Problèmes de vérification pour LTL
NuSMV

- $\models \phi$?

$\phi = \mathbf{G}\,\mathbf{F}\,a \wedge \mathbf{G}\,\mathbf{F}\,b$

```
MODULE main
VAR
    a : boolean;
    b : boolean;
LTLSPEC NAME
prop1 :=  !(G F a & G F b);
```

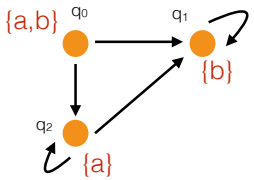

PRISM

Problèmes de vérification pour LTL
Prism

Model-checking

- $\mathbf{S} \models \phi$?

S



$\phi = \mathbf{G}\,(a \Rightarrow \mathbf{F}\,b)$

```
mdp

label "a" = (q=0 | q=2);
label "b" = (q=0) | (q=1);

module K
        q : [0..2] init 0;
    [] q=0 -> (q'=1);
    [] q=0 -> (q'=2);
    [] q=1 -> true;
    [] q=2 -> true;
    [] q=2 -> (q'=1);
endmodule
```

A [G (« a » => (F « b »))]

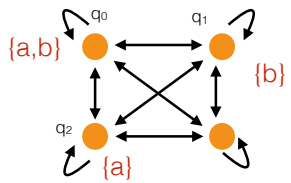
Problèmes de vérification pour LTL
Prism

Satisfaisabilité

- $\models \phi$?

$\phi = \mathbf{G}\,\mathbf{F}\,a \wedge \mathbf{G}\,\mathbf{F}\,b$

S



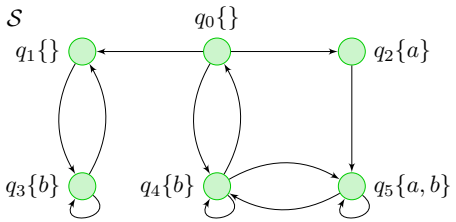

```
mdp

label "a" = (va=true);
label "b" = (vb=true);

module General
        va : bool init false;
        vb : bool init false;
    [] true -> (va'=true) & (vb'=true);
    [] true -> (va'=false) & (vb'=true);
    [] true -> (va'=true) & (vb'=false);
    [] true -> (va'=false) & (vb'=false);
endmodule
```

E [X ((G F « a ») & (G F « b »))]

Problèmes de vérification pour LTL

Prism

- $\mathbf{S} \models \phi$?




```
mdp

label "a" = (q=2 | q=5);
label "b" = (q=3 | q=4 | q=5);

module General
        q : [0..5];
    [] (q=0) -> (q'=1);
    [] (q=0) -> (q'=2);
    [] (q=0) -> (q'=4);
    [] (q=1) -> (q'=3);
    [] (q=3) -> (q'=3);
    [] (q=3) -> (q'=1);
    [] (q=2) -> (q'=5);
    [] (q=4) -> (q'=0);
    [] (q=4) -> (q'=4);
    [] (q=4) -> (q'=5);
    [] (q=5) -> (q'=4);
    [] (q=5) -> (q'=5);
endmodule
```


Algorithme de Peterson

1981

[pb de section critique]

```
int turn := 1      // var. partagées
bool D1:=False
bool D2:=False
```

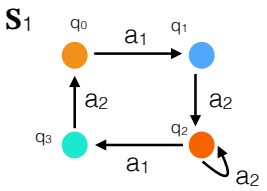
Processus P_1:
loop forever:
p1: section NC
p2: D1:= True
p3: turn:=2
p4: await (turn==1 || D2==False)
p5: section critique
p6: D1 := False

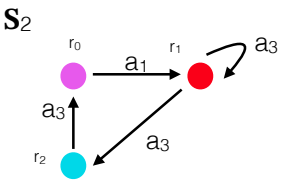
Processus P_2:
loop forever:
q1: section NC
q2: D2:= True
q3: turn:=1
q4: await (turn==2 || D1==False)
q5: section critique
q6: D2 := False

