

IGL502/752 – Techniques de vérification et de validation
 Université de Sherbrooke

Examen final

Enseignant: Michael Blondin
 Date: jeudi 14 décembre 2023
 Durée: 3 heures

Directives:

- Vous devez répondre aux questions dans le **cahier de réponses**, et non sur ce questionnaire;
- **Une seule feuille** de notes au format 8 1/2" × 11" est permise;
- Les **fiches récapitulatives** se trouvent à la fin du questionnaire;
- **Aucun matériel additionnel** (notes de cours, fiches récapitulatives, etc.) n'est permis;
- **Aucun appareil électronique** (calculatrice, téléphone, montre intelligente, etc.) n'est permis;
- Vous devez donner **une seule réponse** par sous-question;
- L'examen comporte **6 questions** sur **3 pages** valant un total de **50 points**;
- La correction se base notamment sur la **clarté**, l'**exactitude** et la **concision** de vos réponses, ainsi que sur la **justification** pour les questions qui en requièrent une.

Question 1: logique temporelle linéaire (LTL)

Soit $AP := \{p, q, r\}$ et les formules LTL suivantes sur AP :

$$\varphi_1 := p \cup ((\neg q) \cup r)$$

$$\varphi_2 := (FGp) \vee (XXr)$$

$$\varphi_3 := G(p \vee Fr)$$

(a) Pour chaque formule φ_i , donnez un mot σ_i qui la satisfait et qui ne satisfait pas les deux autres, c.-à-d.

6 pts

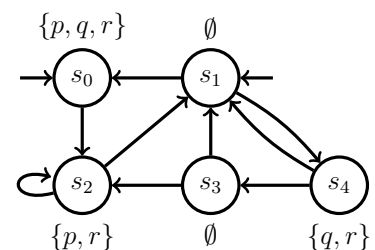
$$\begin{array}{lll} \sigma_1 \models \varphi_1 & \sigma_1 \not\models \varphi_2 & \sigma_1 \not\models \varphi_3, \\ \sigma_2 \not\models \varphi_1 & \sigma_2 \models \varphi_2 & \sigma_2 \not\models \varphi_3, \\ \sigma_3 \not\models \varphi_1 & \sigma_3 \not\models \varphi_2 & \sigma_3 \models \varphi_3. \end{array}$$

(b) Donnez un mot σ qui enfreint à la fois φ_1 , φ_2 et φ_3 , c.-à-d. tel que $\sigma \not\models \varphi_1$, $\sigma \not\models \varphi_2$ et $\sigma \not\models \varphi_3$.

2 pts

(c) Pour chaque i , dites si cette structure de Kripke \mathcal{T} satisfait φ_i . Justifiez.

3 pts



Question 2: automates de Büchi

(a) Donnez un automate de Büchi \mathcal{B} tel que $\mathcal{L}(\mathcal{B}) = \llbracket G(p \rightarrow X(\neg p \cup q)) \rrbracket$ sur alphabet $\Sigma := \{\emptyset, \{p\}, \{q\}, \{p, q\}\}$.

4 pts

(b) Rappelons que selon la construction vue en classe, les états de l'automate, résultant de l'intersection de deux automates de Büchi \mathcal{A} et \mathcal{B} , sont de la forme (p, q, I) où p est un état de \mathcal{A} , q est un état de \mathcal{B} , et $I \in \{\mathcal{A}, \mathcal{B}\}$. Il y a plusieurs façons de choisir l'ensemble des états acceptants. Lesquels de ces choix sont corrects?

