Procédures de décision

LTL sans opérateurs du passé

Pour cette partie, on va considérer une variante de LTL où il n'y a ni Since, ni F^{-1}, ni X^{-1}.

Pourquoi ? pour simplifier un peu la suite…
sans rien perdre sur le fond:

- les mêmes techniques marchent pour LTL avec passé…
- les opérateurs du passé sont pratiques pour exprimer des propriétés mais pas indispensables: on peut toujours se *débrouiller* avec Until et X.

Débrouiller ? Toute formule de LTL avec passé est équivalente à une formule sans passé lorsqu'on les interprète au début d'une exécution.

Simplifier

LTL sans opérateurs du passé

Syntaxe:

$$\phi,\psi ::= P \mid \neg\phi \mid \phi\vee\psi \mid \phi\wedge\psi \mid \mathbf{X}\,\phi \mid \psi\mathbf{U}\phi$$

On peut interprète les formules de LTL sur une exécution ρ d'un STE ***sans position*** !

$\rho \models P$ ssi $P \in L(\rho(0))$

$\rho \models \mathbf{X}\,\phi$ ssi $\rho^1 \models \phi$

$\rho \models \psi\mathbf{U}\phi$ ssi ($\exists\, i\geq 0.$ ($\rho^i \models \phi$ et $\forall\, 0\leq j<i$ on a $\rho^j \models \psi$)

ρ^i est le i-ème suffixe: $\rho(i)\rho(i+1)\ldots$

Simplifier (suite)

A-t-on besoin du nom des états ? Non !

ρ + L = une séquence infinie de sous-ensembles de AP

Exemple:
Si ρ: $q_0 \rightarrow q_1 \rightarrow q_0 \rightarrow q_1 \rightarrow q_0 \rightarrow q_1 \rightarrow \ldots$ et $L(q_0)=\{a\}$, $L(q_1)=\{b,c\}$

ρ + L « = » {a}{b,c}{a}{b,c}{a}{b,c}{a}{b,c}{a}{b,c}{a}{b,c}…

=> On peut interprèter les formules de LTL sur une séquence infinie de sous-ensembles de AP !

Simplifier

ρ + L

$\rho \models P$ ssi $P \in L(\rho(0))$
$\rho \models \mathbf{X}\,\phi$ ssi $\rho^1 \models \phi$
$\rho \models \psi \mathbf{U} \phi$ ssi ($\exists\, i \geq 0$. ($\rho^i \models \phi$ et $\forall\, 0 \leq j < i$ on a $\rho^j \models \psi$)

seq de ss-ens de AP

$\pi \models P$ ssi $P \in \pi(0)$
$\pi \models \mathbf{X}\,\phi$ ssi $\pi^1 \models \phi$
$\pi \models \psi \mathbf{U} \phi$ ssi ($\exists\, i \geq 0$. ($\pi^i \models \phi$ et $\forall\, 0 \leq j < i$ on a $\pi^j \models \psi$)

Une histoire de mots !

$\mathbf{S} = (Q, Act, \rightarrow, q_0, AP, L)$

Lorsqu'on travaille avec LTL, **S** est vu comme un ensemble d'exécutions étiquetées:

$\rho : \underbrace{q_0 \rightarrow q_1 \rightarrow \ldots}_{\in Q^\omega}$ + $L: Q \rightarrow 2^{AP}$

Désormais, on voit **S** comme un ensemble de « séquences de sous-ensembles de AP » … on parle de **traces** de **S**.

Une **trace** est un **mot infini** sur l'alphabet 2^{AP}

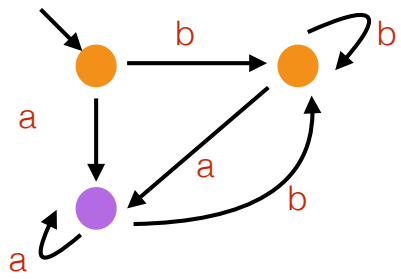
Le problème « S ⊨ ϕ ? » est une histoire de mots…

De cette histoires de mots on va en faire une histoire de langages…

Des langages, on passera bien sûr aux automates !

Automates de mots infinis

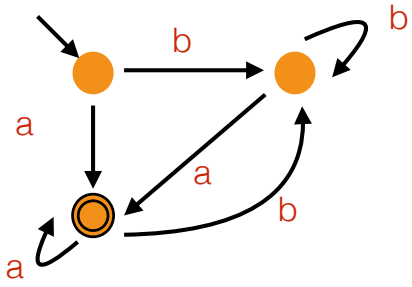
Un automate de mots fini:



: état final

-> les mots finis qui se terminent par a.

