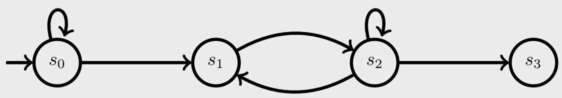
1. Systèmes de transition

Systèmes de transition

- Modélise formellement un système concret
- *États*: ensemble S (sommets)
- *Relation de transition*: $\rightarrow \subseteq S \times S$ (arcs)
- *États initiaux*: $I \subseteq S$ (où S peut débuter)



Prédécesseurs et successeurs

- *Successeurs immédiats*: $\text{Post}(s_1) = \{s_2\}$
- *Prédécesseurs immédiats*: $\text{Pre}(s_1) = \{s_0, s_2\}$
- *Successeurs*: $\text{Post}^*(s_1) = \{s_1, s_2, s_3\}$
- *Prédécesseurs*: $\text{Pre}^*(s_1) = \{s_0, s_1, s_2\}$
- *États terminaux*: s_3 car $\text{Post}(s_3) = \emptyset$

Chemins et exécutions

- *Chemin fini*: $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_1$
- *Chemin infini*: $s_1 \rightarrow s_2 \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots$
- *Ch. maximal*: ne peut être étendu, par ex. $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3$
- *Exécution*: chemin maximal qui débute par un état initial

Structures de Kripke

- Décrit les propriétés des états d'un système
- *Système de transition* $(S, \rightarrow, I)$
- *Propositions atomiques AP*
- *Fonction d'étiquetage* $L\colon S \rightarrow 2^{AP}$
- *Exemple*: si $AP = \{p, q, r\}$ et $L(s_1) = \{p, q\}$, alors s_1 satisfait p et q, mais pas r

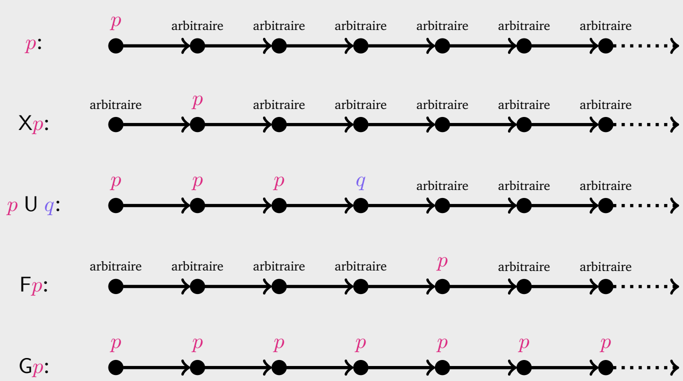
Explosion combinatoire

- Nombre d'états peut croître rapidement, par ex. $\mathcal{O}(2^n)$
- Existe techniques pour surmonter ce problème

2. Logique temporelle linéaire (LTL)

Logique

- *Intérêt*: spécifier formellement des propriétés
- *Syntaxe*: *vrai* $\mid p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid \mathsf{X}\varphi \mid \varphi \,\mathsf{U}\, \varphi \mid \mathsf{F}\varphi \mid \mathsf{G}\varphi$
- *Interprétation*: sur des traces, c.-à-d. mots infinis de $(2^{AP})^\omega$
- *Sémantique*:



Équivalences

- *Distributivité*: X, G, U (gauche) sur $\wedge$; X, F, U (droite) sur $\vee$
- *Dualité*: X dual de lui-même, F dual de G
- *Autre*: seules combinaisons de F et G: $\{\mathsf{F}, \mathsf{G}, \mathsf{FG}, \mathsf{GF}\}$

Types de propriétés

- *Invariant*: propriété toujours vraie: $\mathsf{G}\varphi$
- *Sûreté*: réfutable avec préfixe fini
- *Vivacité*: comportements vers l'infini

Vérification

- *Trace*: états d'une exécution infinie $\mapsto$ leurs étiquettes
- *Satisfaisabilité*: $\mathcal{T} \models \varphi \iff \text{Traces}(\mathcal{T}) \subseteq [\![\varphi]\!]$
- *Équité*: omettre traces triviales où un processus est ignoré
- *En pratique*: Spin (avec Promela), par ex. algorithme de Lamport, protocole de Needham-Schroeder