Algorithme de Peterson en Prism

```
global turn : [1 .. 2];

module P1
        D1 : bool init false;
        p : [0..3] init 0;
    [] (p=0) -> true;
    [] (p=0) -> (p'=1) & (D1'=true);
    [] (p=1) -> (p'=2) & (turn'=2);
    [] (p=2) & ((turn=1) | (D2=false)) -> (p'=3);
    [] (p=2) & ((turn=2) & (D2=true)) -> (p'=2);
    [] (p=3) -> (p'=0) & (D1'=false);
endmodule

module P2
        D2 : bool init false;
        q : [0..3] init 0;
    [] (q=0) -> true;
    [] (q=0) -> (q'=1) & (D2'=true);
    [] (q=1) -> (q'=2) & (turn'=1);
    [] (q=2) & ((turn=2) | (D1=false)) -> (q'=3);
    [] (q=2) & ((turn=1) & (D1=true)) -> (q'=2);
    [] (q=3) -> (q'=0) & (D2'=false);
endmodule
```

Composition parallèle & synchronisation

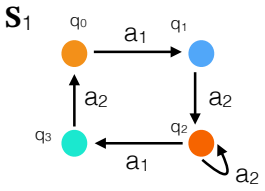




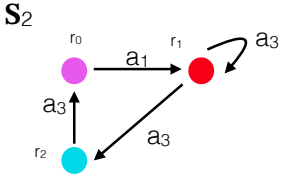
S_1	S_2
a_1	a_1
a_2	-
-	a_3

Composition parallèle & synchronisation

```
module S1
        q : [0..3] init 0;
    [a1] (q=0) -> (q'=1);
    [a2] (q=1) -> (q'=2);
    [a2] (q=2) -> true;
    [a1] (q=2) -> (q'=3);
    [a2] (q=3) -> (q'=0);
endmodule
```




```
module S2
        r : [0..2] init 0;
    [a1] (r=0) -> (r'=1);
    [a3] (r=1) -> true;
    [a3] (r=1) -> (r'=2);
    [a3] (r=2) -> (r'=0);
    endmodule
```



Algorithme de Peterson avec scheduler

```
global turn : [1 .. 2];
```

```
module P1
        D1 : bool init false;
        p : [0..3] init 0;
    [a1] (p=0) -> true;
    [a1] (p=0) -> (p'=1) & (D1'=true);
    [a1] (p=1) -> (p'=2) & (turn'=2);
    [a1] (p=2) & ((turn=1) | (D2=false)) -> (p'=3);
    [a1] (p=2) & ((turn=2) & (D2=true)) -> (p'=2);
    [a1] (p=3) -> (p'=0) & (D1'=false);
endmodule
```

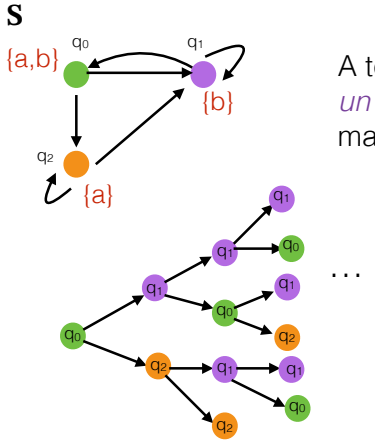
```
module P2
        D2 : bool init false;
        q : [0..3] init 0;
    [a2] (q=0) -> true;
    [a2] (q=0) -> (q'=1) & (D2'=true);
    [a2] (q=1) -> (q'=2) & (turn'=1);
    [a2] (q=2) & ((turn=2) | (D1=false)) -> (q'=3);
    [a2] (q=2) & ((turn=1) & (D1=true)) -> (q'=2);
    [a2] (q=3) -> (q'=0) & (D2'=false);
endmodule
```

```
module Scheduler
        r : [0..1] init 0;
    [a1] (r=0) -> (r'=1);
    [a2] (r=1) -> (r'=0);
endmodule
```

-> force l'alternance entre les deux processus.