Un automate de mots infini:



: état répété

-> les mots infinis qui contiennent un nb infini de a.

-> automate de Büchi.

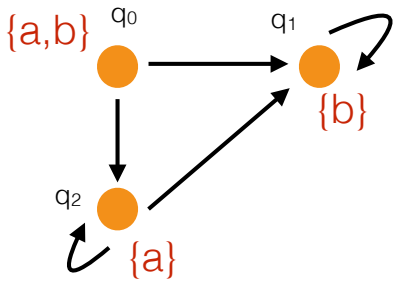
Une histoire de mots !

$\mathbf{S}$ = (Q, Act, →, q_0, AP, L)

Donc $\mathbf{S}$ est vu un ensemble de mots.

Donc $\mathbf{S}$ est vu comme un **langage** → Traces($\mathbf{S}$)

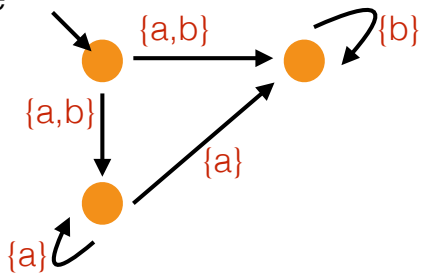
$\mathbf{S}$



Exec: $q_0.q_1^\omega \cup q_0.q_2^\omega \cup q_0.q_2^+.q_1^\omega$

Langage des traces:
$\{a,b\}.\{b\}^\omega \cup \{a,b\}.\{a\}^\omega \cup \{a,b\}.\{a\}^+.\{b\}^\omega$

Un automate de mots infinis $\mathscr{A}_\mathbf{S}$:



Traces($\mathbf{S}$) = $\mathscr{L}(\mathscr{A}_\mathbf{S})$

Automates de mots infinis

Automate de Büchi:
Un automate de Büchi est un quintuplet A=(Q,Q_0,→,Acc,∑) avec:

- Q un ensemble fini d'états,
- $Q_0 \subseteq Q$ l'ensemble des états initiaux,
- ∑ l'alphabet,
- → ⊆ Q x ∑ x Q un ensemble de transitions, et
- Acc ⊆ Q un ensemble d'états acceptants.

Un mot infini $w = w_0\, w_1\, w_2 \ldots \in \Sigma^\omega$ est accepté par A ssi il existe une séquence infinie $\rho = q_0\, q_1\, q_2 \ldots$ d'états de Q tels que:

- $q_0 \in Q_0$,
- pour tout i≥0, on a: $(q_i, w_i, q_{i+1}) \in \rightarrow$
- Si Inf(ρ) désigne les états qui apparaissent infiniment souvent le long de ρ, alors $\text{Inf}(\rho) \cap \text{Acc} \neq \varnothing$

$\mathscr{L}$(A) = l'ensemble des mots acceptés par A.

Une histoire de mots !

Une formule ϕ de LTL décrit une propriété le long d'un mot infini sur l'alphabet 2^{AP}.

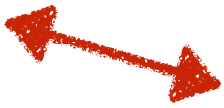
Les modèles de ϕ de LTL (notés mod(ϕ)) sont l'ensemble des mots où ϕ est vraie.

Donc mod(ϕ) = les mots infinis sur l'alphabet 2^{AP} qui vérifient ϕ.

mod(ϕ) est donc aussi un **langage** !

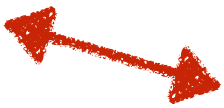
Problèmes de vérification

Model-checking:
input: un modèle (STE) $\mathbf{S}$ et une formule φ
output: oui ssi $\mathbf{S} \models \varphi$.



Traces($\mathbf{S}$) $\subseteq$ mod(φ)

Satisfaisabilité:
input: une formule φ
output: oui ssi il existe un modèle $\mathbf{S}$ t.q. $\mathbf{S} \models \varphi$.



mod(φ) $\neq \varnothing$

Problèmes de vérification pour LTL

Quel lien entre Traces($\mathbf{S}$) et mod(ϕ) ?

1) Traces($\mathbf{S}$) $\subseteq$ mod(ϕ) $\quad\quad \mathbf{S} \models \phi$

2) Traces($\mathbf{S}$) $\cap$ mod(ϕ) = $\varnothing$ $\quad\quad \mathbf{S} \models \neg\, \phi$

mod($\neg\phi$) = $(2^{AP})^{\omega} \setminus$ mod(ϕ)

$(2^{AP})^{\omega}$ = ens. de tous les mots infinis sur l'alphabet 2^{AP}.

