Vérification de formules de logique temporelle

Etienne Renault & Alexandre Duret-Lutz

Avril 2015

https://www.lrde.epita.fr/~renault/teaching/imc/

Exprimer des propriétés à vérifier

Objectifs

Comment exprimer des propriétés complexes sur les systèmes ?

Exprimer des propriétés à vérifier

Objectifs

Comment exprimer des propriétés complexes sur les systèmes ?

On a vu :

Exprimer des propriétés à vérifier

Objectifs

Comment exprimer des propriétés complexes sur les systèmes ?

On a vu :

- Que vérifier des propriétés de sûreté peut se faire en explorant simplement l'espace d'état.

Exprimer des propriétés à vérifier

Objectifs

Comment exprimer des propriétés complexes sur les systèmes ?

On a vu :

- Que vérifier des propriétés de sûreté peut se faire en explorant simplement l'espace d'état.
- Pour les propriétés plus complexes deux logiques principales existent :
  - *Computational Tree Logic (CTL)* : qui utilise une vision arborescente du temps

Exprimer des propriétés à vérifier

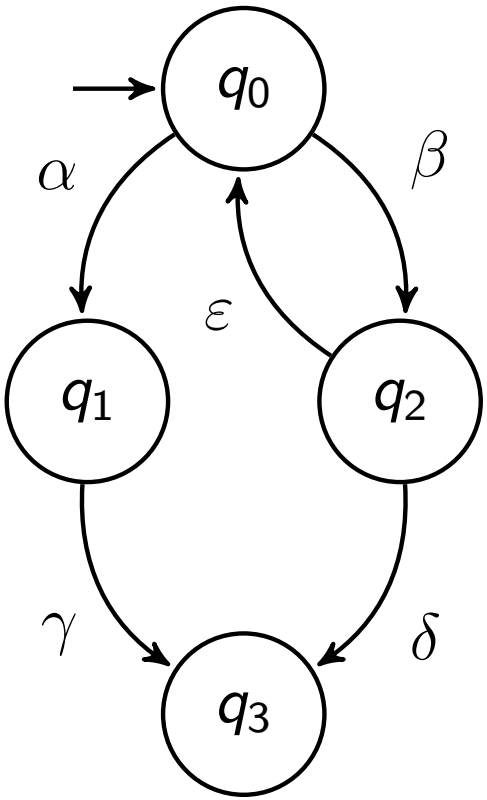
Objectifs

Comment exprimer des propriétés complexes sur les systèmes ?

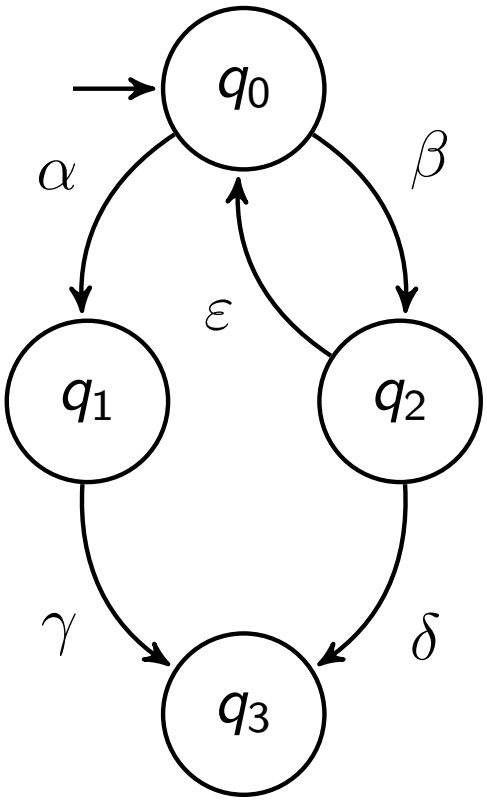
On a vu :

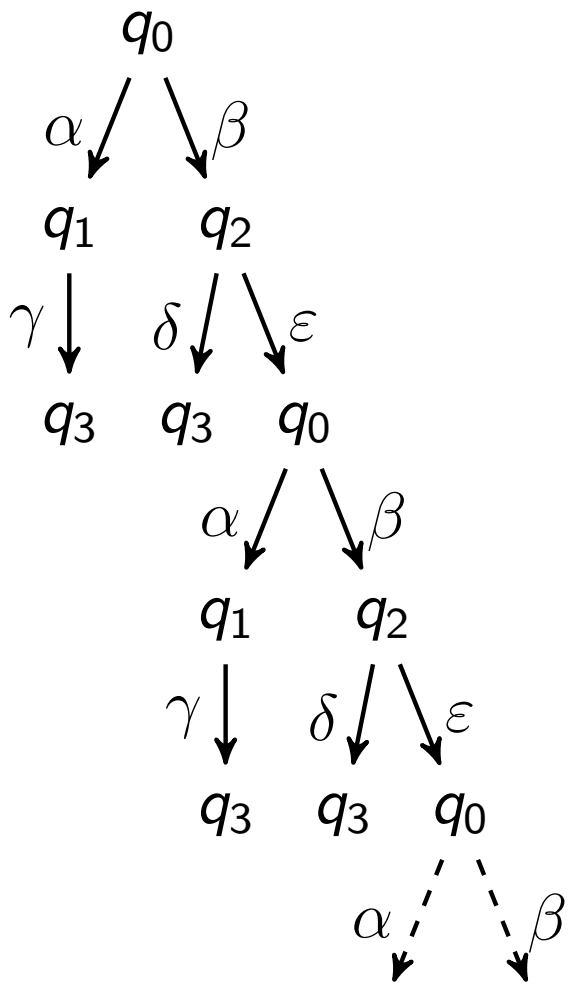
- Que vérifier des propriétés de sûreté peut se faire en explorant simplement l'espace d'état.
- Pour les propriétés plus complexes deux logiques principales existent :
  - *Computational Tree Logic (CTL)* : qui utilise une vision arborescente du temps
  - *Linear Time Logic (LTL)* : qui utilise une vision linéaire du temps

Rappels : temps arborscent ou linéaire

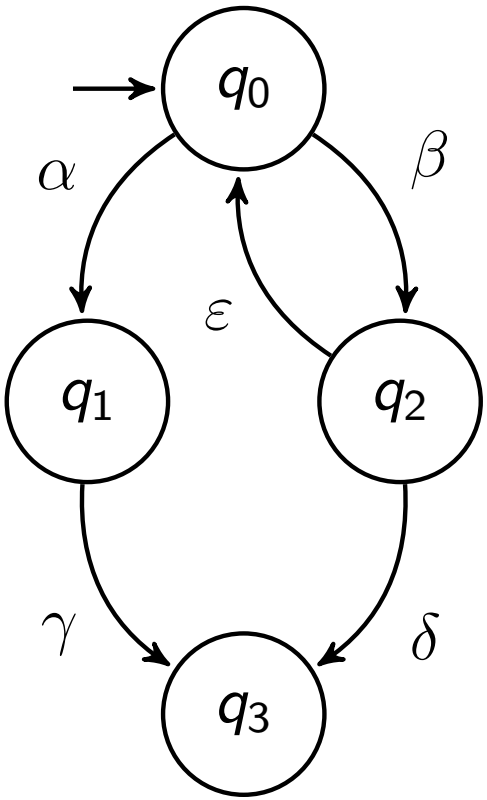


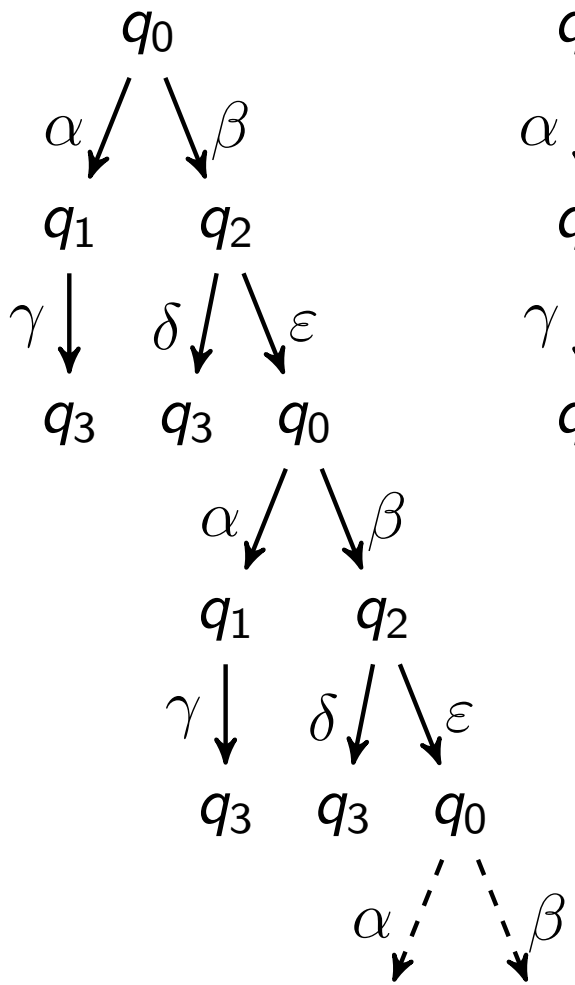
Rappels : temps arborscent ou linéaire

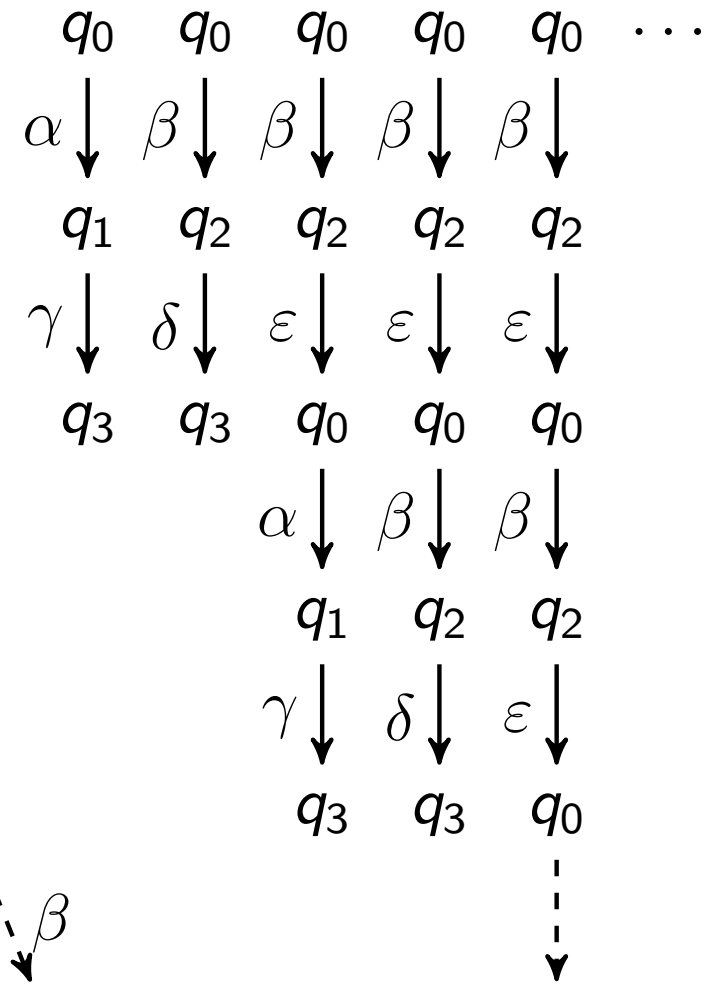




Rappels : temps arborscent ou linéaire







Syntaxe et sémantique

Quelle que soit la logique utilisée :

- la syntaxe correspond aux règles utilisées pour écrire des formules correctes (i.e., bien formées)
- la sémantique donne la signification de formules bien formée. La sémantique est utilisée pour décider si une formule bien formée est vrai ou non sur un système.

Notations

ω : premier ordinal infini

Les entiers naturels peuvent être construits avec des ensembles :

$$
\begin{aligned}
0 &= \{\} \\
1 &= \{0\} = \{\{\}\} \\
2 &= \{0, 1\} = \{\{\}, \{\{\}\}\} \\
3 &= \{0, 1, 2\} = \{\{\}, \{\{\}\}, \{\{\}, \{\{\}\}\}\} \\
&\vdots
\end{aligned}
$$

Tout entier correspond à un ensemble.
L'inclusion sur les ensembles se traduit par un ordre sur les entiers.

ω : premier ordinal infini

Les entiers naturels peuvent être construits avec des ensembles :

$$
\begin{aligned}
0 &= \{\} \\
1 &= \{0\} = \{\{\}\} \\
2 &= \{0,1\} = \{\{\},\{\{\}\}\} \\
3 &= \{0,1,2\} = \{\{\},\{\{\}\},\{\{\},\{\{\}\}\}\} \\
&\vdots \\
n+1 &= n \cup \{n\} \\
&\vdots
\end{aligned}
$$

Tout entier correspond à un ensemble.
L'inclusion sur les ensembles se traduit par un ordre sur les entiers.

ω : premier ordinal infini

Les entiers naturels peuvent être construits avec des ensembles :

$$
\begin{aligned}
0 &= \{\} \\
1 &= \{0\} = \{\{\}\} \\
2 &= \{0,1\} = \{\{\},\{\{\}\}\} \\
3 &= \{0,1,2\} = \{\{\},\{\{\}\},\{\{\},\{\{\}\}\}\} \\
&\vdots \\
n+1 &= n \cup \{n\} \\
&\vdots \\
&= \mathbb{N}
\end{aligned}
$$

Tout entier correspond à un ensemble.
L'inclusion sur les ensembles se traduit par un ordre sur les entiers.

ω : premier ordinal infini

Les entiers naturels peuvent être construits avec des ensembles :

$$
\begin{aligned}
0 &= \{\} \\
1 &= \{0\} = \{\{\}\} \\
2 &= \{0,1\} = \{\{\},\{\{\}\}\} \\
3 &= \{0,1,2\} = \{\{\},\{\{\}\},\{\{\},\{\{\}\}\}\} \\
&\vdots \\
n+1 &= n \cup \{n\} \\
&\vdots \\
\omega &= \mathbb{N}
\end{aligned}
$$

Tout entier correspond à un ensemble.
L'inclusion sur les ensembles se traduit par un ordre sur les entiers.

ω : premier ordinal infini

Les ordinaux peuvent être construits avec des ensembles :

$$
\begin{aligned}
0 &= \{\} \\
1 &= \{0\} = \{\{\}\} \\
2 &= \{0,1\} = \{\{\},\{\{\}\}\} \\
3 &= \{0,1,2\} = \{\{\},\{\{\}\},\{\{\},\{\{\}\}\}\} \\
&\vdots \\
n+1 &= n \cup \{n\} \\
&\vdots \\
\omega &= \mathbb{N} \\
\omega+1 &= \mathbb{N} \cup \{\omega\}
\end{aligned}
$$

Tout ordinal correspond à un ensemble.
L'inclusion sur les ensembles se traduit par un ordre sur les ordinaux.
(Paradoxe : les ordinaux ne forment pas un ensemble.)

ω-mots

Soient Σ un alphabet (ensemble de lettres) et $n \in \mathbb{N} \cup \{\omega\}$ un ordinal.

