S;
    Y := SAT(φ);
    repeat until X = Y
    begin
        X := Y;
        Y := Y ∪ pre∀(Y)
    end
    return Y
end
```

Pseudo-código para o algoritmo

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

Pseudo-código para o algoritmo

```
function SAT_EG(φ)
/* determines the set of states satisfying EG φ */
local var X, Y
begin
    Y := SAT(φ);
    X := ∅;
    repeat until X = Y
    begin
        X := Y;
        Y := Y ∩ pre∃(Y)
    end
    return Y
end
```

Funções monótonas

Pontos Fixos

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

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 9.1. *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 é atingido e é a semântica pretendida.

Teorema 9.1 (Knaster-Tarski)**.** *Se S é um conjunto com $n + 1$ elementos e $F : \mathcal{P}(S) \to \mathcal{P}(S)$ 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.*

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. $\texttt{SAT}_{\texttt{EG}}$, $\texttt{SAT}_{\texttt{AF}}$ e $\texttt{SAT}_{\texttt{EU}}$ codificam o cálculo destes pontos fixos (o primeiro máximo, e os outros mínimos).

Correção e Terminação do Algoritmo

Seja $[\![\phi]\!] \subseteq S$ o conjunto de estados que satisfaz ϕ.

Os caso base $(\bot, \top, p)$ são calculados directamente.

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

$[\![\neg\phi]\!] = S \setminus [\![\phi]\!]$

$[\![\phi_1 \vee \phi_2]\!] = [\![\phi_1]\!] \cup [\![\phi_2]\!]$.

E para EX vem:

$[\![\mathrm{EX}\phi]\!] = pre_{\exists}([\![\phi]\!])$

Correção e Terminação do Algoritmo

Para as restantes conectivas temporais temos:

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

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

Correção de EGϕ

Teorema 9.2. *Seja* $F(X) = [\![\phi]\!] \cap pre_{\exists}(X)$ *e* S *com* $n+1$ *elementos. Então* $F^{n+1}(S) = [\![EG\phi]\!]$.

1. F é monótona: basta ver que se $X \subseteq Y$ então $pre_{\exists}(X) \subseteq pre_{\exists}(Y)$
2. $[\![\text{EG}\phi]\!]$ ponto fixo máximo de F
3. Se $F(X) = X$ então $X \subseteq [\![\text{EG}\phi]\!]$

Correção de EUϕ

$[\![\text{E}[\phi \text{U} \psi]]\!] = [\![\psi]\!] \cup ([\![\phi]\!] \cap pre_{\exists}[\![\text{E}[\phi \text{U} \psi]]\!])$

Teorema 9.3. *Seja* $G(X) = [\![\psi]\!] \cup ([\![\phi]\!] \cup pre_{\exists}(X))$ *e* S *com* $n+1$ *elementos. Então* $G^{n+1}(\emptyset) = [\![EG\phi]\!]$.

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

Model checking with Fairness

```
MODULE prc(other-st, turn, myturn)
   VAR
      st: {n, t, c};
   ASSIGN
      init(st) := n;
      next(st) :=
         case
            (st = n)                                    : {t,n};
```

```
         (st = t) & (other-st = n)                    : c;
         (st = t) & (other-st = t) & (turn = myturn): c;
         (st = c)                                     : {c,n};
         1                                            : st;
      esac;
   next(turn) :=
      case
         turn = myturn & st = c : !turn;
         1                      : turn;
      esac;
FAIRNESS running
FAIRNESS  !(st = c)
```

Model checking with Fairness

- Verificar uma propriedade ϕ somente ao longo de caminhos onde se verifica um número infinito de vezes uma condição ψ:
  - em LTL? $\mathrm{GF}\psi \rightarrow \phi$
  - e em CTL?

Seja $C = \{\psi_1, \ldots, \psi_n\}$ um conjunto de condições. Um caminho $s_0 \rightarrow s_1 \rightarrow \ldots$ 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 $\mathcal{M}, s \models \mathrm{A}_C\mathrm{G}\phi$ sse ϕ for verdadeiro em todos os estados ao longo de todos os caminhos `fair`, etc.

Como é que podemos adaptar os algoritmos para verificar $\mathrm{E}_C\mathrm{U}$, $\mathrm{E}_C\mathrm{X}$ e $\mathrm{E}_C\mathrm{G}$?

Model checking with Fairness

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