3. Langages ω-réguliers

Expressions ω-régulières

- *Décrivent*: les langages ω-réguliers de mots infinis
- *Syntaxe*:

$$s ::= r^\omega \mid (r \cdot s) \mid (s + s)$$
$$r ::= r^* \mid (r \cdot r) \mid (r + r) \mid a \mid \varepsilon$$

- *Exemples*:

$a(a+b)^\omega$: mots qui débutent par a,

$(ab)^\omega$: l'unique mot $abababab\cdots$

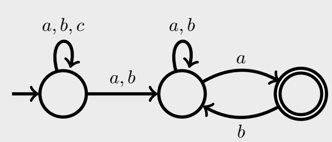
$b^*(aa^*bb^*)^\omega$: mots avec une infinité de a *et* de b

$(a+b)^*b^\omega$: mots avec un nombre fini de a

Automates de Büchi

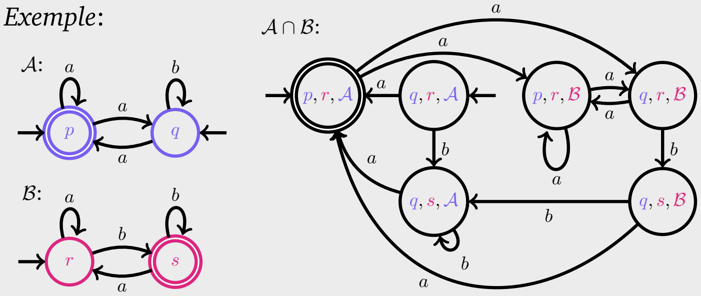
- *Définition*: automates usuels; plusieurs états initiaux
- *Langage*: mots qui visitent états acceptants ∞ souvent
- *Expressivité*: $\equiv$ expressions ω-rég.; déterminisme $\neq$ non dét.

- *Exemple*: mots tels que $\#a = \infty$, $\#b = \infty$ et $\#c \neq \infty$:



Intersection d'automates de Büchi

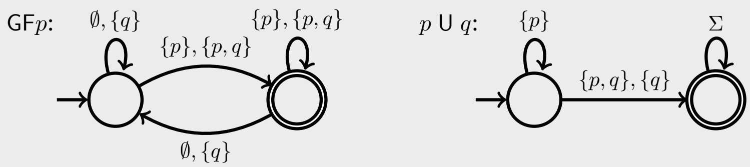
- $1^{\text{ère}}$ *idée*: simuler $\mathcal{A}$ et $\mathcal{B}$ en parallèle via produit; pas suffisant
- *Solution*: faire deux copies, alterner aux états acceptants
- *Exemple*:



4. Vérification algorithmique de formules LTL

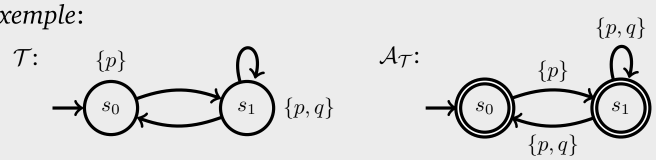
LTL vers automates

- *Alphabet*: $\Sigma := 2^{AP}$
- *Conversion*: $\varphi \to \mathcal{A}_\varphi$ (pire cas: $2^{\mathcal{O}(|\varphi|)}$ états)
- *Exemples*:



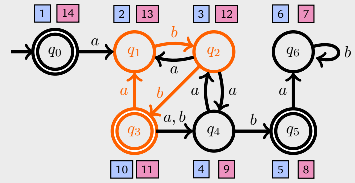
Structures de Kripke vers automates

- *Conversion*: étiquettes ≡ lettres + tout acceptant
- *Exemple*:

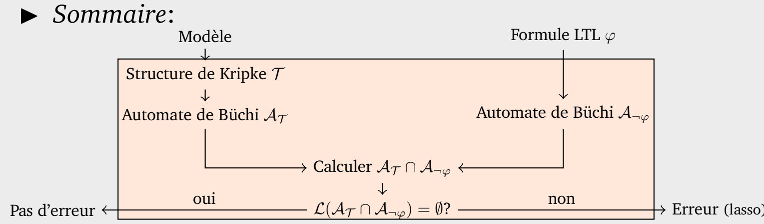


Test du vide

- *Vérification*: $\mathcal{T} \models \varphi \iff \mathcal{L}(\mathcal{A}_\mathcal{T}) \cap \mathcal{L}(\mathcal{A}_{\neg\varphi}) = \emptyset$
- *Lassos*: $\mathcal{L}(\mathcal{B}) \neq \emptyset \iff \exists q_0 \xrightarrow{*} q \xrightarrow{+} q$ où $q_0 \in Q_0, q \in F$
- *Détection*: double parcours en profondeur (temps linéaire)



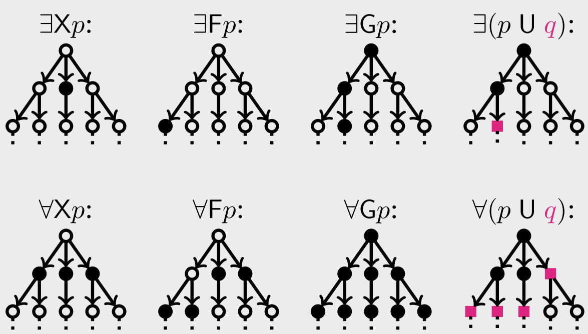
- *Mieux*: identifier les composantes fort. conn. (temps linéaire)
- *Sommaire*:



5. Logique temporelle arborescente (CTL)

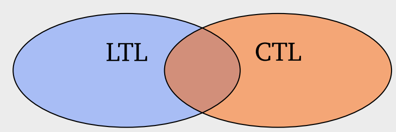
Logique

- *Intérêt*: raisonne sur le temps avec un futur indéterminé
- *Syntaxe*: *vrai* | p | $\Phi \land \Phi$ | $\Phi \lor \Phi$ | $\neg\Phi$ | $QT\Phi$ | $Q(\Phi \mathsf{U} \Phi)$
  où $Q \in \{\exists, \forall\}$, $T \in \{\mathsf{X}, \mathsf{F}, \mathsf{G}\}$
- *Interprétation*: sur l'arbre de calcul d'une structure de Kripke
- *Sémantique*:



Propriétés d'un système

- *Satisfiabilité dépend des états*: $[\![\Phi]\!] := \{s \in S : s \models \Phi\}$
- *Spécification*: $\mathcal{T} \models \Phi \iff I \subseteq [\![\Phi]\!]$
- *Expressivité*: incomparable à LTL



Équivalences

- *Distributivité*:

$$\exists\mathsf{F}(\Phi_1 \lor \Phi_2) \equiv (\exists\mathsf{F}\Phi_1) \lor (\exists\mathsf{F}\Phi_2)$$
$$\forall\mathsf{G}(\Phi_1 \land \Phi_2) \equiv (\forall\mathsf{G}\Phi_1) \land (\forall\mathsf{G}\Phi_2)$$

- *Attention*: pas équiv. si on change les quantificateurs
- *Dualité*: effet d'une négation: $\exists \leftrightarrow \forall$, $\mathsf{X} \leftrightarrow \mathsf{X}$ et $\mathsf{F} \leftrightarrow \mathsf{G}$
- *Idempotence*: $QT\,QT\Phi \equiv QT\Phi$ où $Q \in \{\exists, \forall\}$, $T \in \{\mathsf{F}, \mathsf{G}\}$

6. Vérification algorithmique de formules CTL

Algorithme

- *Approche*: calculer $[\![\Phi']\!]$ pour chaque sous-formule Φ' de Φ
- *Vérification*: tester $I \subseteq [\![\Phi]\!]$
- *Forme normale existentielle* plus simple, mais pas nécessaire
- *Complexité*: $\mathcal{O}((|S| + |{\to}|) \cdot |\Phi|)$ avec bonne implémentation
- *En pratique*: NuSMV + langage de description de haut niveau