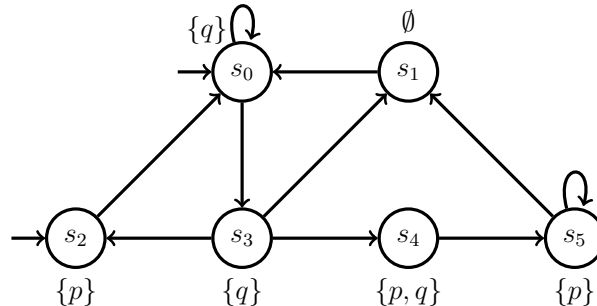
2 pts

- (i) les états où p est acceptant dans \mathcal{A} et $I = \mathcal{A}$;
- (ii) les états où p est acceptant dans \mathcal{A} et $I = \mathcal{B}$;
- (iii) les états où p est acceptant dans \mathcal{A} et $I = \mathcal{A}$, ainsi que ceux où q est acceptant dans \mathcal{B} et $I = \mathcal{B}$.

Question 3: logique temporelle arborescente (CTL) et vérification symbolique

Rappel: l'abréviation « BDD » réfère à « diagramme de décision binaire (ordonné et réduit) ».

Supposons que chaque état de la structure de Kripke \mathcal{T} ci-dessous soit codé par la représentation binaire de son indice: $s_0 = 000$, $s_1 = 001$, $s_2 = 010$, $s_3 = 011$, $s_4 = 100$ et $s_5 = 101$.



- (a) Pour chaque formule Φ ci-dessous, donnez l'ensemble $\llbracket \Phi \rrbracket$ des états de \mathcal{T} qui satisfont Φ . 6 pts
- (i) $\exists G(\neg p \vee q)$ (ii) $\exists(q \cup (p \wedge \neg q))$ (iii) $\forall X \exists G(\neg p \vee q)$
- (b) Construisez un BDD qui représente l'ensemble des états initiaux I . 2 pts
- (c) Construisez un BDD qui représente l'ensemble $\llbracket p \rrbracket$. 2 pts
- (d) Expliquez comment vérifier algorithmiquement si $\mathcal{T} \models p$ à partir des BDDs construits en (b) et (c). 2 pts

Question 4: systèmes à pile

Considérons ce programme constitué de deux fonctions et d'une variable booléenne globale x :

```

bool x ∈ {faux, vrai}

main():
  si x:
    foo(x)
  sinon:
    main()
m1:    foo(¬x)
m2:    assert(¬x)

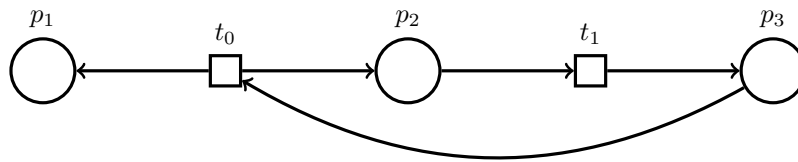
foo(bool y):
  x = ¬y

```

- (a) Modélisez le programme avec un système à pile \mathcal{P} . 2,5 pts
- (b) Donnez un \mathcal{P} -automate \mathcal{A} tel que $Conf(\mathcal{A})$ est l'ensemble des configurations où l'assertion est enfreinte. 1 pt
- (c) Construisez partiellement un \mathcal{P} -automate \mathcal{B} qui accepte $Pre^*(Conf(\mathcal{A}))$. Plus précisément, donnez au moins trois nouvelles transitions obtenues à partir de \mathcal{A} en exécutant l'algorithme de saturation vu en classe. Au moins deux de ces transitions doivent être obtenues *sans* utiliser une transition de \mathcal{P} étiquetée par une règle de la forme « lettre $\rightarrow \varepsilon$ ». 2,5 pts
- (d) Si vous aviez entièrement calculé le \mathcal{P} -automate \mathcal{B} en (c), comment auriez-vous pu déterminer si l'assertion du programme est toujours satisfaite? 1 pt

Question 5: systèmes à compteurs

Soit le réseau de Petri $\mathcal{N} = (P, T, F)$ suivant:



(a) Dessinez un graphe de couverture qui débute en $m := (1, 1, 1)$. 3 pts

(b) Dites lesquels de ces marquages peuvent être couverts à partir de $m := (1, 1, 1)$. Justifiez brièvement. 1,5 pts