CTL
Computation Tree Logic

idée: ne plus voir le système comme un ensemble d'exécutions linéaires… être plus proche du STE.



A tout moment le système est dans *un* état et il peut évoluer de plusieurs manières.

…

—> un arbre d'exécution.

CTL

Formules de CTL

$\varphi,\psi ::= P \mid \neg\varphi \mid \varphi \vee \psi \mid \mathbf{EX}\varphi \mid \mathbf{AX}\varphi \mid \mathbf{E}\varphi\mathbf{U}\psi \mid \mathbf{A}\varphi\mathbf{U}\psi$

avec $P \in AP$

+ Abbréviations :

$\top$, $\bot$, $\wedge$, $\Rightarrow$

$\mathbf{F}\varphi = \top\ \mathbf{U}\ \varphi$: "eventually",

$\mathbf{G}\ \varphi = \neg\ \mathbf{F}\ \neg\varphi$: "always"

$\varphi\mathbf{W}\ \psi = \varphi\mathbf{U}\ \psi\ \vee \mathbf{G}\ \varphi$: "weak until"

$\mathbf{EF}\varphi$ $\mathbf{AF}\varphi$ $\mathbf{EG}\varphi$ $\mathbf{AG}\varphi$ $\mathbf{E}\varphi\mathbf{W}\psi$ $\mathbf{A}\varphi\mathbf{W}\psi$

CTL - sémantique

$\mathbf{S} = (Q, Act, \rightarrow, q_{init}, AP, L)$

Exec(q) = ens. des exécutions infinies partant de q.

$\pi \in Exec(q)$: $\pi = q_0 q_1 q_2 q_3 q_4 \ldots$ avec $q_0 = q$ et $q_i \rightarrow q_{i+1}$

Notation: $\pi(i) = q_i \quad \forall\ i \geq 0$

On interprète les formules de CTL sur des états de $\mathbf{S}$.

$q \models P$ iff $P \in L(q)$

$q \models \mathbf{EX}\varphi$ iff $\exists\ q \rightarrow q'$ t.q. $q' \models \varphi$

$q \models \mathbf{AX}\ \varphi$ iff $\forall\ q \rightarrow q'$, on a: $q' \models \varphi$

$q \models \mathbf{E}\varphi\mathbf{U}\psi$ iff $\exists \pi \in Exec(q)$ t.q. $\exists\ i \geq 0$ t.q. ($\pi(i) \models \psi$ et $(\forall\ 0 \leq j < i: \pi(j) \models \varphi)$)

$q \models \mathbf{A}\varphi\mathbf{U}\psi$ iff $\forall\ \pi \in Exec(q), \exists\ i \geq 0$ t.q. ($\pi(i) \models \psi$ et $(\forall\ 0 \leq j < i: \pi(j) \models \varphi)$)

CTL

Définition alternative (équivalente !!):

Formules d'état:

$\varphi,\psi ::= P \mid \neg\varphi \mid \varphi \vee \psi \mid \mathbf{E}\ \varphi_p \mid \mathbf{A}\ \varphi_p$

$P \in AP$

Formules de chemin:

$\varphi_p,\psi_p ::= \mathbf{X}\ \varphi \mid \varphi\ \mathbf{U}\ \psi$

$\mathbf{E}\ \varphi_p$ = « il existe un chemin vérifiant φ_p »

$\mathbf{A}\ \varphi_p$ = « tous les chemins vérifient φ_p »

CTL - sémantique

Définition alternative (équivalente !!):

$q \models P$ iff $P \in L(q)$

$q \models \mathbf{E}\ \varphi_p$ iff $\exists\ \pi \in Exec(q)$ t.q. $\pi \models \varphi_p$