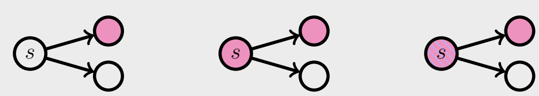
Logique propositionnelle

- *Règles récursives*:

$$[\![\mathit{vrai}]\!] = S,$$
$$[\![p]\!] = \{s \in S : p \in L(s)\},$$
$$[\![\Phi_1 \land \Phi_2]\!] = [\![\Phi_1]\!] \cap [\![\Phi_2]\!],$$
$$[\![\neg\Phi]\!] = S \setminus [\![\Phi]\!].$$

Opérateurs temporels existentiels

- *Calcul de* $[\![\exists\mathsf{X}\Phi]\!]$: $\{s \in S : \mathrm{Post}(s) \cap [\![\Phi]\!] \neq \emptyset\}$
- *Calcul de* $[\![\exists\mathsf{G}\Phi]\!]$: $T \subseteq [\![\Phi]\!]$ max. t.q. $\forall s \in T : \mathrm{Post}(s) \cap T \neq \emptyset$
- *Calcul de* $\exists(\Phi_1 \mathsf{U} \Phi_2)$: $T \supseteq [\![\Phi_2]\!]$ min. t.q.
  $s \in [\![\Phi_1]\!] \land \mathrm{Post}(s) \cap T \neq \emptyset) \implies s \in T$



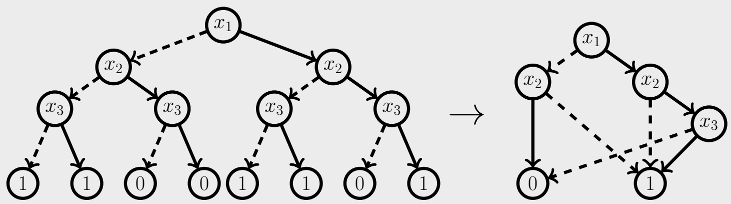
Optimisations

- *Autres opérateurs*: $\forall$ et F s'implémentent directement; nécessaire pour obtenir un algorithme polynomial
- *Points fixes*: temps linéaire si calculs directs sans raffinements itératifs; par ex. calcul de composantes fortement connexes

7. Vérification symbolique : diagrammes de décision binaire

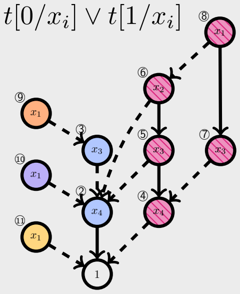
Diagramme de décision binaire

- *But*: représenter des fonctions booléennes de façon compacte
- *Utilité*: manipuler efficacement de grands ensembles d'états
- *Propriétés*:
  - graphe dirigé acyclique
  - sommets étiquetés par variables ordonnées sauf 0 et 1
  - les chemins respectent l'ordre des variables
  - sommets *uniques* et *non redondants* ($lo(u) \neq hi(u)$)
- *Canonicité*: pas deux BDDs pour la même fonction booléenne



Manipulation

- *Représentation*: tableau associatif *sommet* $\leftrightarrow (x_i, lo, hi)$
- *Ajout d'un sommet*: temps constant avec `make`(x_i, lo, hi)
- *Construction*: par substitutions récursives avec `build`(φ)
- *Op. bool.*: application récursive « synchronisée » avec `apply`$_\bigcirc$
- *Quantif.*: `exists`$x_i \in \{0,1\} : t$ obtenu via $t[0/x_i] \vee t[1/x_i]$
- *Complexité*: polynomiale sauf pour `build`



Vérification

- *État*: représenté par identifiant binaire
- *Transition*: paire d'identifiants binaires
- *Ensemble*: représenté par sommet de BDD
- *Logique prop.*: manipulation de BDD
- *Opérateurs temporels*: via calculs de Post ou Pre sur BDD
- *Satisfiabilité*: $I \subseteq [\![\Phi]\!] \Leftrightarrow I \cap \overline{[\![\Phi]\!]} = \emptyset \Leftrightarrow$ `apply`$_\wedge(u_I, u_{\overline{[\![\Phi]\!]}}) = 0$

8. Systèmes avec récursion

Contexte

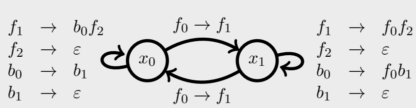
- *Espace d'états*: pile d'appel ou d'éléments (+ valeurs locales)
- *Défi*: gérer un nombre infini ou arbitraire d'états
- *Approche*: construction et analyse de systèmes à pile