Une séquence de taille n (ou n-mot) de Σ est une fonction $\sigma : [\![0, n[\![\mapsto \Sigma$ associant une lettre chaque entier naturel inférieur à n.

Notations :

- Σ^n l'ensemble des séquences de taille n,
- $\Sigma^\star$ l'ensemble des séquences finies ($n < \omega$),
- Σ^ω l'ensemble des séquences infinies ($n = \omega$),
- $\Sigma^\infty = \Sigma^\star \cup \Sigma^\omega$ l'ensemble des séquences ($n \leq \omega$),
- σ^i suffixe de σ commençant à la position i : $\sigma^i(j) = \sigma(i + j)$ pour les j tels que $i + j < n$,
- ε_Σ la séquence de taille nulle sur Σ.

Expressions ω-rationnelles

L'ensemble des expressions ω-rationnelles sur Σ, par induction :

- un mot $m \in \Sigma^\infty$ est une expression ω-rationnelle ;
- si w_1 et w_2 sont deux expressions ω-rationnelles, alors w_1^ω, w_1^*, $(w_1 + w_2)$ et $(w_1 \cdot w_2)$ sont des expressions ω-rationnelles.

$\mathscr{L}(w)$, le langage d'une expression ω-rationnelle w, est défini par

$$\begin{aligned}
\mathscr{L}(m) &= \{m\} \\
\mathscr{L}\big((w_1 + w_2)\big) &= \mathscr{L}(w_1) \cup \mathscr{L}(w_2) \\
\mathscr{L}\big((w_1 \cdot w_2)\big) &= \{m_1 \cdot m_2 \mid m_1 \in \mathscr{L}(w_1), m_2 \in \mathscr{L}(w_2)\} \\
\mathscr{L}(w_1^*) &= \{\varepsilon_\Sigma\} \cup \bigcup_{n \in \mathbb{N}} \{m_0 \cdot m_1 \cdots m_n \mid \forall i \leq n,\ m_i \in \mathscr{L}(w_1)\} \\
\mathscr{L}(w_1^\omega) &= \{m_0 \cdot m_1 \cdot m_2 \cdots \mid \forall i \in \mathbb{N},\ m_i \in \mathscr{L}(w_1)\}
\end{aligned}$$

Logique Monadique

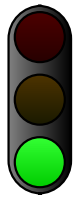
Logique des propositions : l'instant présent

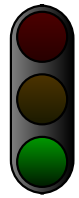
La logique propositionnelle peut caractériser un instant.

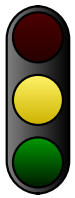
r : feu rouge allumé

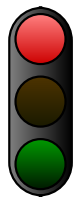
o : feu orange allumé

v : feu vert allumé

$r \wedge o \wedge v =$  , $r \wedge \neg o \wedge \neg v =$  , $\neg r \wedge \neg o \wedge v =$  ,

$\neg r \wedge \neg o \wedge \neg v =$  .

Comment dire que  précède  ?

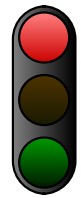
Comment dire que le système ne reste pas toujours sur  ?

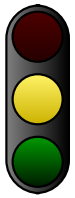
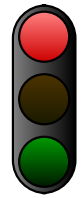
$\Rightarrow$ besoin de faire apparaître le temps

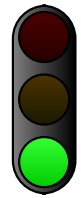
F1S : Logique monadique du 1er ordre à un succ.

Les prop. deviennent des prédicats unaires, paramétrés par le temps.

$r(t)$, $o(t)$, $v(t)$: feux allumés à l'instant t
$t+1$: instant successeur immédiat
$t \leq u$: ordre total sur les instants
$\exists t$, $\forall t$: quantificateurs du premier ordre

$\neg\forall t.(r(t) \wedge \neg o(t) \wedge \neg v(t))$: le système ne reste pas tout le temps  .

$\forall t.((\neg r(t) \wedge o(t) \wedge \neg v(t)) \rightarrow (r(t+1) \wedge \neg o(t+1) \wedge \neg v(t+1)))$:
toute configuration  est immédiatement suivie de  .

$\forall t.\exists u.(t \leq u) \wedge (\neg r(u) \wedge \neg o(u) \wedge v(u))$:
le système passe infiniment souvent par la configuration  .

S1S : Logique monadique du 2^{nd} ordre à un succ.

$r(t)$, $o(t)$, $v(t)$: feux allumés à l'instant t
0 : instant initial
$t+1$: instant successeur immédiat
$t \leq u$: ordre total sur les instants
$\exists t$, $\forall t$: quantificateurs du premier ordre
$\exists^2 X$, $\forall^2 X$: quantificateurs du second ordre
$t \in X$: appartenance d'une variable du premier ordre a une variable du second

$$\exists^2 X. \underbrace{(0 \in X \wedge (\forall t.(t \in X \rightarrow (\neg(t+1 \in X) \wedge (t+1+1 \in X)))))}_{Pair(X)}$$

$\exists^2 X.Pair(X) \wedge \forall t.(t \in X \rightarrow r(t))$: le feu rouge doit toujours être allumé aux instants pairs.

LTL

LTL : Logique Temporelle à temps Linéaire

BNF

$$\varphi ::= \top \mid \bot \mid \neg\varphi \mid \varphi \vee \psi \mid \varphi \,\mathbf{U}\, \psi \mid \mathbf{X}\, \varphi$$

Sucre Syntaxique

$$\begin{aligned}
\mathbf{F}\, \varphi &\equiv \top \,\mathbf{U}\, \varphi \\
\varphi \,\mathbf{R}\, \psi &\equiv \neg(\neg\varphi \,\mathbf{U}\, \psi) \\
\mathbf{G}\, \varphi &\equiv \bot \,\mathbf{R}\, \varphi \\
\varphi \,\mathbf{W}\, \psi &\equiv \psi \,\mathbf{R}(\varphi \vee \psi)
\end{aligned}$$

Globally

Sémantique : $w \models \mathbf{G}\,\varphi \iff \forall i, w_i \models \varphi$

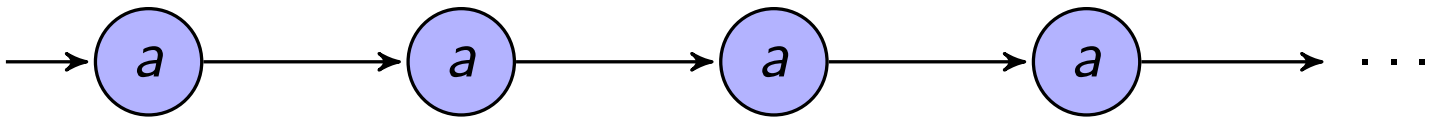
Explication : la propriété f est vérifiée tout au long du chemin w ssi tout sous-chemin de w vérifie φ

Globally

Sémantique : $w \models \mathbf{G}\,\varphi \iff \forall i, w_i \models \varphi$

Explication : la propriété f est vérifiée tout au long du chemin w ssi tout sous-chemin de w vérifie φ

Système vérifiant : $\mathbf{G}\,a$



Finally

Sémantique : $w \models F\varphi \iff \exists i, w_i \models \varphi$

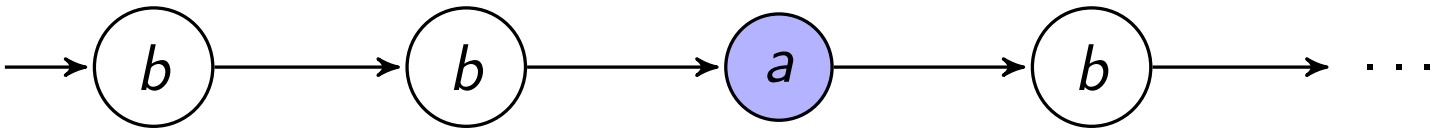
Explication : la propriété f est vérifiée au moins une fois le long du chemin c ssi l'un des sous-chemins de c vérifie f

Finally

Sémantique : $w \models F\varphi \iff \exists i, w_i \models \varphi$

Explication : la propriété f est vérifiée au moins une fois le long du chemin c ssi l'un des sous-chemins de c vérifie f

Système vérifiant : $\mathbf{F}\, a$



Next

Sémantique : $w \models X\varphi \iff c_1 \models \varphi$

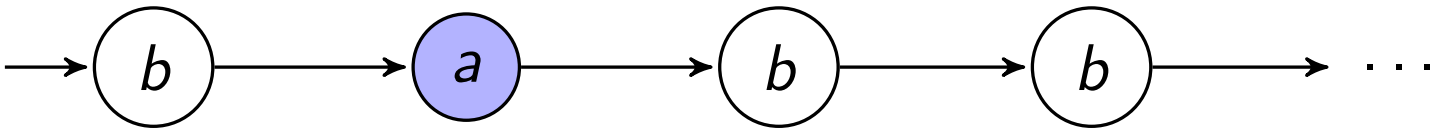
Explication : la propriété φ est vérifiée par l'état successeur le long de w

Next

Sémantique : $w \models X\varphi \iff c_1 \models \varphi$

Explication : la propriété φ est vérifiée par l'état successeur le long de w

Système vérifiant : $\mathbf{X}\, a$



Until

Sémantique : $c \models fUg \iff \exists i, c_i \models g \wedge \forall j < i, c_j \models f$

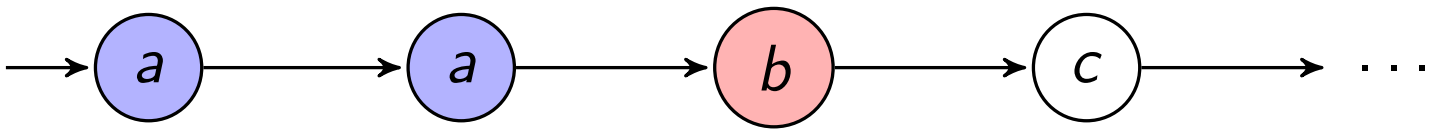
Explication : à partir d'une certaine étape du chemin c, tous les sous-chemins vérifient g, et f est vérifiée par tous les sous-chemins le précédant

Until

Sémantique : $c \models fUg \iff \exists i, c_i \models g \wedge \forall j < i, c_j \models f$

Explication : à partir d'une certaine étape du chemin c, tous les sous-chemins vérifient g, et f est vérifiée par tous les sous-chemins le précédant

Système vérifiant : $a \,\mathbf{U}\, b$



Weak until

Sémantique : $w \models fWg \iff fUg \vee Gf$

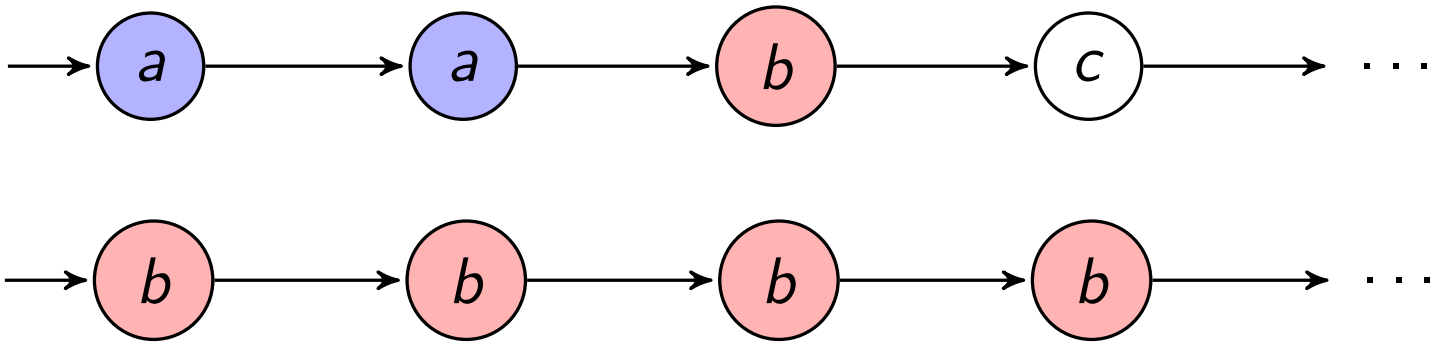
Explication : si g est vérifiée à partir d'une certaine étape du chemin, alors f a été vérifiée tout au long du chemin précédant

Weak until

Sémantique : $w \models fWg \iff fUg \vee Gf$

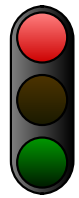
Explication : si g est vérifiée à partir d'une certaine étape du chemin, alors f a été vérifiée tout au long du chemin précédant

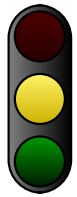
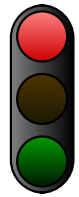
Système vérifiant : $a \mathbf{W} b$

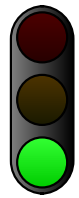


LTL : Logique Temporelle à temps Linéaire

Équivalente à la logique monadique du premier ordre à un successeur.

$\neg \mathbf{G}(r \wedge \neg o \wedge \neg v)$: le système ne reste pas tout le temps  .

$\mathbf{G}((\neg r \wedge o \wedge \neg v) \rightarrow \mathbf{X}(r \wedge \neg o \wedge \neg v))$:  est tjs imm. suivi de  .

$\mathbf{G}\,\mathbf{F}(\neg r \wedge \neg o \wedge v)$: le système passe infiniment souvent par  .

LTL : Logique Temporelle à temps Linéaire

F, **G** et **R** (Release) peuvent être vus comme du sucre :

$$\mathbf{F}\, f = \top\, \mathbf{U}\, f$$
$$f\, \mathbf{R}\, g = \neg(\neg f\, \mathbf{U}\, \neg g)$$
$$\mathbf{G}\, f = \neg\, \mathbf{F}\, \neg f = \neg(\top\, \mathbf{U}\, \neg f) = \bot\, \mathbf{R}\, f$$

D'autre part on a :

$$\neg\, \mathbf{X}\, f = \mathbf{X}\, \neg f$$
$$\neg\, \mathbf{F}\, f = \mathbf{G}\, \neg f \qquad \neg(f\, \mathbf{U}\, g) = (\neg f)\, \mathbf{R}(\neg g)$$
$$\neg\, \mathbf{G}\, f = \mathbf{F}\, \neg f \qquad \neg(f\, \mathbf{R}\, g) = (\neg f)\, \mathbf{U}(\neg g)$$

Interprétation sur une séquence

Pour toute proposition atomique p_i et toutes formules LTL f_1 et f_2, la satisfaction d'une formule LTL f par rapport à $\sigma \in (2^{AP})^\omega$ est notée $\sigma \models f$ et définie inductivement de la façon suivante :

$$
\begin{array}{ll}
\sigma \models p & \text{ssi } p \in \sigma(0) \\
\sigma \models \neg f_1 & \text{ssi } \neg(\sigma \models f_1) \\
\sigma \models f_1 \wedge f_2 & \text{ssi } \sigma \models f_1 \text{ et } \sigma \models f_2 \\
\sigma \models \mathbf{X}\, f_1 & \text{ssi } \sigma^1 \models f_1 \\
\sigma \models f_1 \,\mathbf{U}\, f_2 & \text{ssi } \exists i \geq 0 \text{ tel que } \sigma^i \models f_2 \text{ et } \forall j \in [\![0, i-1]\!],\ \sigma^j \models f_1
\end{array}
$$

Le langage de la formule φ est l'ensemble des séquence infinies sur 2^{AP} qui satisfont φ.

$$\mathscr{L}_{AP}(\varphi) = \{\sigma \in (2^{AP})^\omega \mid \sigma \models \varphi\}$$

Lien avec F1S

Notons $p_1, p_2, \ldots$ les propositions atomiques de LTL, et $p_1(t), p_2(t), \ldots$ les prédicats correspondants en F1S. Une formule LTL f correspond à l'expression F1S $[f]_0$ définie inductivement de la façon suivante (où f_1 et f_2 sont des formules LTL) :

$$
\begin{aligned}
[p_i]_t &= p_i(t) \\
[\neg f_1]_t &= \neg [f_1]_t \\
[f_1 \wedge f_2]_t &= [f_1]_t \wedge [f_2]_t \\
[\mathbf{X}\, f_1]_t &= [f_1]_{t+1} \\
[f_1 \,\mathbf{U}\, f_2]_t &= \exists u.((\forall v.((t \leq v) \wedge (v+1 \leq u)) \rightarrow [f_1]_v) \wedge [f_2]_u)
\end{aligned}
$$

(Le nom des variables u et v étant bien entendu choisi de façon unique dans le cas où plusieurs **U** sont imbriqués.)