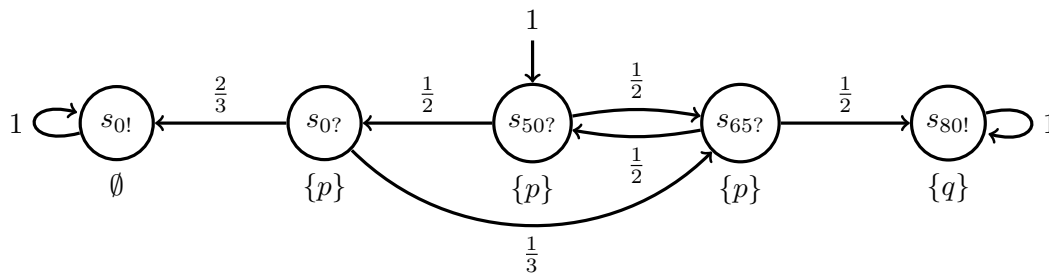
$$m_0 := (1, 1, 2), \quad m_1 := (0, 3, 2), \quad m_2 := (4, 2, 0).$$

(c) L'ensemble des marquages qui peuvent couvrir $m' := (1, 0, 0)$ est $\uparrow\{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$. Pourquoi? 2,5 pts

Question 6: systèmes probabilistes

Considérons un casino où l'on peut entrer avec 50\$ en poche. On est forcé d'y jouer à répétition jusqu'à gagner le montant maximal ou tout perdre. Lorsqu'on perd, le casino nous rend parfois un montant d'argent et nous force à rejouer.

Le casino est modélisé par la chaîne de Markov \mathcal{M} ci-dessous. Un état de la forme $s_{x?}$ indique qu'on détient x \$, mais qu'on ne sait pas si c'est notre montant final. Un état de la forme $s_{x!}$ indique qu'on détient x \$ et qu'on doit quitter le casino. Les propositions atomiques p et q indiquent respectivement que nous sommes en train de jouer et que nous avons gagné.



Remarque: il n'est pas obligatoire d'appliquer des algorithmes pour répondre aux questions.

(a) Donnez $\mathbb{P}(s_{50?} \models G p)$, c.-à-d. la probabilité de ne jamais quitter le casino. Justifiez. 2 pts

(b) Donnez $\mathbb{P}(s_{50?} \models G^{\leq 3} p)$, c.-à-d. la probabilité d'être encore dans le casino après trois joutes. Justifiez. 2 pts

(c) Le propriétaire du casino prétend que son établissement est « juste » car \mathcal{M} satisfait $\mathcal{P}_{=\frac{1}{2}}(F q)$. Montrez que \mathcal{M} satisfait en effet cette propriété PCTL. (le gain espéré du casino est donc de 10\$, ce qui n'est pas si juste!) 3 pts

FICHES RÉCAPITULATIVES

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: $s_1 \rightarrow s_2 \rightarrow s_1 \rightarrow s_2 \rightarrow \dots$
- ▶ 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 $L: S \rightarrow 2^{AP}$
- ▶ Exemple: si $AP = \{p, q, r\}$ et $L(s_1) = \{p, q\}$, alors s_1 satisfait p et q , mais pas r

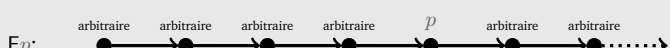
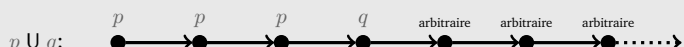
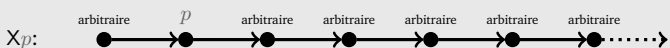
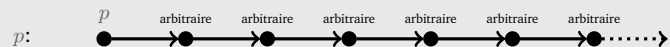
Explosion combinatoire