3) sinon $\quad\quad \mathbf{S} \not\models \phi$, $\mathbf{S} \not\models \neg\phi$

Rappel: $\mathbf{S} \models \phi \iff (\rho \models \phi \quad \forall \rho \in \text{Exec}(\mathbf{S}))$

Construire les modèles de ϕ

Etant donnée ϕ, on sait construire un automate $\mathscr{A}_\phi$ qui reconnait les modèles de ϕ !

C'est-à-dire tel que:

$$\text{mod}(\phi) = \mathscr{L}(\mathscr{A}_\phi)$$

Pourquoi chercher des automates ? Parce que nous disposons de nombreux outils pour les manipuler (union, intersection, complément, inclusion, *etc.*) !

Satisfaisabilité de LTL

Comment tester si il existe un modèle pour ϕ ?

→ Tester $\mathrm{mod}(\phi)$ est non vide.

C'est-à-dire tester si $\mathscr{L}(\mathscr{A}_\phi) \neq \varnothing$?

Model-checking de LTL

Comment tester si $\mathbf{S} \models \phi$?

Tester $\mathrm{Traces}(\mathbf{S}) \subseteq \mathrm{mod}(\phi)$?

C'est-à-dire tester si $\mathscr{L}(\mathscr{A}_\mathbf{S}) \subseteq \mathscr{L}(\mathscr{A}_\phi)$?

On préfère plutôt tester si $\mathrm{Traces}(\mathbf{S}) \cap \mathrm{mod}(\neg\phi) = \varnothing$

(donc tester si $\mathscr{L}(\mathscr{A}_\mathbf{S}) \cap \mathscr{L}(\mathscr{A}_{\neg\phi}) = \varnothing$)

car tester le vide est plus simple que tester l'inclusion de deux langages, et faire l'intersection est facile.

Satisfaisabilité et Model-checking de LTL

Les deux problèmes se ramènent donc aux deux questions suivantes:

- $\mathscr{L}(\mathscr{A}_\phi) \neq \varnothing$
- $\mathscr{L}(\mathscr{A}_S) \cap \mathscr{L}(\mathscr{A}_{\neg\phi}) = \varnothing$?

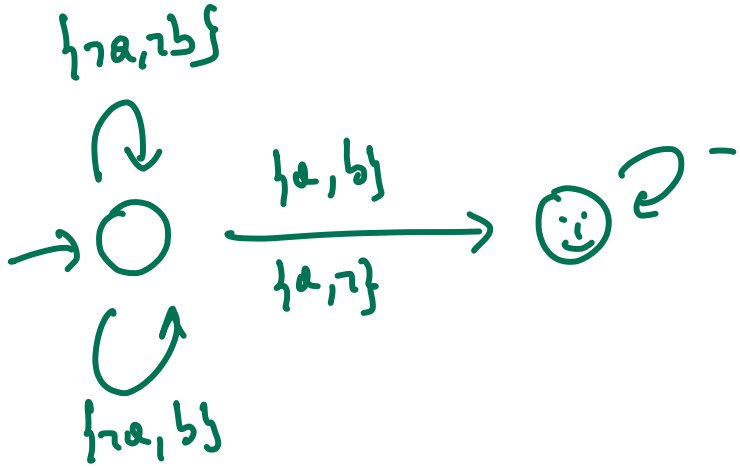
Tout repose sur les deux automates $\mathscr{A}_S$ et $\mathscr{A}_\phi$ (ou $\mathscr{A}_{\neg\phi}$).

$\mathscr{A}_S$ ne pose pas de problème: il est facile à construire à partir de S.