Systèmes à pile

- *Définition*: états P, alphabet Γ, transitions $\{p \xrightarrow{a \to u} q, \ldots\}$
- *But*: décrire un ensemble de piles (et non accepter des mots)
- *Configuration*: $\langle p, w\rangle \in P \times \Gamma^* \;\mapsto$ p + pile $w_1, \vdots, w_n$
- *Exemple de modélisation*:

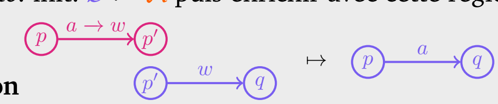
```
      bool x ∈ {faux,vrai}

      foo():              bar():
f0:     x = ¬x:        b0:  si x: foo()
f1:     si x: foo()    b1:  retourner
        sinon: bar()
f2:     retourner
```



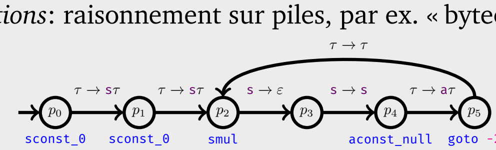
Calcul de prédécesseurs/successeurs

- *Déf.*: $\mathrm{Pre}^*(C) := \bigcup_{i \geq 0} \mathrm{Pre}^i(C)$ et $\mathrm{Post}^*(C) := \bigcup_{i \geq 0} \mathrm{Post}^i(C)$
- *Représentation*: symbolique de C avec un $\mathcal{P}$-automate $\mathcal{A}$
- *Idée*: (états initiaux = états de $\mathcal{P}$) + mots sur alphabet de pile
- *Décrit*: $\mathit{Conf}(\mathcal{A}) := \{\langle p, w\rangle : p \xrightarrow{w}_{\mathcal{A}} \circledcirc\}$
- *Algorithme*: permet de calculer $\mathrm{Pre}^*(\mathit{Conf}(\mathcal{A}))$ par saturation
- *Approche*: init. $\mathcal{B} := \mathcal{A}$ puis enrichir avec cette règle:



Vérification

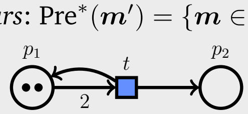
- *Approche*: système $\mapsto$ sys. à pile, spécification $\mapsto$ $\mathcal{P}$-automate, vérification: par Pre^*/Post^*/automate de Büchi (LTL)
- *Applications*: raisonnement sur piles, par ex. « bytecode »

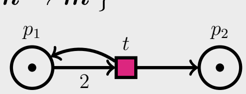


9. Systèmes infinis

Réseaux de Petri

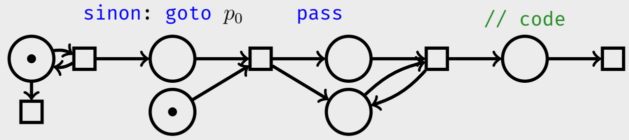
- *Déf.*: places et transitions reliées par arcs pondérés
- *Marquage*: nombre de jetons par place
- *Déclenchement*: si assez de jetons pour chaque arc entrant, les retirer, et en ajouter de nouveaux selon les arcs sortants
- *Successeurs*: $\mathrm{Post}^*(\boldsymbol{m}) = \{\boldsymbol{m}' \in \mathbb{N}^P : \boldsymbol{m} \xrightarrow{*} \boldsymbol{m}'\}$
- *Prédécesseurs*: $\mathrm{Pre}^*(\boldsymbol{m}') = \{\boldsymbol{m} \in \mathbb{N}^P : \boldsymbol{m} \xrightarrow{*} \boldsymbol{m}'\}$
- *Exemple*:





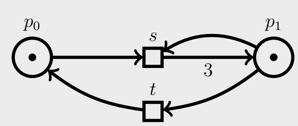
Modélisation

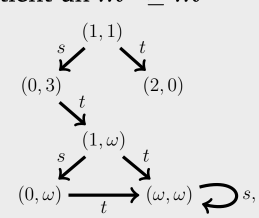
- *Processus*: comptés par les places
- *Exemple*: `si` $\neg x$: `x = vrai tant que` $\neg x$:
  `sinon: goto` p_0 `pass` `// code`