On a $\forall \sigma \in (2^{AP})^\omega$, $\sigma \models f \iff \sigma \models [f]_0$.

Forme Normale Positive

$$\neg \mathbf{X}\, f = \mathbf{X}\, \neg f$$
$$\neg \mathbf{F}\, f = \mathbf{G}\, \neg f \qquad \neg(f\, \mathbf{U}\, g) = (\neg f)\, \mathbf{R} (\neg g)$$
$$\neg \mathbf{G}\, f = \mathbf{F}\, \neg f \qquad \neg(f\, \mathbf{R}\, g) = (\neg f)\, \mathbf{U} (\neg g)$$

Les négations ($\neg$, mais aussi $\rightarrow$ et $\leftrightarrow$) ne portent que sur les propositions atomiques.

$\neg \mathbf{G}(r \wedge \neg o \wedge \neg v) =$

$\neg \mathbf{G}\, \mathbf{F}(a \vee \neg b) =$

$\neg(a\, \mathbf{U}((b \leftrightarrow \mathbf{X}\, c)\, \mathbf{U}\, d)) =$

Forme Normale Positive

$$\neg \mathbf{X}\, f = \mathbf{X}\, \neg f$$
$$\neg \mathbf{F}\, f = \mathbf{G}\, \neg f \qquad \neg(f\, \mathbf{U}\, g) = (\neg f)\, \mathbf{R} (\neg g)$$
$$\neg \mathbf{G}\, f = \mathbf{F}\, \neg f \qquad \neg(f\, \mathbf{R}\, g) = (\neg f)\, \mathbf{U} (\neg g)$$

Les négations ($\neg$, mais aussi $\rightarrow$ et $\leftrightarrow$) ne portent que sur les propositions atomiques.

$\neg \mathbf{G}(r \wedge \neg o \wedge \neg v) = \mathbf{F}(\neg r \vee o \vee v)$

$\neg \mathbf{G}\, \mathbf{F}(a \vee \neg b) =$

$\neg(a\, \mathbf{U}((b \leftrightarrow \mathbf{X}\, c)\, \mathbf{U}\, d)) =$

Forme Normale Positive

$$\neg \mathbf{X}\, f = \mathbf{X}\, \neg f$$
$$\neg \mathbf{F}\, f = \mathbf{G}\, \neg f \qquad \neg(f\, \mathbf{U}\, g) = (\neg f)\, \mathbf{R} (\neg g)$$
$$\neg \mathbf{G}\, f = \mathbf{F}\, \neg f \qquad \neg(f\, \mathbf{R}\, g) = (\neg f)\, \mathbf{U} (\neg g)$$

Les négations ($\neg$, mais aussi $\rightarrow$ et $\leftrightarrow$) ne portent que sur les propositions atomiques.

$\neg \mathbf{G}(r \wedge \neg o \wedge \neg v) = \mathbf{F}(\neg r \vee o \vee v)$

$\neg \mathbf{G}\, \mathbf{F}(a \vee \neg b) = \mathbf{F}\, \mathbf{G}(\neg a \wedge b)$

$\neg(a\, \mathbf{U}((b \leftrightarrow \mathbf{X}\, c)\, \mathbf{U}\, d)) =$

Forme Normale Positive

$$\neg \mathbf{X}\, f = \mathbf{X}\, \neg f$$
$$\neg \mathbf{F}\, f = \mathbf{G}\, \neg f \qquad \neg(f\, \mathbf{U}\, g) = (\neg f)\, \mathbf{R} (\neg g)$$
$$\neg \mathbf{G}\, f = \mathbf{F}\, \neg f \qquad \neg(f\, \mathbf{R}\, g) = (\neg f)\, \mathbf{U} (\neg g)$$

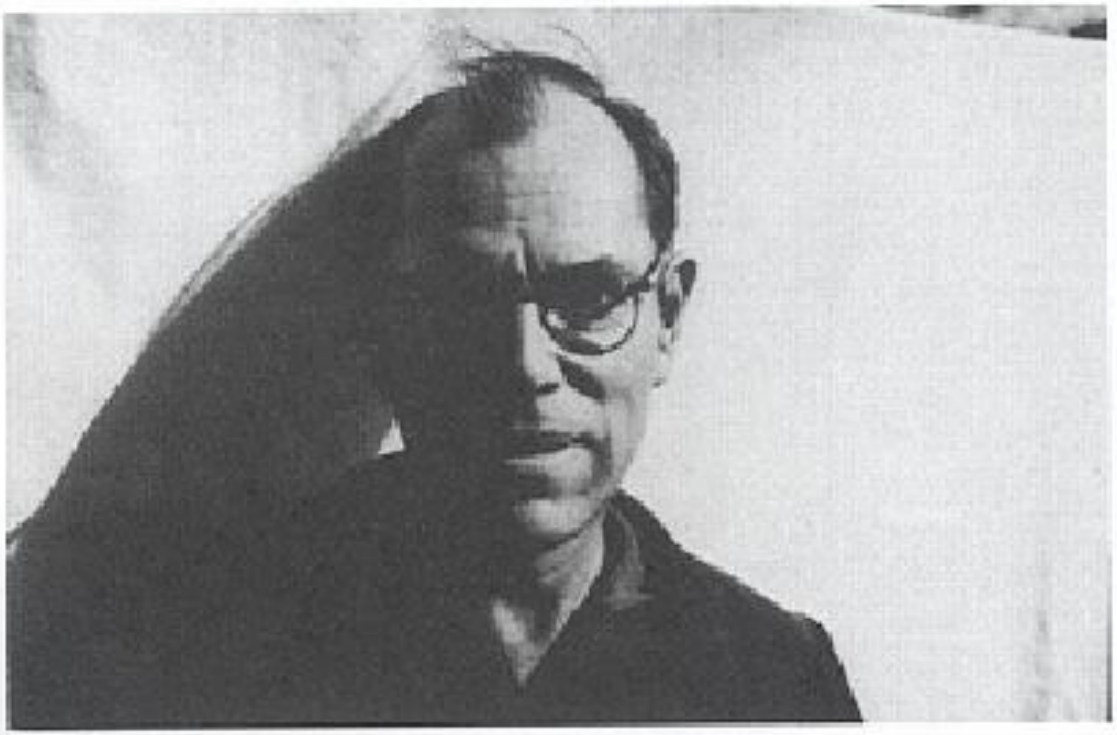
Les négations ($\neg$, mais aussi $\rightarrow$ et $\leftrightarrow$) ne portent que sur les propositions atomiques.

$\neg \mathbf{G}(r \wedge \neg o \wedge \neg v) = \mathbf{F}(\neg r \vee o \vee v)$

$\neg \mathbf{G}\, \mathbf{F}(a \vee \neg b) = \mathbf{F}\, \mathbf{G}(\neg a \wedge b)$

$\neg(a\, \mathbf{U}((b \leftrightarrow \mathbf{X}\, c)\, \mathbf{U}\, d)) = (\neg a)\, \mathbf{R}(((\neg b \wedge \mathbf{X}\, c) \vee (b \wedge \mathbf{X}\, \neg c))\, \mathbf{R} (\neg d))$

Julius Richard Büchi (1924–1984)



J. Richard Büchi, 1983

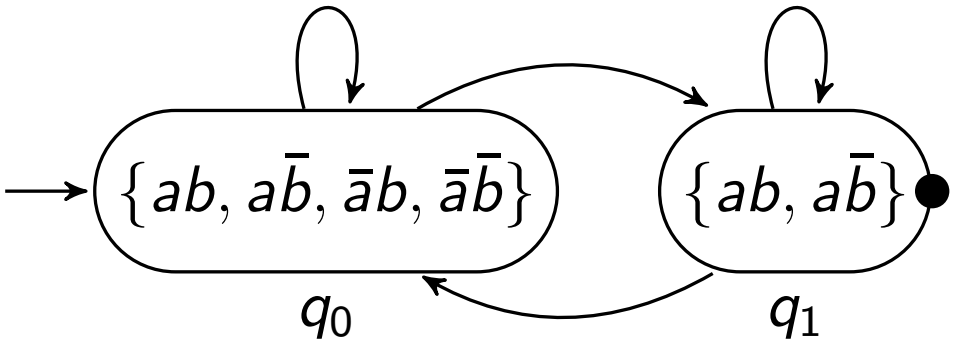
Logicien et mathématicien suisse.
Thèse à Zürich en 1950, s'installe aux USA ensuite.
Montre la décidabilité de S1S.

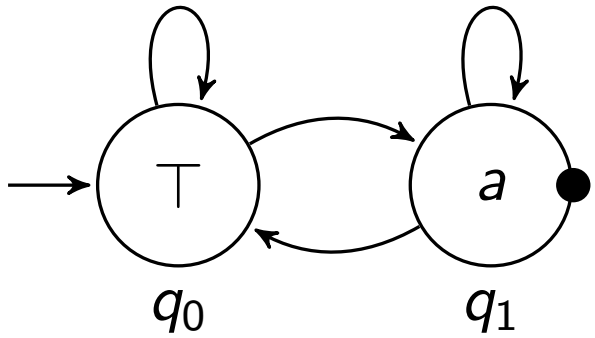
Automates de Büchi

Un automate de Büchi est un sextuplet $A = \langle \Sigma, \mathcal{Q}, \mathcal{Q}^0, \mathcal{F}, \delta, I \rangle$ où

- Σ est un alphabet,
- $\mathcal{Q}$ est un ensemble fini d'états,
- $\mathcal{Q}^0 \subseteq \mathcal{Q}$ est un ensemble d'états initiaux,
- $\mathcal{F} \subseteq \mathcal{Q}$ est un ensemble d'états d'acceptation,
- $\delta : \mathcal{Q} \mapsto 2^{\mathcal{Q}}$ est une fonction indiquant les successeurs d'un état,
- $I : \mathcal{Q} \mapsto 2^{\Sigma} \setminus \{\emptyset\}$ étiquette chaque état par un ensemble non-vide de lettres.

Exemple avec $AP = \{a, b\}$, $\Sigma = 2^{AP}$:





Automates de Büchi : langage

Les chemins de A :

$$\mathrm{Run}(A) = \{q_0 \cdot q_1 \cdot q_2 \cdots \in \mathcal{Q}^\omega \mid q_0 \in \mathcal{Q}^0 \text{ et } \forall i \geq 0,\ q_{i+1} \in \delta(q_i)\}$$

Les chemins acceptants de A sont ceux qui traversent des états d'acceptation infiniment souvent :

$$\mathrm{Acc}(A) = \{r \in \mathrm{Run}(A) \mid \forall i \geq 0,\ \exists j \geq i,\ r(j) \in \mathcal{F}\}$$

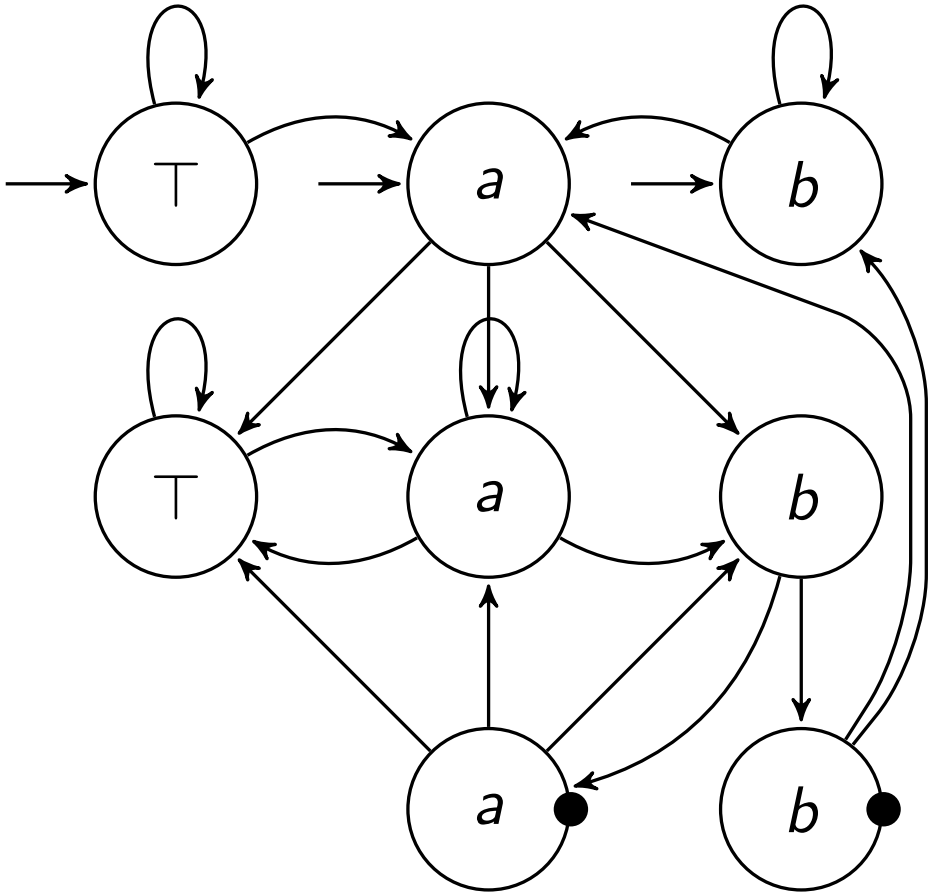
Un exécution de A est une séquence $\sigma \in \Sigma^\omega$ pour laquelle il existe un chemin acceptant $q_0 \cdot q_1 \cdots \in \mathrm{Acc}(A)$ dont les étiquettes en contiennent les lettres : $\forall i \in \mathbb{N},\ \sigma(i) \in I(q_i)$.