- ▶ 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 | p | $\varphi \wedge \varphi$ | $\varphi \vee \varphi$ | $\neg \varphi$ | $X\varphi$ | $\varphi U \varphi$ | $F\varphi$ | $G\varphi$
- ▶ 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 : $\{F, G, FG, GF\}$

Types de propriétés

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

Vérification

- ▶ Trace: états d'une exécution infinie \mapsto leurs étiquettes
- ▶ Satisfaisabilité: $\mathcal{T} \models \varphi \iff \text{Traces}(\mathcal{T}) \subseteq \llbracket \varphi \rrbracket$
- ▶ 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 $ababab \dots$

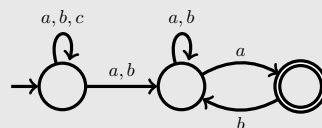
$b^*(aa^*bb^*)^\omega$: mots avec une infinité de a et de b

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

Automates de Büchi

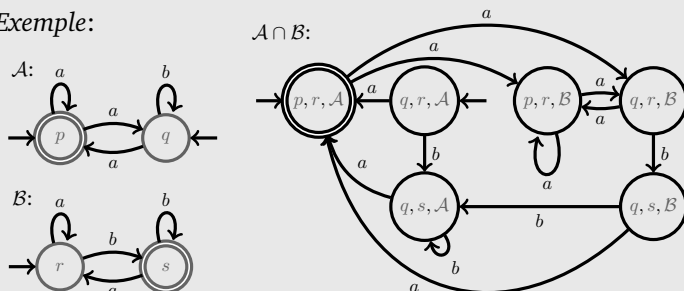
- ▶ Définition: automates usuels; plusieurs états initiaux
- ▶ Langage: mots qui visitent états acceptants ∞ souvent
- ▶ Expressivité: \equiv expressions ω -rég.; déterminisme \neq non dét.

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



Intersection d'automates de Büchi

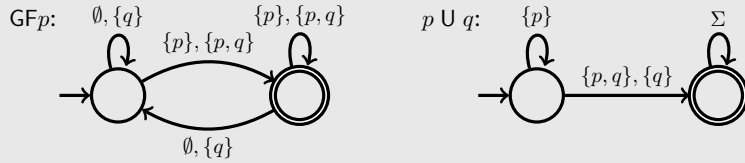
- ▶ 1^{ère} idée: simuler \mathcal{A} et \mathcal{B} en parallèle via produit; pas suffisant
- ▶ Solution: faire deux copies, alterner aux états acceptants
- ▶ Exemple:



4. Vérification algorithmique de formules LTL

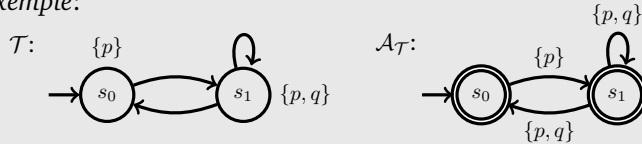
LTL vers automates

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



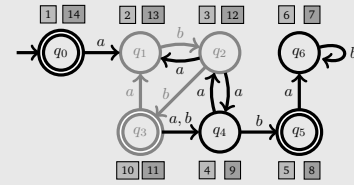
Structures de Kripke vers automates

- **Conversion:** étiquettes \equiv lettres + tout acceptant
- **Exemple:**

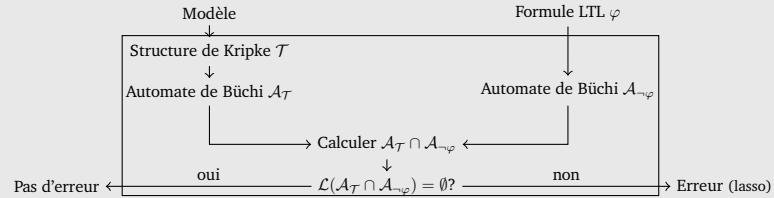


Test du vide

- **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)