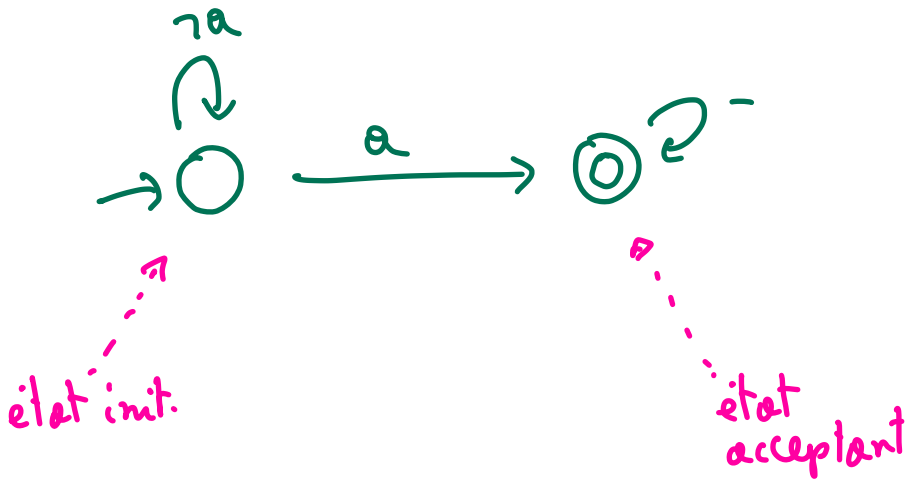
Et $\mathscr{A}_\phi$?