Le langage de A est l'ensemble des exécutions de A :

$$\mathscr{L}(A) = \{\sigma \in \Sigma^\omega \mid \exists q_0 \cdot q_1 \cdot q_2 \cdots \in \mathrm{Acc}(A),\ \forall i \in \mathbb{N},\ \sigma(i) \in I(q_i)\}$$

Automate de Büchi : plus d'états acceptants

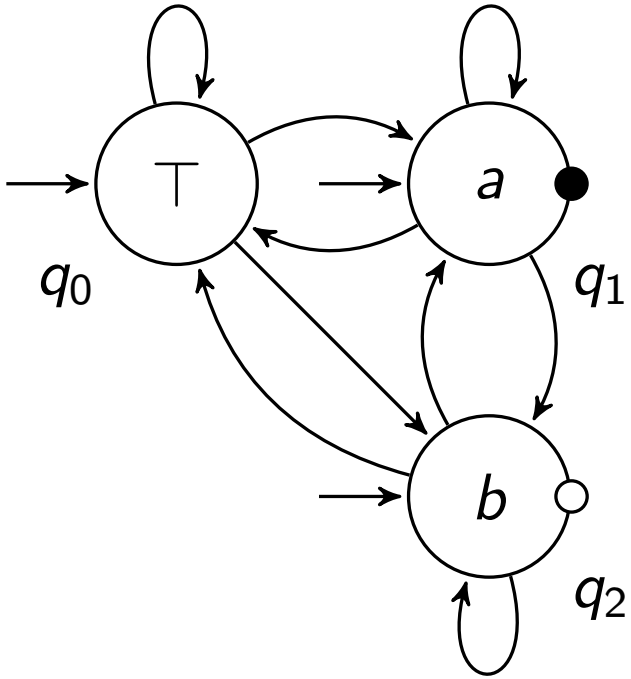
Exemple avec $AP = \{a, b\}$, $\Sigma = 2^{AP}$:



Automate de Büchi généralisé (GBA)

C'est un sextuplet $A = \langle \Sigma, \mathcal{Q}, \mathcal{Q}^0, \mathcal{F}, \delta, I \rangle$ où

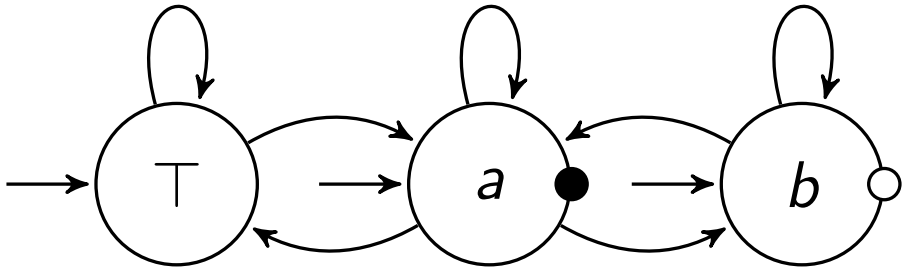
- $\mathcal{F} \subseteq 2^{\mathcal{Q}}$ est un ensemble d'ensembles d'états d'acceptation,



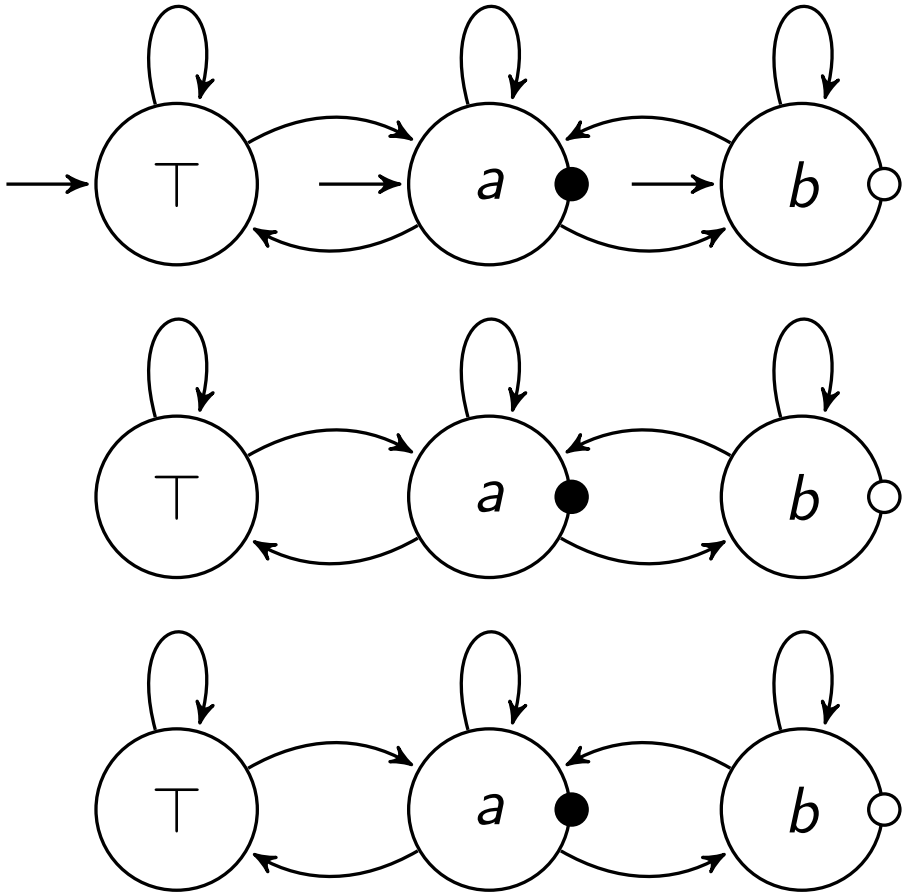
$$\mathrm{Acc}(A) = \{r \in \mathrm{Run}(A) \mid \forall F \in \mathcal{F}, \forall i \geq 0, \exists j \geq i, r(j) \in F\}$$

$$\mathscr{L}(A) = \{\sigma \in \Sigma^\omega \mid \exists q_0 \cdot q_1 \cdot q_2 \cdots \in \mathrm{Acc}(A), \forall i \in \mathbb{N}, \sigma(i) \in I(q_i)\}$$