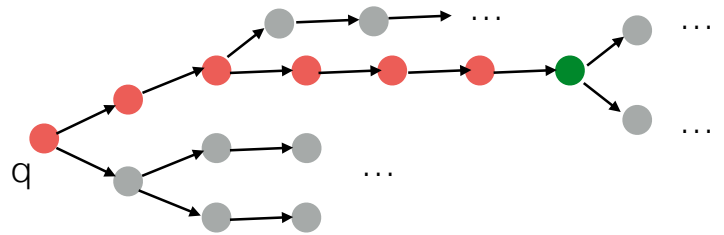
$q \models \mathbf{A}\ \varphi_p$ iff $\forall\ \pi \in Exec(q)$, $\pi \models \varphi_p$

$\pi \models \mathbf{X}\ \varphi$ iff $\pi(1) \models \varphi$

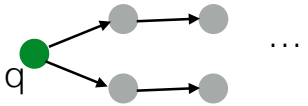
$\pi \models \varphi\ \mathbf{U}\ \psi$ iff $\exists\ i \geq 0$ ($\pi(i) \models \psi$ et $(\forall\ 0 \leq j < i: \pi(j) \models \varphi)$)

CTL - sémantique

$q \models$ **E rouge U vert**

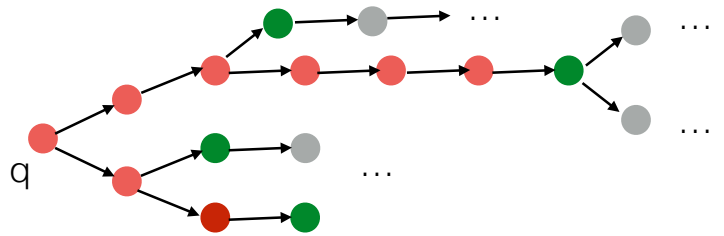


ou:

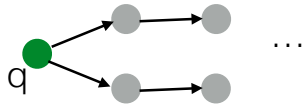


CTL - sémantique

$q \models$ **A rouge U vert**

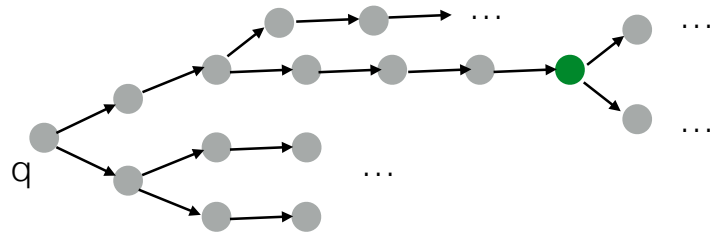


ou:

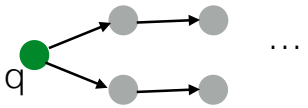


CTL - sémantique

$q \models$ **EF vert**

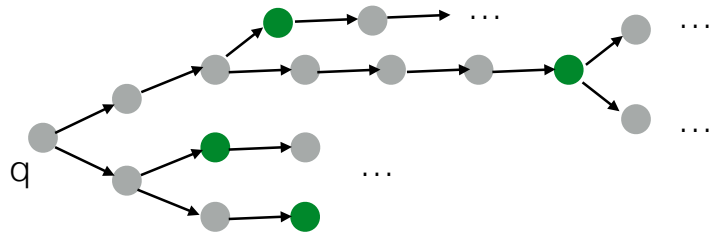


ou:

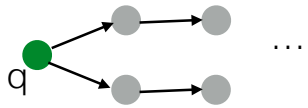


CTL - sémantique

$q \models$ **AF vert**

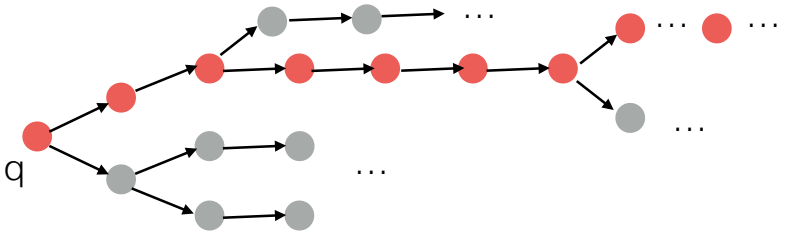


ou:



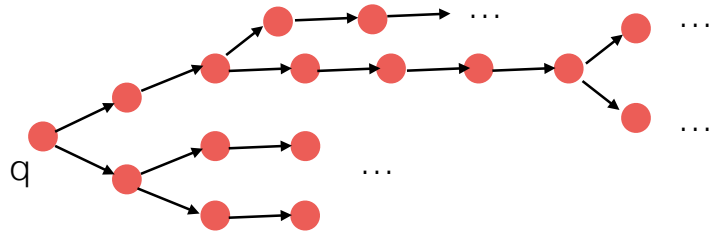
CTL - sémantique

$q \models$ **EG rouge**



CTL - sémantique

$q \models$ **AG rouge**



Tout ce qui est accessible depuis q est rouge.

Exemples

AG (problème ⇒ **AF** alarme)

« tout état accessible qui vérifie problème est suivi inévitablement, un jour, par un état vérifiant alarme »

AG (**EX** a)

« tout état accessible a un successeur immédiat vérifiant a »

E (**EX** a) **U** b

« il est possible d'atteindre un état vérifiant b le long d'un chemin où tout état a un successeur immédiat vérifiant a »

AG (**EF** a)

« Depuis tout état accessible, il est possible d'atteindre un état vérifiant a »

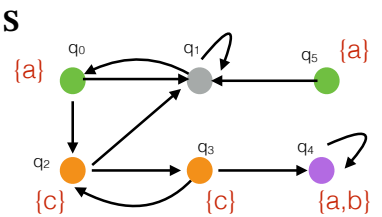
Comment évaluer une formule de CTL sur un état d'un STE?

φ = **AG** (a ⇒ **E** (**EX** c) **U** b)

$q_0 \models \varphi$?

Les sous-formules de φ :

φ, a , a ⇒ **E** (**EX** c) **U** b ,
E (**EX** c) **U** b , **EX** c , b , c



	a	b	c	**EX** c	**E** (**EX** c) **U** b	a ⇒…	φ
q_0	⊤	⊥	⊥	⊤	⊤	⊤	⊤
q_1	⊥	⊥	⊥	⊥	⊥	⊤	⊤
q_2	⊥	⊥	⊤	⊤	⊤	⊤	⊤
q_3	⊥	⊥	⊤	⊤	⊤	⊤	⊤
q_4	⊤	⊤	⊥	⊥	⊤	⊤	⊤
q_5	⊤	⊥	⊥	⊥	⊥	⊥	⊥

Algorithme de Model-Checking pour CTL

Model-checking de CTL

procédure Marquage (φ) :

cas 1 : $\varphi = p$
- Pour tout $q \in Q$:
  - Si $p \in \ell(q)$ Alors $q.\varphi := \top$
  - Sinon $q.\varphi := \bot$

cas 2 : $\varphi = \neg\psi$
- Marquage (ψ)
- Pour tout $q \in Q$:
  - $q.\varphi := \neg\ q.\psi$

cas 3 : $\varphi = \psi_1 \wedge \psi_2$
- Marquage (ψ_1) , Marquage (ψ_2)
- Pour tout $q \in Q$:
  - $q.\varphi := q.\psi_1 \wedge q.\psi_2$

cas 3' : $\varphi = \psi_1 \vee \psi_2$
- Marquage (ψ_1) , Marquage (ψ_2)
- Pour tout $q \in Q$:
  - $q.\varphi := q.\psi_1 \vee q.\psi_2$

Model-checking de CTL

procédure Marquage (φ) (suite)

cas 4 : $\varphi = EX\psi$
- Marquage (ψ)
- Pour tout $q \in Q$: $q.\varphi := \bot$
- Pour toute $(q \to q')$:
  - Si $q'.\psi$ Alors $q.\varphi := \top$