Graphes de couverture

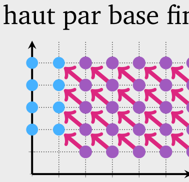
- *Idée*: construire $\mathrm{Post}^*(\boldsymbol{m})$ mais accélérer avec ω si $\boldsymbol{x} < \boldsymbol{x}'$
- *Test*: $\boldsymbol{m}'$ couvrable ssi le graphe contient un $\boldsymbol{m}'' \geq \boldsymbol{m}'$
- *Exemple*:





Algorithme arrière

- *Idée*: construire $\uparrow\mathrm{Pre}^*(\uparrow\boldsymbol{m}')$ en déclenchant vers l'arrière
- *Représentation*: d'ensemble clos par le haut par base finie
- *Test*: $\boldsymbol{m}$ peut couvrir $\boldsymbol{m}'$ ssi découvert



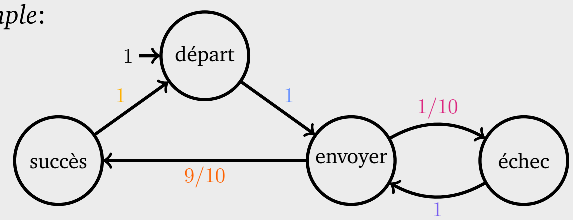
Accessibilité

- *Problème*: tester si $\boldsymbol{m}' \in \mathrm{Post}^*(\boldsymbol{m})$
- *Décidable* mais plus compliqué

10. Systèmes probabilistes

Chaîne de Markov

- *But*: remplacer non-déterminisme par probabilités
- *Déf.*: struct. de Kripke avec proba. sur transitions / états init.
- *Représentation*: probabilités = matrice **P** et vecteur **init**
- *Exemple*:



- *Événements*: exéc. inf. décrites par préfixes finis (cylindres)
- *Probabilité*: somme du produit des transitions de cylindres
- *Exemple*: $\mathbb{P}(\mathsf{F}\,\text{succès}) = \sum_{i=0}^{\infty} 1 \cdot ((1/10) \cdot 1)^i \cdot (9/10) \cdot 1 = 1$
- *Outils*: PRISM/Storm (PCTL, analyse quantitative, etc.)

Accessibilité

- *Accessibilité*: événement de la forme $A \mathbin{\mathsf{U}} B$
- *Partition*: $\underbrace{S_0 \coloneqq [\![\neg\exists(A \mathbin{\mathsf{U}} B)]\!]}_{\text{prob. nulle}}$, $\underbrace{S_1 \coloneqq B}_{\text{prob. } = 1}$, $\underbrace{S_? \coloneqq S \setminus (S_0 \cup S_1)}_{\text{prob. à déterminer}}$
- *Approche*: $\mathbf{A} \coloneqq \mathbf{P}$ sur $S_?$; $\boldsymbol{b}(s) \coloneqq$ proba. d'aller de s vers S_1; $\boldsymbol{x}(s) = \mathbb{P}(s \models A \mathbin{\mathsf{U}} B)$ est la solution de $(\mathbf{I} - \mathbf{A}) \cdot \boldsymbol{x} = \boldsymbol{b}$
- *Approx.*: $A \mathbin{\mathsf{U}}^{\leq n} B$ obtenu par $f^n(\mathbf{0})$ où $f(\boldsymbol{y}) \coloneqq \mathbf{A} \cdot \boldsymbol{y} + \boldsymbol{b}$

Comportements limites

- *CFC terminales*: une est atteinte et parcourue avec proba. 1
- FG *et* GF: se calculent via accessibilité et CFC terminales

CTL probabiliste (PCTL)

- *Syntaxe*: comme CTL, mais $\exists$ / $\forall$ deviennent $\mathcal{P}_I$, et ajout $\mathsf{U}^{\leq n}$
- $\mathcal{P}_I(\varphi)$: proba. de φ dans intervalle I?
- $\mathsf{U}^{\leq n}$: côté droit satisfait en $\leq n$ étapes?
- *Vérification*: calcul récursif + éval. proba. d'accessibilité