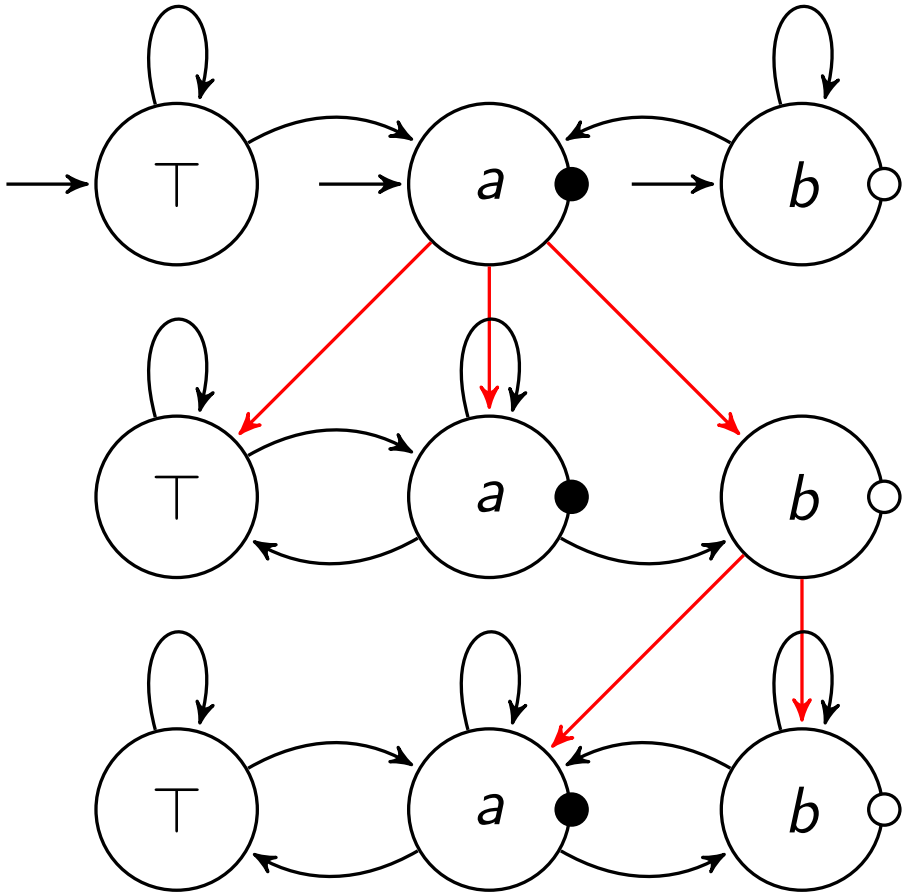
Dégénéralisation : exemple



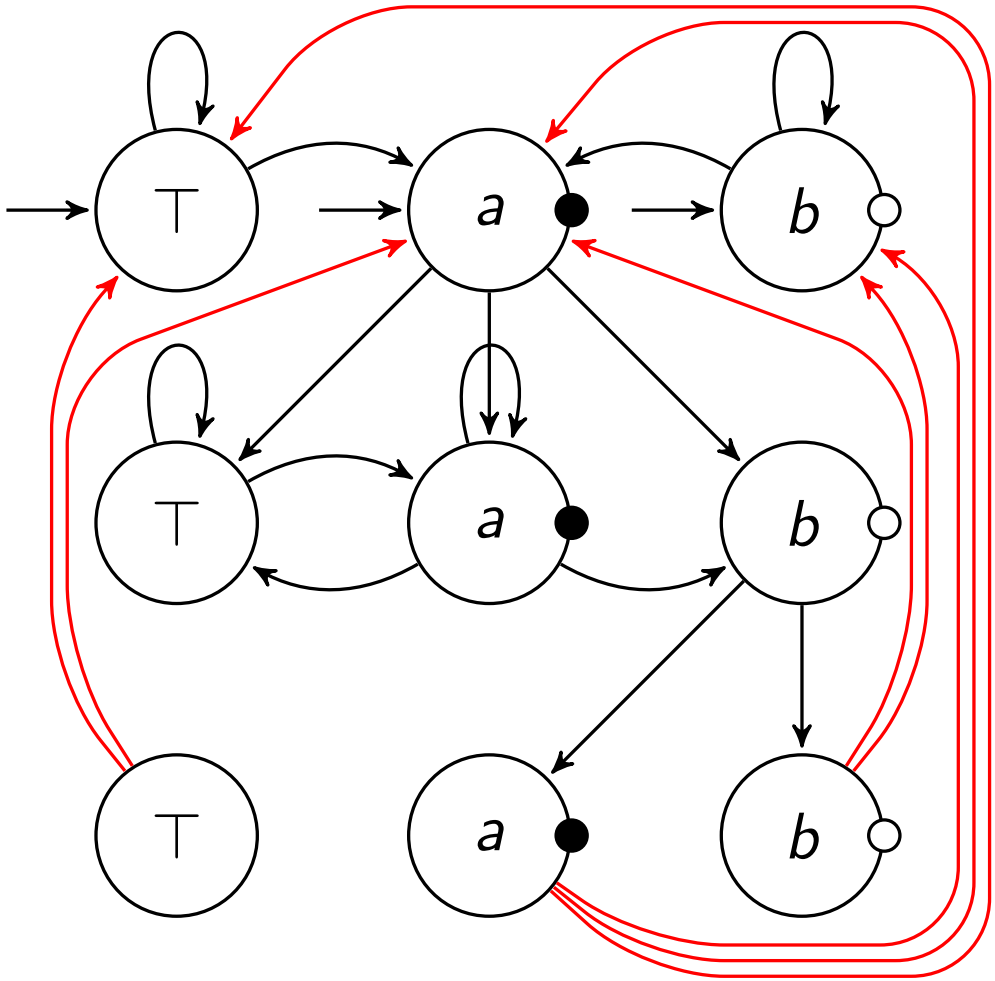
Dégénéralisation : exemple



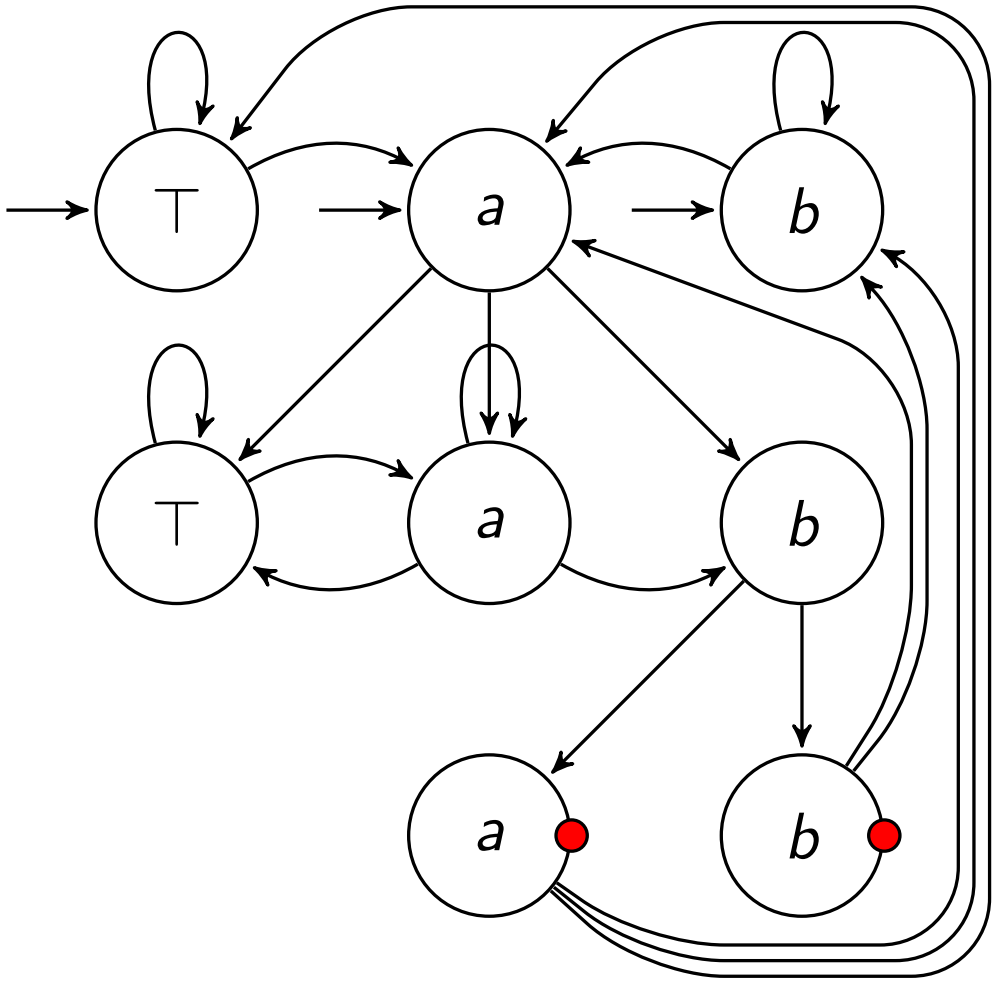
Dégénéralisation : exemple



Dégénéralisation : exemple



Dégénéralisation : exemple



Dégénéralisation : définition

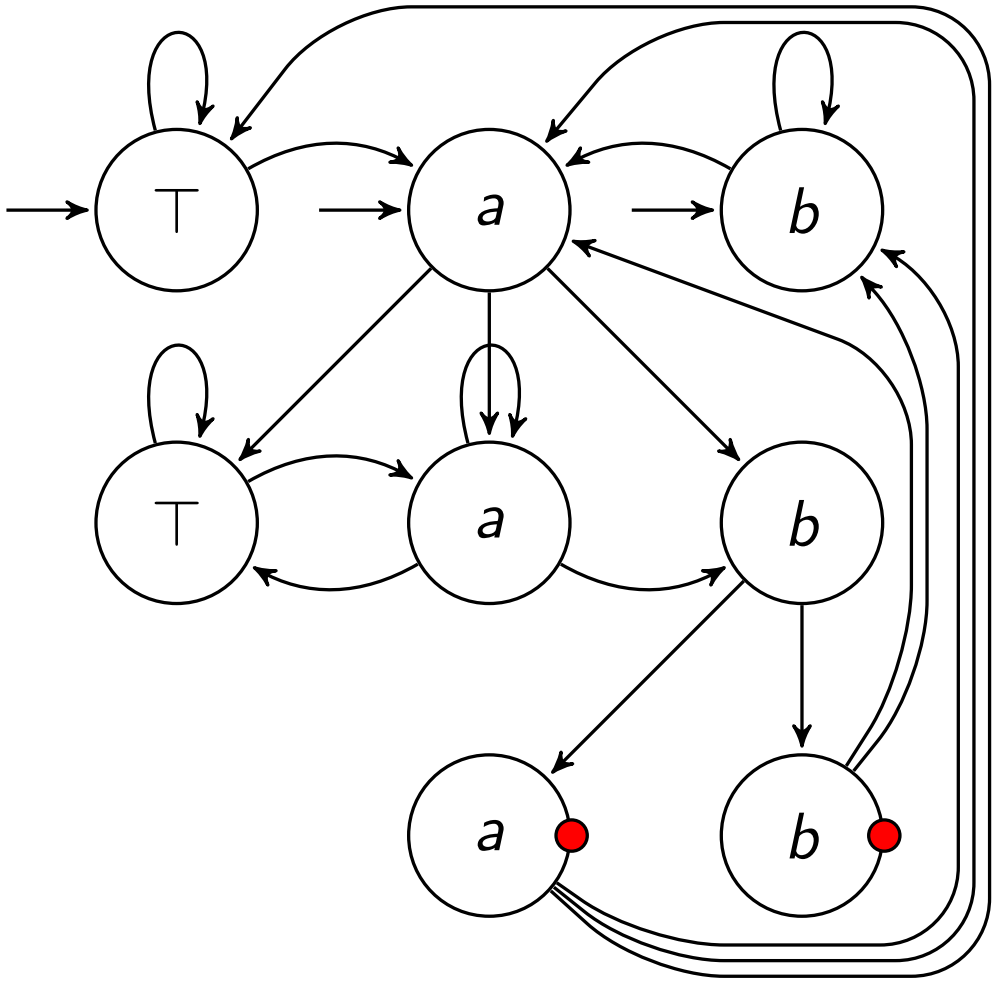
Un automate de Büchi généralisé $A = \langle \Sigma, \mathcal{Q}, \mathcal{Q}^0, \{\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_{r-1}\}, \delta, l \rangle$ peut être converti en un automate de Büchi non-généralisé $A' = \langle \Sigma, \mathcal{Q}', \mathcal{Q}'^0, \mathcal{F}', \delta', l' \rangle$ où

- $\mathcal{Q}' = \mathcal{Q} \times [\![0, r]\!]$
- $\mathcal{Q}'^0 = \mathcal{Q} \times \{0\}$
- $\mathcal{F}' = \mathcal{Q} \times \{r\}$
- $\forall (q, j) \in Q'$, $l'((q, j)) = l(q)$
- $\forall (q, j) \in Q'$, $\delta'((q, j)) = \{(q', \beta_j(q)) \mid q' \in \delta(q)\}$ avec

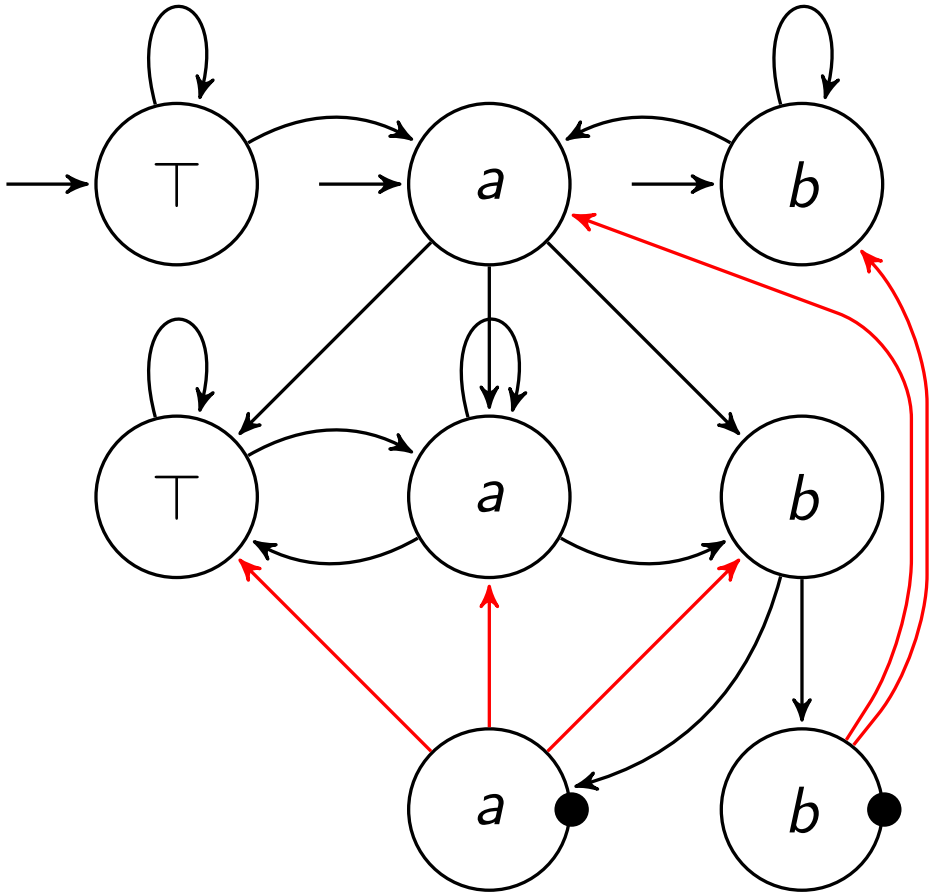
$$\beta_j(q) = \begin{cases} 0 & \text{si } j = r \\ j + 1 & \text{si } q \in \mathcal{F}_j \\ j & \text{sinon} \end{cases}$$

Si A possède n états accessibles, A' en possède au pire $n(r + 1)$.

Dégénéralisation : exemple



Dégénéralisation : exemple



Dégénéralisation : définition

Un automate de Büchi généralisé $A = \langle \Sigma, \mathcal{Q}, \mathcal{Q}^0, \{\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_{r-1}\}, \delta, I \rangle$ peut être converti en un automate de Büchi non-généralisé $A' = \langle \Sigma, \mathcal{Q}', \mathcal{Q}'^0, \mathcal{F}', \delta', I' \rangle$ où

- $\mathcal{Q}' = \mathcal{Q} \times [\![0, r]\!]$
- $\mathcal{Q}'^0 = \mathcal{Q} \times \{0\}$
- $\mathcal{F}' = \mathcal{Q} \times \{r\}$
- $\forall (q, j) \in Q'$, $I'((q, j)) = I(q)$
- $\forall (q, j) \in Q'$, $\delta'((q, j)) = \{(q', \beta_j(q)) \mid q' \in \delta(q)\}$ avec

$$\beta_j(q) = \begin{cases} 0 & \text{si } j = r \\ j+1 & \text{si } q \in \mathcal{F}_j \\ j & \text{sinon} \end{cases}$$

Si A possède n états accessibles, A' en possède au pire $n(r+1)$.

Dégénéralisation : définition

Un automate de Büchi généralisé $A = \langle \Sigma, \mathcal{Q}, \mathcal{Q}^0, \{\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_{r-1}\}, \delta, I \rangle$ peut être converti en un automate de Büchi non-généralisé $A' = \langle \Sigma, \mathcal{Q}', \mathcal{Q}'^0, \mathcal{F}', \delta', I' \rangle$ où

- $\mathcal{Q}' = \mathcal{Q} \times [\![0, r]\!]$
- $\mathcal{Q}'^0 = \mathcal{Q} \times \{0\}$
- $\mathcal{F}' = \mathcal{Q} \times \{r\}$
- $\forall (q, j) \in Q'$, $I'((q, j)) = I(q)$
- $\forall (q, j) \in Q'$, $\delta'((q, j)) = \{(q', \beta_j(q)) \mid q' \in \delta(q)\}$ avec

$$\beta_j(q) = \begin{cases} j & \text{si } j < r,\ q \notin \mathcal{F}_j, \\ \max\{n \in [\![j, r]\!] \mid \forall k \in [\![j, n]\!], q \in \mathcal{F}_k\} & \text{si } j < r,\ q \in \mathcal{F}_j, \\ 0 & \text{si } j = r,\ q \notin \mathcal{F}_0, \\ \max\{n \in [\![0, r]\!] \mid \forall k \in [\![0, n]\!], q \in \mathcal{F}_k\} & \text{si } j = r,\ q \in \mathcal{F}_0 \end{cases}$$

Si A possède n états accessibles, A' en possède au pire $n(r+1)$.

TGBA : GBA étiquetés sur les transitions

Un automate de Büchi généralisé étiqueté sur les transitions (TGBA) est un automate de Büchi dans lequel les étiquettes sont portées par les transitions et où les conditions d'acceptation de Büchi généralisées portent sur les transitions. C'est-à-dire un quintuplet $A = \langle \Sigma, \mathcal{Q}, \mathcal{Q}^0, \mathcal{F}, \delta \rangle$ où

- Σ est un alphabet,
- $\mathcal{Q}$ est un ensemble fini d'états,
- $\mathcal{Q}^0 \subseteq \mathcal{Q}$ est l'ensemble des états initiaux,
- $\mathcal{F}$ est un ensemble fini d'éléments appelés conditions d'acceptation,
- $\delta \subseteq \mathcal{Q} \times (2^\Sigma \setminus \{\emptyset\}) \times 2^\mathcal{F} \times \mathcal{Q}$ est la relation de transition de l'automate (chaque transition étant étiquetée par une formule propositionnelle ainsi qu'un ensemble de conditions d'acceptation).

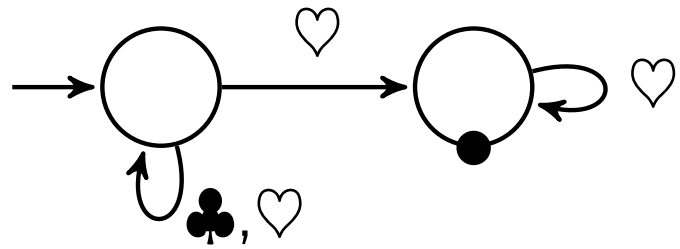
Propriétés des automates de Büchi

Les automates de Büchi, étiquetés sur les états ou les transitions, avec états ou transitions d'acceptation, généralisés ou non, sont tous aussi expressifs. I.e., ils peuvent reconnaître les mêmes langages (pas forcement avec autant d'états ou de transitions).
D'autre part les langages reconnaissables par des automates de Büchi

- sont clos par union (évident)
- sont clos par intersection (produit synchronisé)
- sont clos par complémentation (difficile à montrer)
  Büchi (1960) : construction en $2^{2^{O(n)}}$ états,
  Klarlund (1991), Safra (1992) : $2^{O(n \log n)}$, la borne théorique.
- ont leur vide décidable

Un automate de Büchi n'est pas toujours déterminisable.

Un automate de Büchi n'est pas tjrs déterminisable

$A =$ 

$\mathscr{L}(A) = (\clubsuit + \heartsuit)^\star \heartsuit^\omega$

Supp. $\exists$ automate dét. $B = \langle \{\clubsuit, \heartsuit\}, \mathcal{Q}, \delta, \{q_0\}, F \rangle$ avec un seul ensemble d'acceptation, tel que $\mathscr{L}(B) = \mathscr{L}(A)$.
$u_0 = \heartsuit^\omega \in \mathscr{L}(A)$, donc $\exists v_0$, préfixe fini de u_0 qui amène B dans F.
$u_1 = v_0 \clubsuit \heartsuit^\omega \in \mathscr{L}(A)$, donc il $\exists$ un préfixe fini $v_0 \clubsuit v_1$ de u_1 qui amène B dans F.
$\vdots$
$u_n = v_{n-1} \clubsuit \heartsuit^\omega \in \mathscr{L}(A)$, donc $\exists$ un préfixe fini $v_0 \clubsuit v_1 \clubsuit \cdots \clubsuit v_n$ de u_n qui amène B dans F.
Puisque $\mathcal{Q}$ est fini, il existe i et j, $0 \leqslant i < j$, tels que les mots $v_0 \clubsuit v_1 \clubsuit \cdots \clubsuit v_i$ et $v_0 \clubsuit v_1 \clubsuit \cdots \clubsuit v_i \clubsuit \cdots \clubsuit v_j$ mènent au même état.
Donc $m = v_0 \clubsuit v_1 \clubsuit \cdots \clubsuit v_i (\clubsuit \cdots \clubsuit v_j)^\omega$ est accepté par B.
Or m contient une infinité de $\clubsuit$, il ne peut pas appartenir à $\mathscr{L}(A)$!

Produit synchronisé

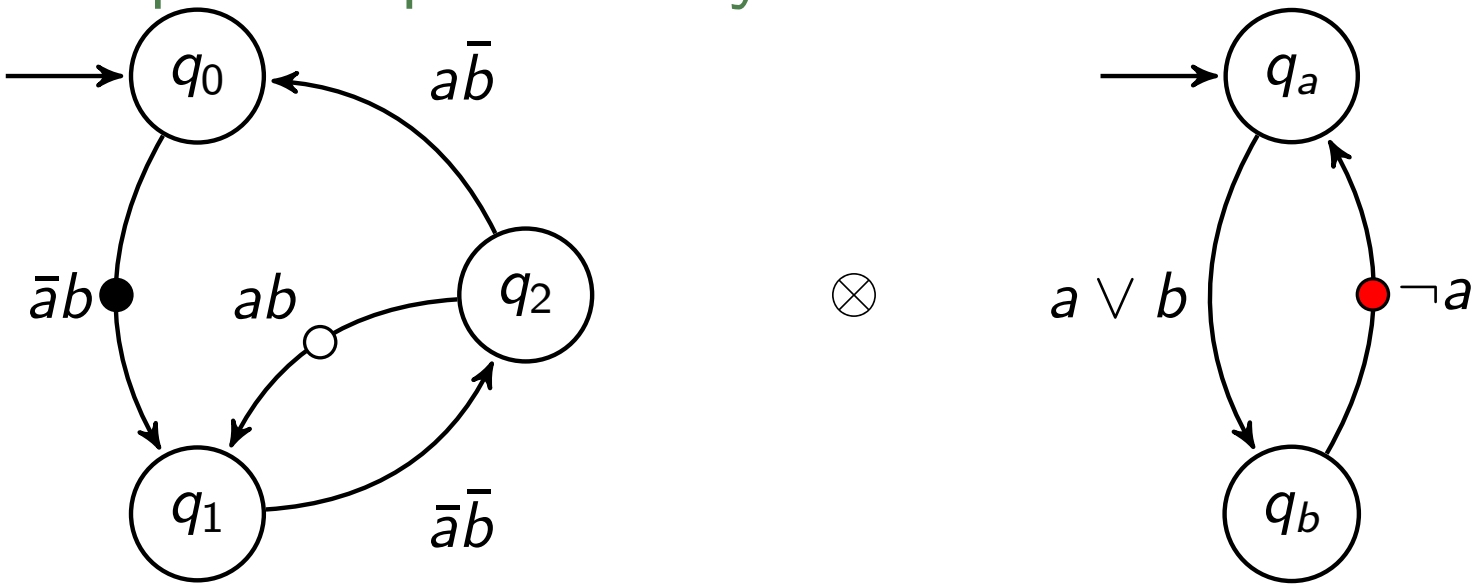
Soient $A_1 = \langle \Sigma, \mathcal{Q}_1, \mathcal{Q}_1^0, \mathcal{F}_1, \delta_1 \rangle$ et $A_2 = \langle \Sigma, \mathcal{Q}_2, \mathcal{Q}_2^0, \mathcal{F}_2, \delta_2 \rangle$ deux TGBA partageant le même ensemble de propositions atomiques.