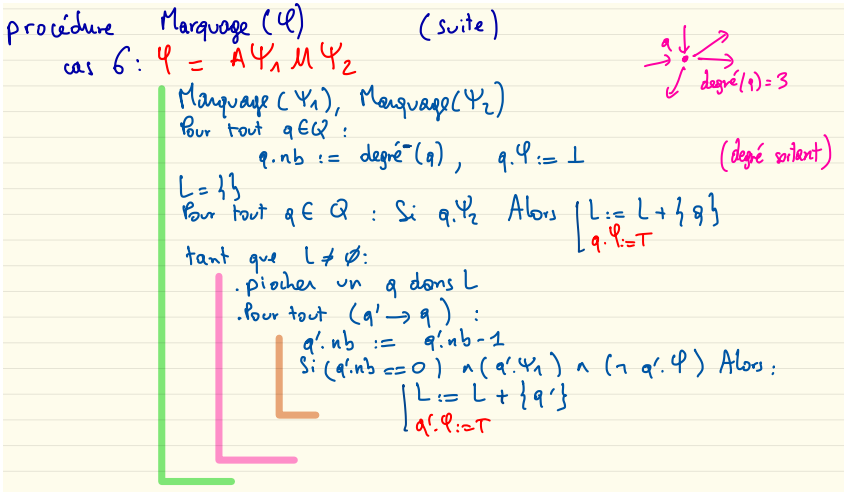
$AX\varphi = \neg EX\neg\varphi$

Model-checking de CTL

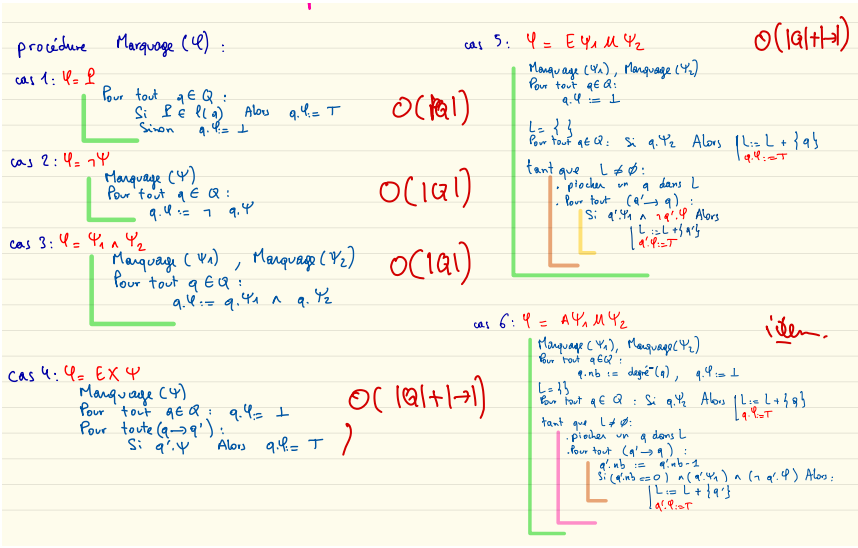
procédure Marquage (φ) (suite)

cas 5 : $\varphi = E\psi_1 \mathcal{U} \psi_2$
- Marquage (ψ_1) , Marquage (ψ_2)
- Pour tout $q \in Q$:
  - $q.\varphi := \bot$
- $L = \{\}$
- Pour tout $q \in Q$: Si $q.\psi_2$ Alors
  - $L := L + \{q\}$
  - $q.\varphi := \top$
- tant que $L \neq \emptyset$:
  - . piocher un q dans L [..le retirer]
  - . Pour tout $(q' \to q)$:
    - Si $q'.\psi_1 \wedge \neg q'.\varphi$ Alors
      - $L := L + \{q'\}$
      - $q'.\varphi := \top$

Model-checking de CTL



Model-checking de CTL



Model-checking de CTL

$|\varphi|$ = nb d'opérateurs + prop.

$$\begin{cases} |\varphi_1 \wedge \varphi_2| = |\varphi_1 \vee \varphi_2| = |E\varphi_1 \,M\, \varphi_2| = |A\varphi_1 \,M\, \varphi_2| = 1 + |\varphi_1| + |\varphi_2| \\ |\neg \varphi_1| = |EX\,\varphi_1| = |AX\,\varphi_1| = 1 + |\varphi_1| \\ |p| = 1 \end{cases}$$

La complexité de l'algo. de model-checking est donc en $O(|\varphi| \cdot |S|)$

$|S| = |Q| + |\to|$

La complexité polynomiale !