Fa

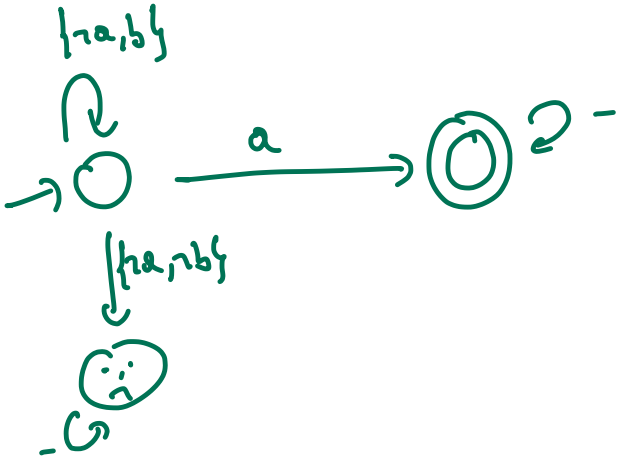
AP = {a, b}



$F a$ $\quad AP = \{a, b\}$

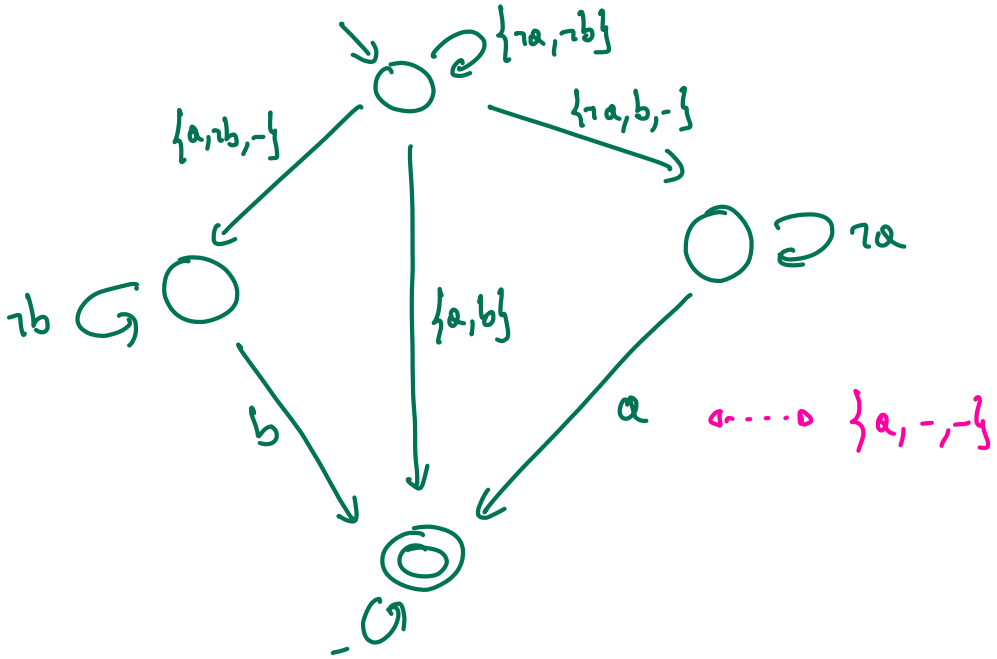


$b \, M \, a$ $\quad AP = \{a, b\}$



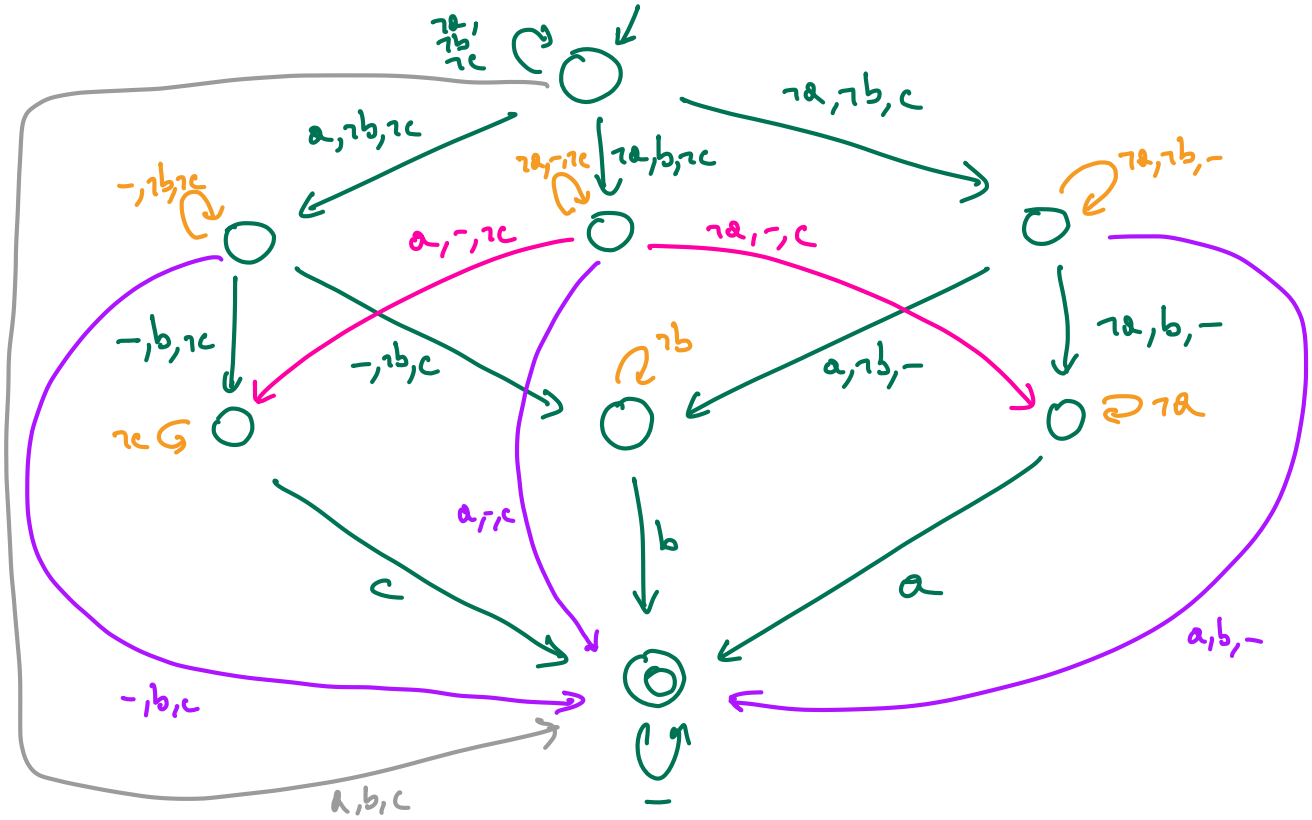
$F a \wedge F b$

$AP = \{a, b, c\}$



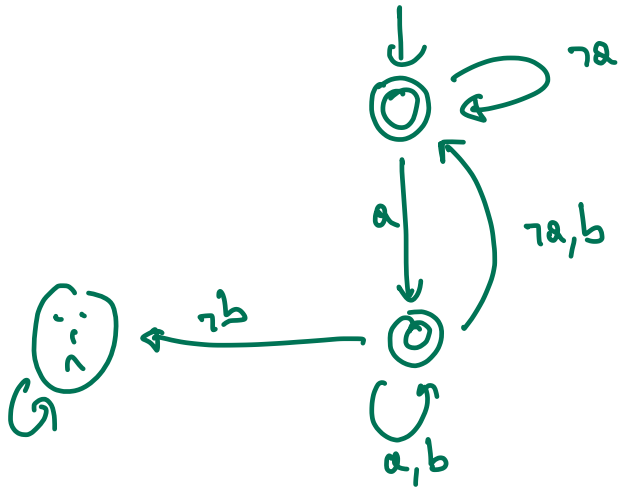
$F a \wedge F b \wedge F c$

$AP = \{a, b, c\}$



$G(a \Rightarrow Xb)$

$AP = \{a, b\}$



$GF\ a$

$AP = \{a, b, c\}$