Le produit synchronisé de A_1 et A_2 est l'automate noté $A_1 \otimes A_2 = \langle \Sigma, \mathcal{Q}, Q^0, \mathcal{F}, \delta_1 \rangle$ où

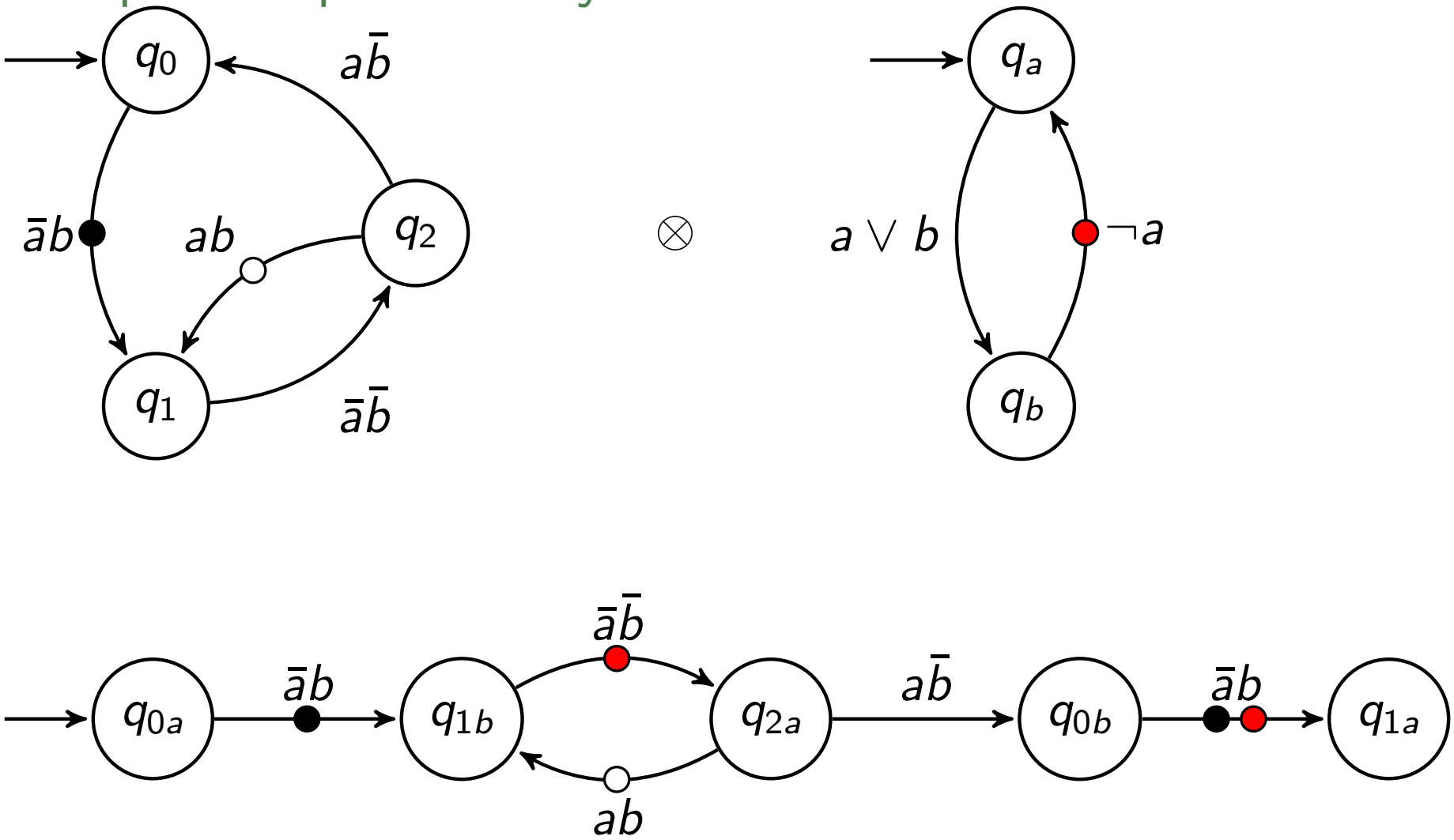
- $\mathcal{Q} = \mathcal{Q}_1 \times \mathcal{Q}_2$
- $\mathcal{Q}^0 = \mathcal{Q}_1^0 \times \mathcal{Q}_2^0$
- $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2$ à condition que $\mathcal{F}_1$ et $\mathcal{F}_2$ soient disjoints
- $\delta = \{((t_1^{\text{in}}, t_2^{\text{in}}), t_1^{\text{prop}} \cap t_2^{\text{prop}}, t_1^{\text{acc}} \cup t_2^{\text{acc}}, (t_1^{\text{out}}, t_2^{\text{out}})) \mid t_1 \in \delta_1,\ t_2 \in \delta_2,\ t_1^{\text{prop}} \cap t_2^{\text{prop}} \neq \emptyset\}$

On a alors $\mathscr{L}(A_1 \otimes A_2) = \mathscr{L}(A_1) \cap \mathscr{L}(A_2)$.