Quelle logique choisir ?

Quelle est la « meilleure »? CTL* ? LTL ? CTL ? …

Plusieurs critères:

- L'expressivité
- La complexité des procédures de décision
- Les outils (model-checkers...)
- …

Comparer l'expressivité de LTL et CTL

Temps linéaire / Temps arborescent

- Temps linéaire:

Equivalences de formules:

$\varphi,\psi \in$ LTL :

$\varphi \equiv \psi$ ssi (pour toute ***exécution*** π d'un STE, on a $(\pi \models \varphi$ ssi $\pi \models \psi)$)

Déf.: $\mathbf{S} \models \psi$ signifie « pour toute exec. π de $\mathbf{S}$, on a $\pi \models \psi$».

(ie $\mathbf{S} \models \mathbf{A}\,\psi$)

- Temps arborescent:

Equivalences de formules:

$\varphi,\psi \in$ CTL :

$\varphi \equiv \psi$ ssi (pour tout ***état*** q d'un STE, on a $(q \models \varphi$ ssi $q \models \psi)$)

Déf.: $\mathbf{S} \models \psi$ signifie « $q_{init} \models \psi$ ».

Expressivité

3 notions différentes:

- Le pouvoir de distinction
  - La logique $\mathcal{L}$ distingue au moins autant que $\mathcal{L}'$ ($\mathcal{L} \geq \mathcal{L}'$) ssi
    pour tous $\mathbf{S}$ et $\mathbf{S}'$, $\quad \mathbf{S} \equiv_{\mathcal{L}} \mathbf{S}' \Rightarrow \mathbf{S} \equiv_{\mathcal{L}'} \mathbf{S}'$
  - Avec: $\mathbf{S} \equiv_{\mathcal{L}} \mathbf{S}'$ ssi $(\forall\, \varphi \in \mathcal{L}, \mathbf{S} \models \varphi \iff \mathbf{S}' \models \varphi)$
- Le pouvoir d'expression
  - $\mathcal{L}$ est au moins aussi expressive que $\mathcal{L}'$ ($\mathcal{L} \geqslant \mathcal{L}'$) ssi

  $\forall\ \varphi' \in \mathcal{L}',\ \exists\, \varphi \in \mathcal{L}$ s.t. $\varphi \equiv \varphi'$
- La concision:

  Quand $\mathcal{L}$ et $\mathcal{L}'$ sont aussi expressive, une des deux peut être plus concise (exprimer les mêmes propriétés mais avec des formules exponentiellement plus petites)...

Pouvoir de distinction

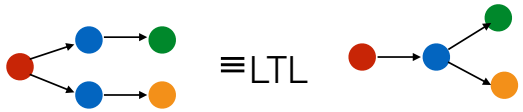
Pouvoir de distinction:
→ capacité à distinguer deux modèles **S** et **S'**.

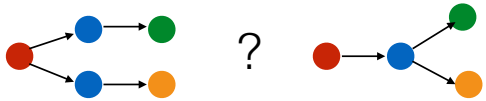
distinguer = trouver une formule vraie pour l'un et fausse pour l'autre.

Pouvoir de distinction de LTL

→ Si deux STE ont le même ensemble d'exécutions (*), alors ils vérifient les mêmes formules de LTL.

(*) on dit qu'ils sont « trace-equivalent »





A-t-on envie que ce soit équivalent ?

Une machine à café… deux implémentations.