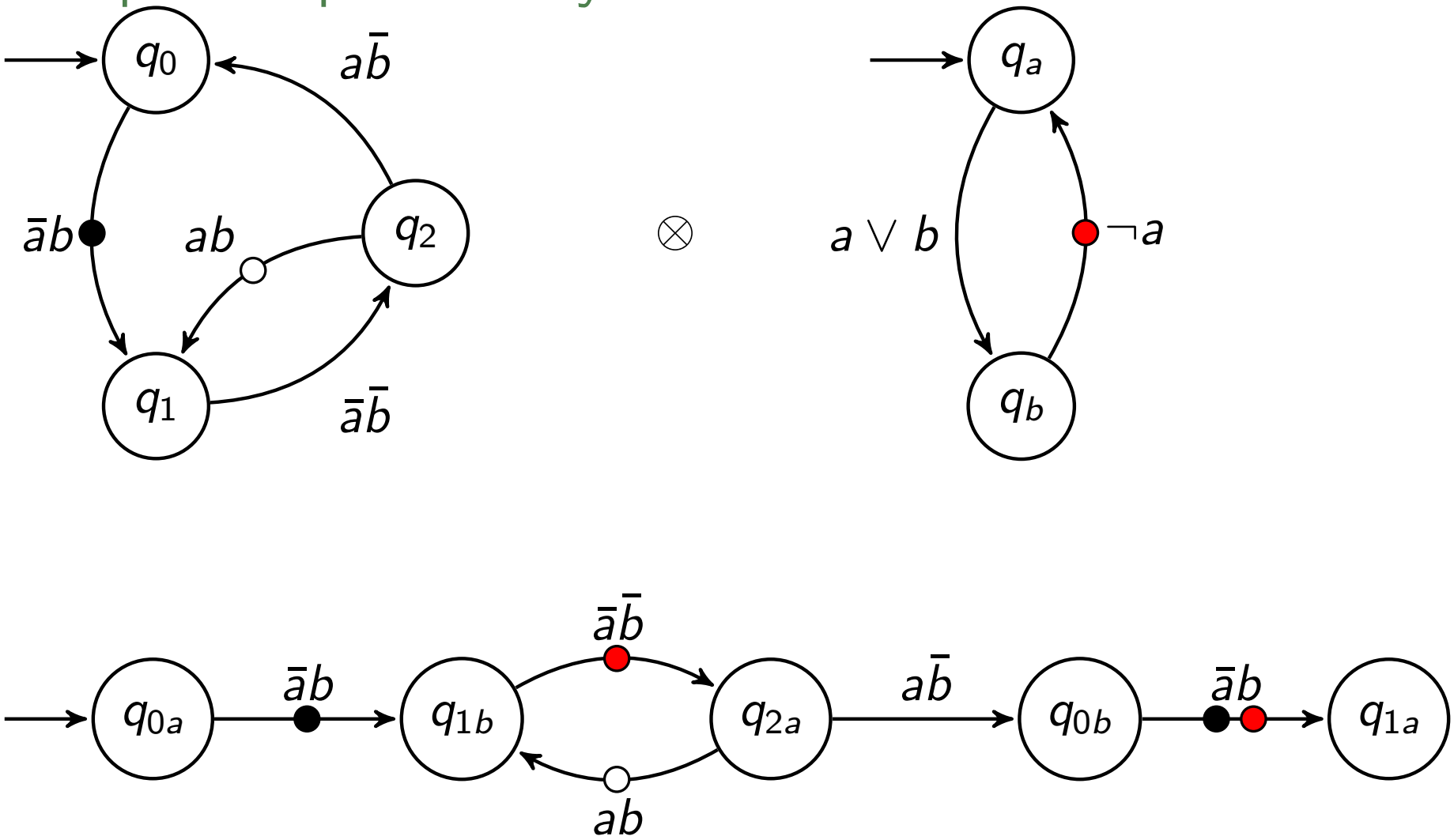
Exemple de produit synchronisé



Exemple de produit synchronisé

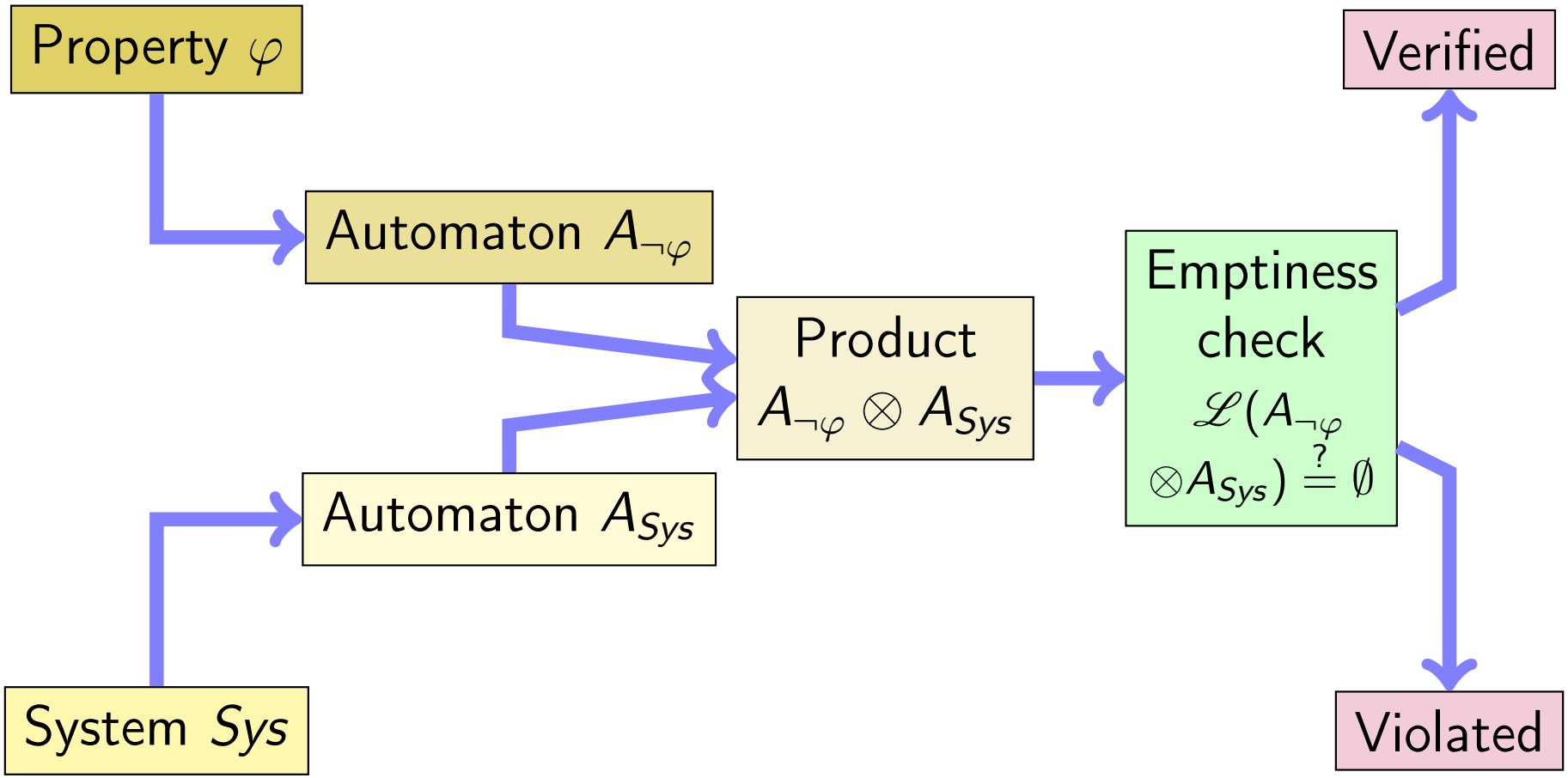


Exemple de produit synchronisé

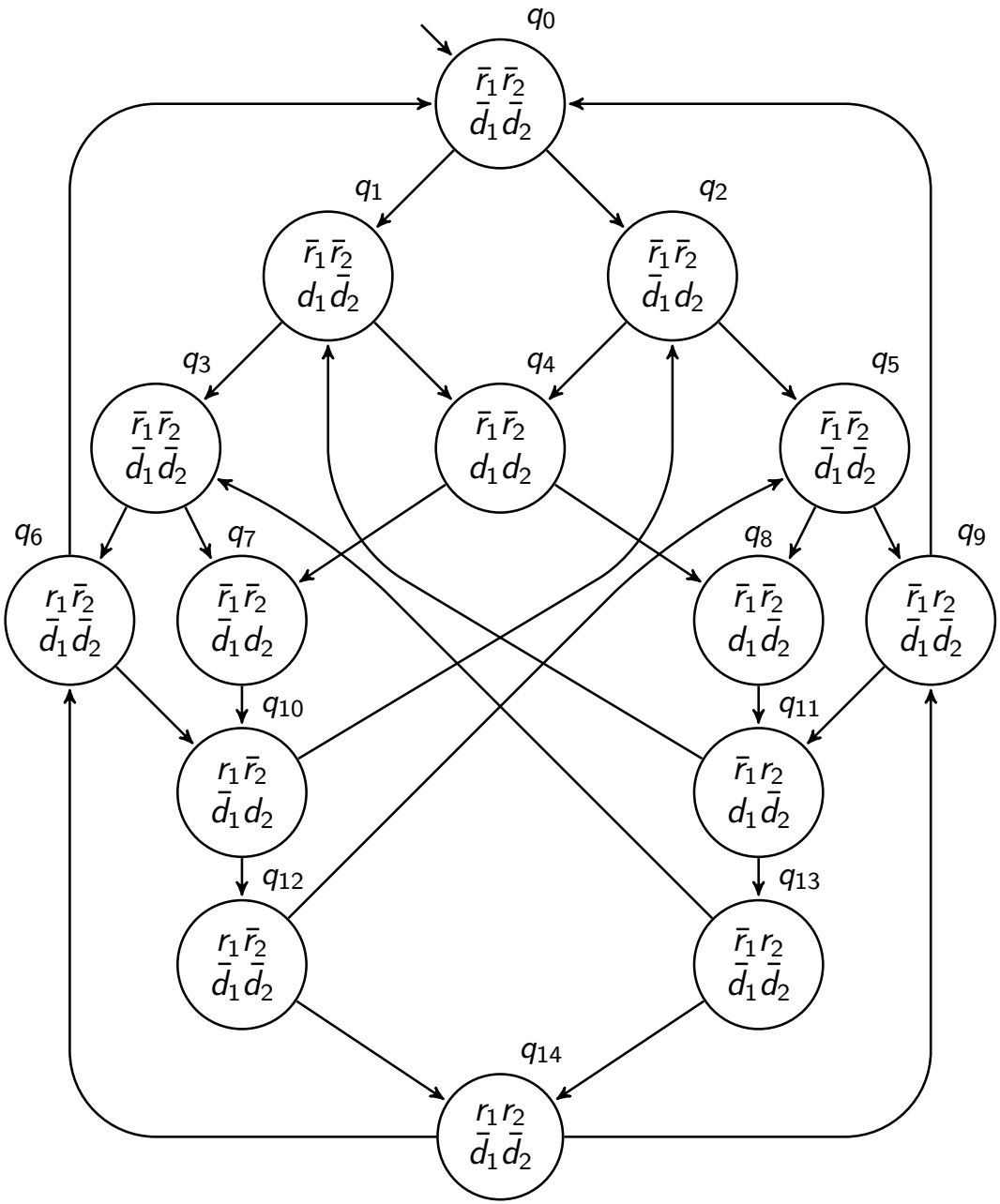


- Que devient le produit si l'on retire ● du premier automate ?

Automata-Theoretic Approach to Model Checking



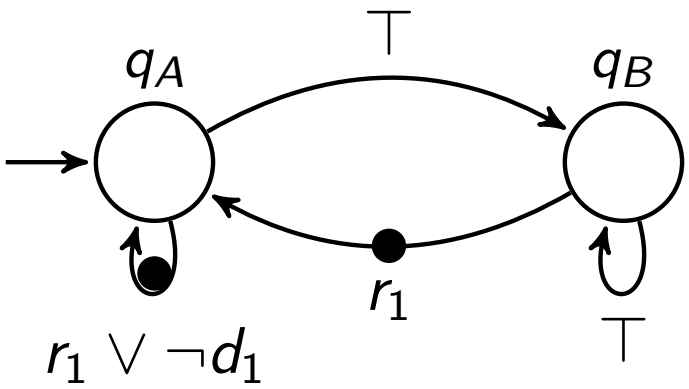
Structure de Kripke pour l'exemple



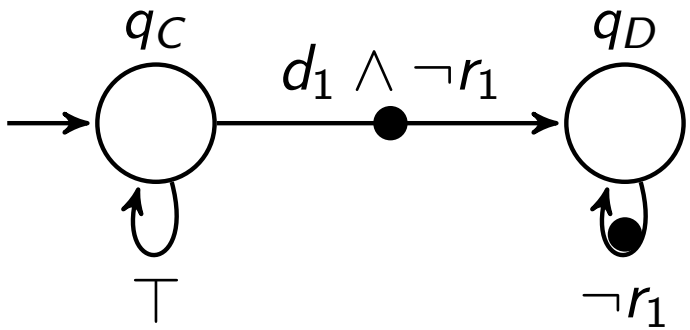
Formule à vérifier

Pour tout $i \in \{1, 2\}$, si un état vérifie d_i alors dans tous ses futurs possibles il possède un successeur qui vérifie r_i.
Par symétrie on peut se limiter à $i = 1$.

En LTL : $\mathbf{G}(d_1 \rightarrow \mathbf{F}\, r_1)$.

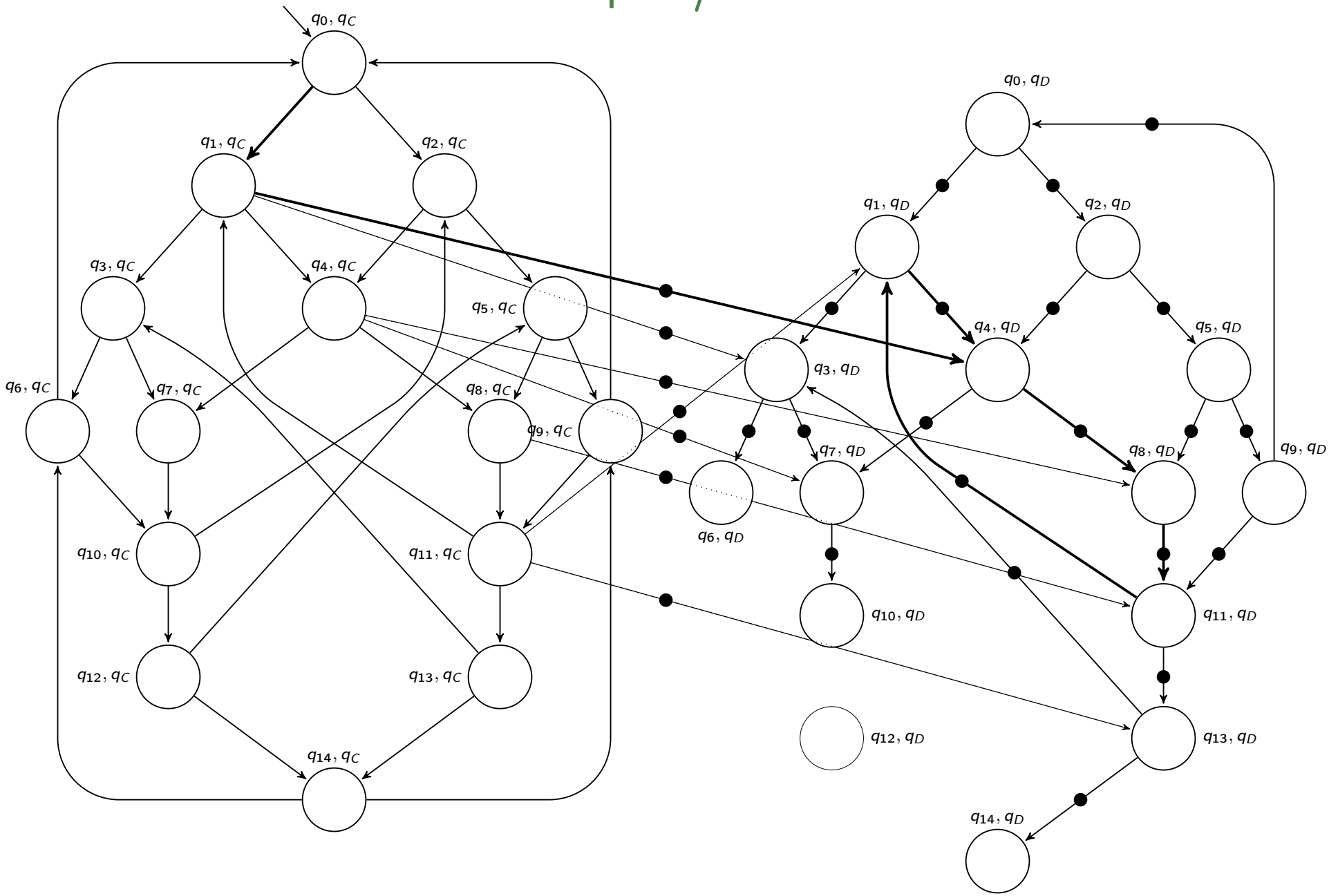


$A_{\mathbf{G}(d_1 \rightarrow \mathbf{F}\, r_1)}$



$A_{\neg\, \mathbf{G}(d_1 \rightarrow \mathbf{F}\, r_1)}$

Produit structure de Kripke/Automate



Emptiness Checks

Objectif

$$\text{Calculer si } \mathscr{L}(A_{\neg\varphi} \otimes A_{Sys}) \stackrel{?}{=} \emptyset$$

Un automate est non-vide si :

1. Il existe un chemin vers un état acceptant,
2. Il existe un cycle autour de cet état

Un prochain cours détaillera les techniques exisitantes.

Construire l'automate associé à une formule LTL

Méthodologie

Utilisation d'un *tableau sémantique* est une façon de montrer qu'une formule de logique propositionnelle n'est pas satisfiable

La preuve par tableau prend la forme d'un arbre dont les noeuds sont étiquetés par un ensemble de formules logiques

Dans le cadre de la logique propositionnelle, cet arbre peut être vu comme une mise sous forme normale disjonctive : les feuilles de l'arbre représentent des conjonctions de sous-formules atomiques à satisfaire tandis que l'arbre lui-même représente la disjonction de ces conjonctions

Une branche peut être vue comme une suite d'implications, des feuilles vers la racine ; c'est-à-dire que la satisfiabilité d'une feuille implique celle de la formule.

Méthodologie 2

Les règles de tableau indiquent comment construire le tableau à partir de la formule.

- on applique les règles,
- une branche qui contient des noeuds contradictoire est *fermée*,
- la formule est satisfiable s'il existe une branche non-fermée

Application à LTL

- Introduction de la notion d'instants présents et d'instants suivants
- Les équations étant récursives, l'arbre est théoriquement infini. Le nombre d'étiquettes étant fini, on construit une représentation finie
- Trouver une branche sans incohérence ne signifie pas que la formule est satisfiable (à cause de F et U qui introduisent des promesses).

Règles de tableau

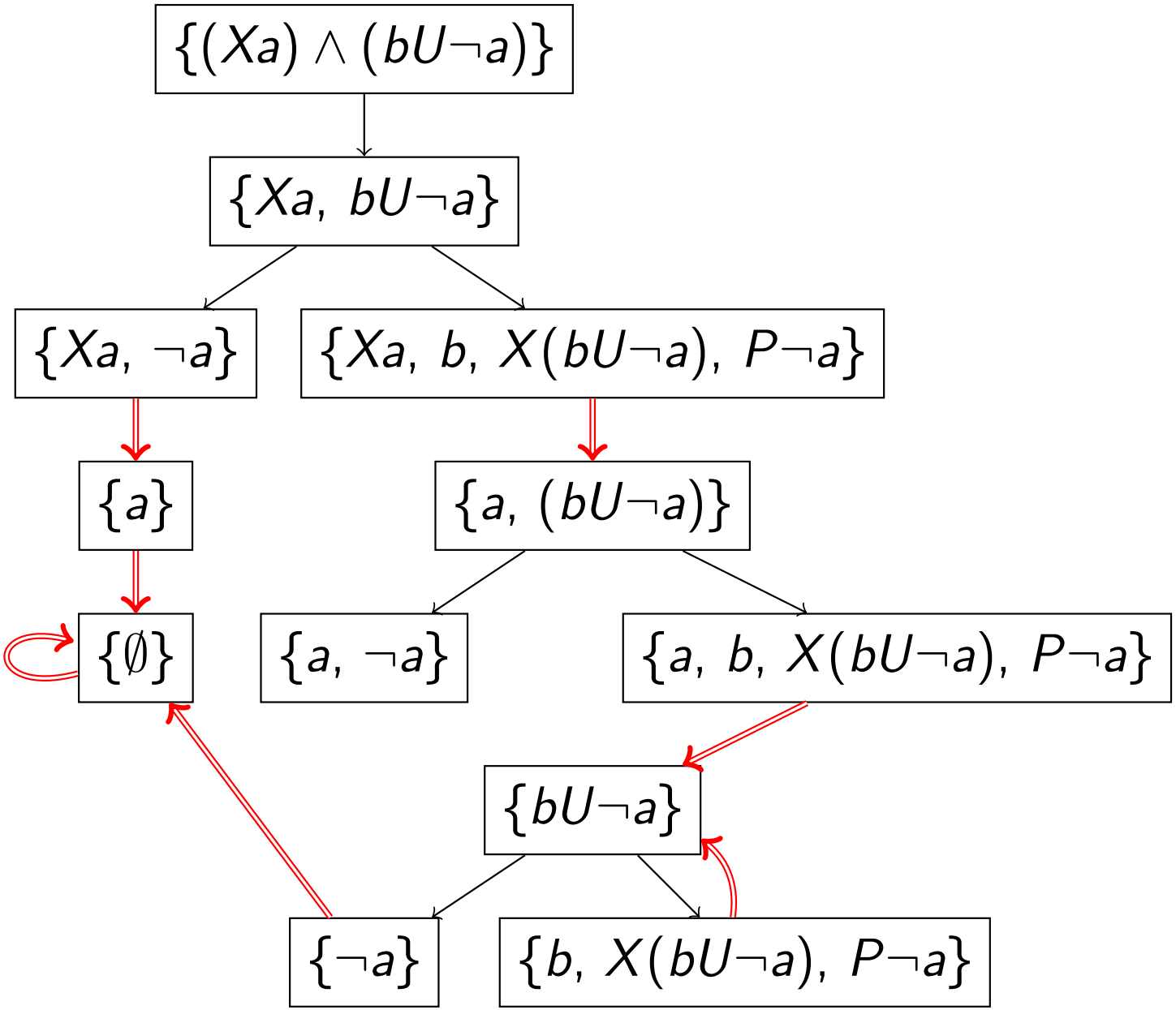
formule	1er fils	2e fils
$\Gamma \cup \{\neg\top\}$	$\Gamma \cup \{\bot\}$	
$\Gamma \cup \{\neg\bot\}$	$\Gamma \cup \{\top\}$	
$\Gamma \cup \{\neg\neg f\}$	$\Gamma \cup \{f\}$	
$\Gamma \cup \{f \wedge g\}$	$\Gamma \cup \{f, g\}$	
$\Gamma \cup \{f \vee g\}$	$\Gamma \cup \{f\}$	$\Gamma \cup \{g\}$
$\Gamma \cup \{\neg(f \wedge g)\}$	$\Gamma \cup \{\neg f\}$	$\Gamma \cup \{\neg g\}$
$\Gamma \cup \{\neg(f \vee g)\}$	$\Gamma \cup \{\neg f, \neg g\}$	

Règles de tableau

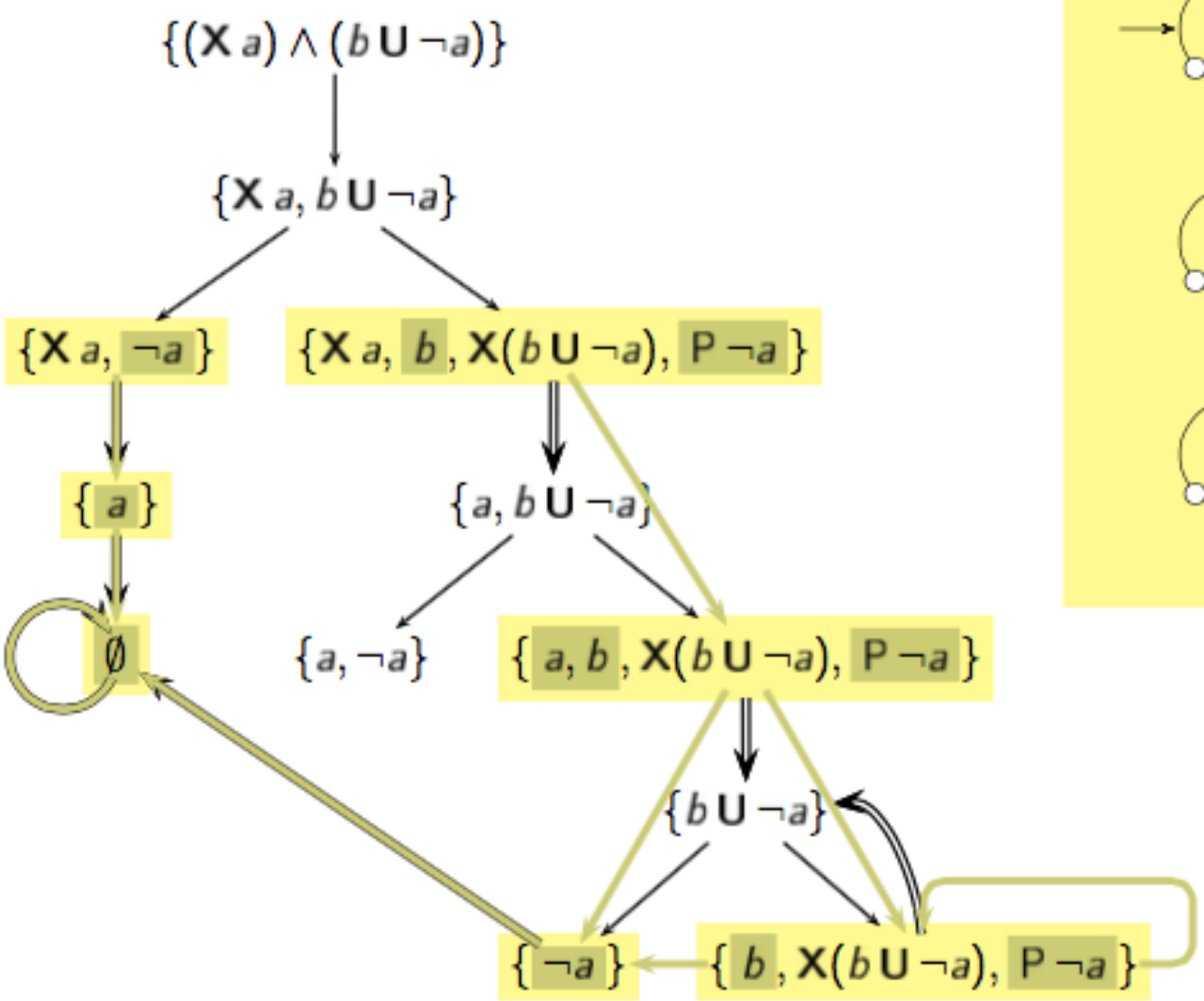
formule	1^{er} fils	2^{e} fils
$\Gamma \cup \{\neg\top\}$	$\Gamma \cup \{\bot\}$	
$\Gamma \cup \{\neg\bot\}$	$\Gamma \cup \{\top\}$	
$\Gamma \cup \{\neg\neg f\}$	$\Gamma \cup \{f\}$	
$\Gamma \cup \{f \wedge g\}$	$\Gamma \cup \{f, g\}$	
$\Gamma \cup \{f \vee g\}$	$\Gamma \cup \{f\}$	$\Gamma \cup \{g\}$
$\Gamma \cup \{\neg(f \wedge g)\}$	$\Gamma \cup \{\neg f\}$	$\Gamma \cup \{\neg g\}$
$\Gamma \cup \{\neg(f \vee g)\}$	$\Gamma \cup \{\neg f, \neg g\}$	
$\Gamma \cup \{\neg \mathbf{X} f\}$	$\Gamma \cup \{\mathbf{X} \neg f\}$	
$\Gamma \cup \{f \mathbf{U} g\}$	$\Gamma \cup \{g\}$	$\Gamma \cup \{f, \mathbf{X}(f \mathbf{U} g), \mathsf{P}\, g\}$
$\Gamma \cup \{\neg(f \mathbf{U} g)\}$	$\Gamma \cup \{\neg f, \neg g\}$	$\Gamma \cup \{\neg g, \mathbf{X} \neg(f \mathbf{U} g)\}$

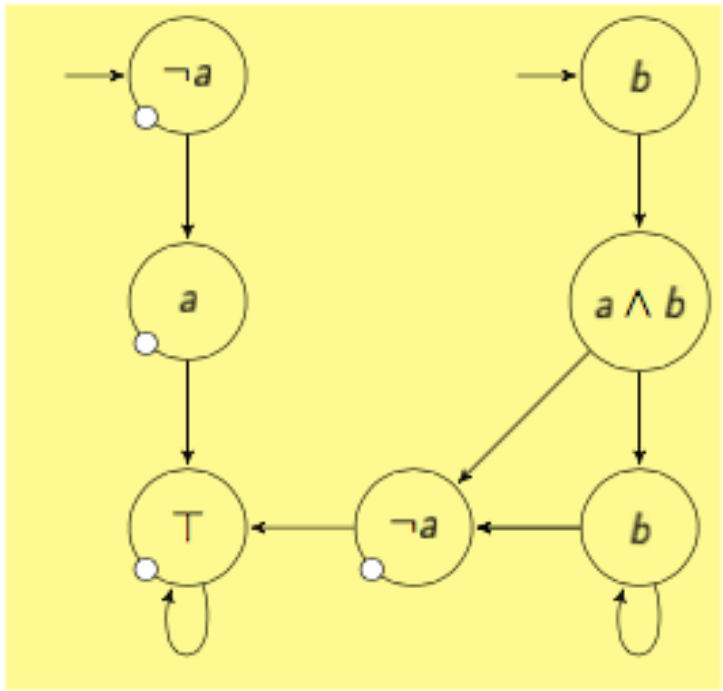
$\mathsf{P}\, g$ est une promesse que g sera vérifié

$(Xa) \wedge (bU\neg a)$



$(Xa) \wedge (bU\neg a)$





Traduisez vos formules LTL

En ligne :

`http://spot.lip6.fr/ltl2tgba.html`

En python !

DEMO !

Opérateurs

p : propriété atomique, p est vérifiée

$f \wedge g$, $f \vee g$, $\neg f$, ... : opérateurs logiques

G f : tout au long du chemin, f est vérifiée

F f : au moins une fois dans le chemin, f est vérifiée

X f : l'état successeur vérifie f

f **U** g : f est vérifiée jusqu'à ce que g soit vérifiée (g finira par être vérifiée)

f **W** g : f est vérifiée jusqu'à ce que g soit vérifiée (g ne sera peut-être jamais vérifiée)

Remarques :

- Les opérateurs manipulent un chemin
- Où est le temps arborescent ?

Computational Tree Logic

Computational Tree Logic

A partir d'un état, on s'intéresse à tous les états qui peuvent suivre

- Temps arborescent
- Indéterminisme

CTL introduit la notion de quantificateurs sur les chemins

Quantificateurs

Pour passer à une logique arborescente, chaque opérateur doit être quantitifé par :

- $\forall f$: pour tous les chemins, f est vérifiée (opérateur **A**)
- $\exists f$: il existe un chemin pour lequel f est vérifiée (opérateur **E**)

On passe d'une vue chemin par chemin (LTL) à une vue arborescente.

BNF

$$\varphi ::= \top \mid \bot \mid \neg\varphi \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid$$
$$\mathbf{A}\,\mathbf{X}\,\varphi \mid \mathbf{E}\,\mathbf{X}\,\varphi \mid \mathbf{A}\,\mathbf{F}\,\varphi \mid \mathbf{E}\,\mathbf{F}\,\varphi \mid \mathbf{A}\,\mathbf{G}\,\varphi \mid \mathbf{E}\,\mathbf{G}\,\varphi \mid$$
$$\mathbf{A}(\varphi\,\mathbf{U}\,\psi) \mid \mathbf{E}(\varphi\,\mathbf{U}\,\psi)$$

Attention ! en CTL on ne peut pas combiner les opérateurs à notre guise !

Forall

Sémantique : $e \models Af \iff \forall c \in C(e) \models f$

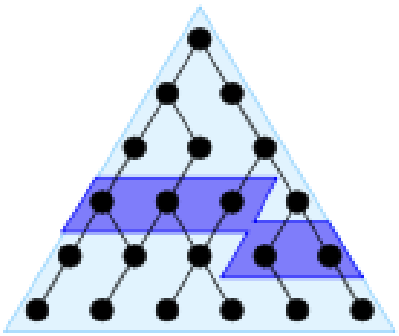
Explication : dans tout chemin partant de l'état e, f est vérifiée

Exists

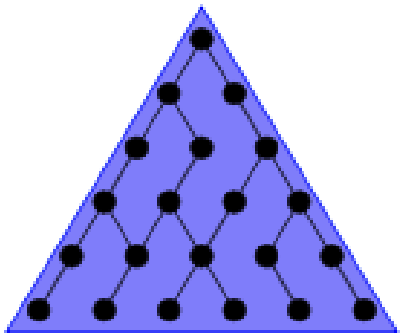
Sémantique : $e \models Ef \iff \exists c \in C(e) \models f$

Explication : dans au moins un chemin partant de l'état e, f est vérifiée

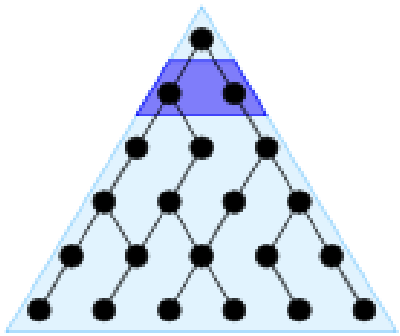
Sémantique complète



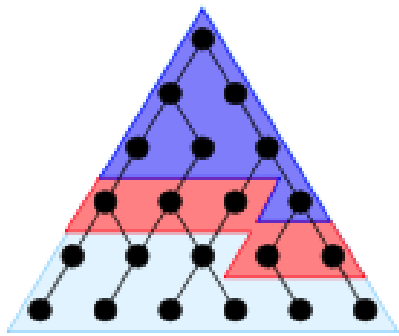
AF p



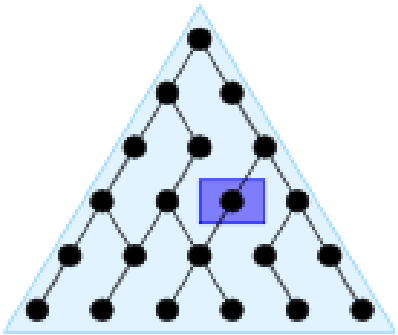
AG p



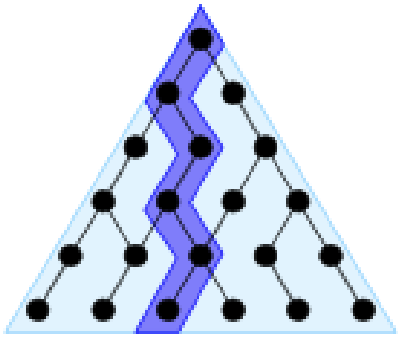
AX p



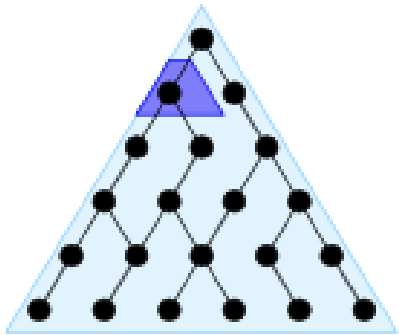
A [p U q]



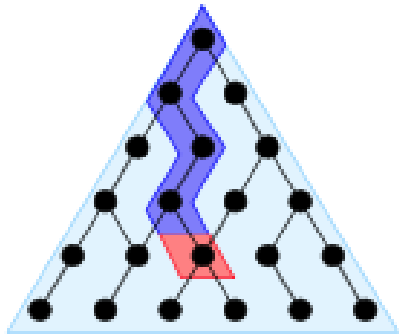
EF p



EG p



EX p



E [p U q]

Algorithme de vérification

Soit φ une formule CTL et K une structure de Kripke.

1. Étiqueter les états de K avec les sous formules de φ qui sont satisfaites depuis cet état
2. Si l'état initial de la structure de Kripke est étiqueté par φ alors φ est vérifié :

$$s_0 \in \{s \in K \mid s \models \varphi\} \implies K \models \varphi$$

Algorithme d'étiquetage (1/3)

Soit ψ une sous formule de φ, supposons que tous les états ont été déjà étiquetés par les sous formules de ψ.
On souhaite donc savoir quels états étiqueter avec ψ. Si ψ est :

1. $\bot$: alors aucun état est étiqueté avec φ
2. $ap \in AP$: tous les états pour lesquels ap est à $\top$ sont étiquetés par φ
3. $f_1 \vee f_2$: tous les états pour lesquels f_1 est à $\top$ ou f_2 est à $\top$ sont étiquetés par φ
4. $f_1 \wedge f_2$: tous les états pour lesquels f_1 est à $\top$ et f_2 est à $\top$ sont étiquetés par φ

Algorithme d'étiquetage (2/3)

6. $\neg f_1$: tous les états qui ne sont pas déjà étiquettes par f_1 sont étiquetés par φ

7. $\mathbf{E}\,\mathbf{X}\,f_1$: tous les états qui ont un successeur étiqueté par f_1 sont étiquetés par φ

8. $\mathbf{A}\,\mathbf{F}\,f_1$:
   - Si un état est étiqueté par f_1, on l'étiquette par φ,
   - Répéter : étiqueter par φ tous les états dont tous les successeurs sont étiquetés par φ jusqu'à ce qu'il n'y ait plus de changements.

Algorithme d'étiquetage (2/3)

10. $\mathbf{E}(f_1 \,\mathbf{U}\, f_2)$:
    - Si un état est étiqueté par f_2, on l'étiquette par φ,
    - Répéter : étiqueter par φ tous les états au moins un successeur est étiqueté par φ jusqu'à ce qu'il n'y ait plus de changements.
11. $\mathbf{E}\,\mathbf{G}\, f_1$:
    - étiqueter tous les états par $\mathbf{E}\,\mathbf{G}\, f_1$
    - Si un état n'est pas étiqueté par f_1, on supprime l'étiquette,
    - Répéter : supprimer l'étiquette φ de tous les états dont aucun des successeurs n'est étiqueté par φ jusqu'à ce qu'il n'y ai plus de changements.

Conclusion

- LTL et CTL n'ont pas la même expressivité : certaines formules LTL n'ont pas d'équivalent en CTL et vice-versa.
- LTL passe pour être plus intuitif
- dans la pratique LTL a été plus étudié et possède de plus d'optimisations (t.q. on-the-fly)