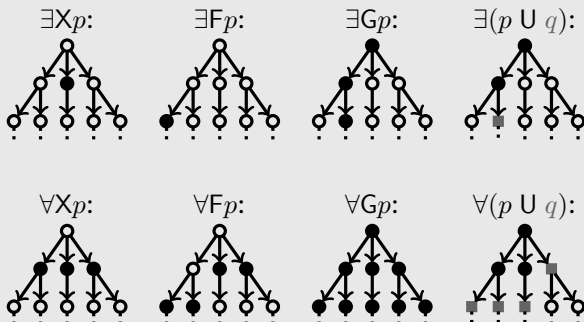
- **Mieux:** identifier les composantes fort. conn. (temps linéaire)
- **Sommaire:**



5. Logique temporelle arborescente (CTL)

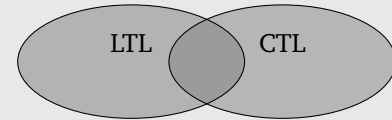
Logique

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



Propriétés d'un système

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



Équivalences

- **Distributivité:**
 $\exists F(\Phi_1 \vee \Phi_2) \equiv (\exists F\Phi_1) \vee (\exists F\Phi_2)$
 $\forall G(\Phi_1 \wedge \Phi_2) \equiv (\forall G\Phi_1) \wedge (\forall G\Phi_2)$
- **Attention:** pas équiv. si on change les quantificateurs
- **Dualité:** effet d'une négation: $\exists \leftrightarrow \forall, X \leftrightarrow X$ et $F \leftrightarrow G$
- **Idempotence:** $QTQT\Phi \equiv QT\Phi$ où $Q \in \{\exists, \forall\}, T \in \{F, G\}$

6. Vérification algorithmique de formules CTL

Algorithme

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

Logique propositionnelle

- **Règles récursives:**

$$\llbracket \text{vrai} \rrbracket = S,$$

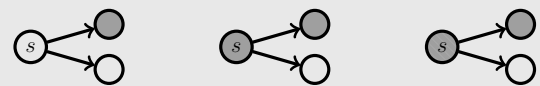
$$\llbracket p \rrbracket = \{s \in S : p \in L(s)\},$$

$$\llbracket \Phi_1 \wedge \Phi_2 \rrbracket = \llbracket \Phi_1 \rrbracket \cap \llbracket \Phi_2 \rrbracket,$$

$$\llbracket \neg\Phi \rrbracket = S \setminus \llbracket \Phi \rrbracket.$$

Opérateurs temporels existentiels

- **Calcul de $\llbracket \exists X\Phi \rrbracket$:** $\{s \in S : \text{Post}(s) \cap \llbracket \Phi \rrbracket \neq \emptyset\}$
- **Calcul de $\llbracket \exists G\Phi \rrbracket$:** $T \subseteq \llbracket \Phi \rrbracket$ max. t.q. $\forall s \in T : \text{Post}(s) \cap T \neq \emptyset$
- **Calcul de $\exists(\Phi_1 \cup \Phi_2)$:** $T \supseteq \llbracket \Phi_2 \rrbracket$ min. t.q.
 $s \in \llbracket \Phi_1 \rrbracket \wedge \text{Post}(s) \cap T \neq \emptyset \implies s \in T$



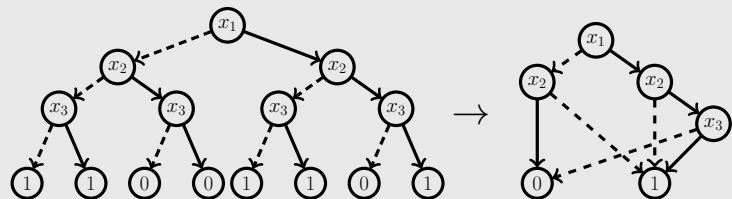
Optimisations

- **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

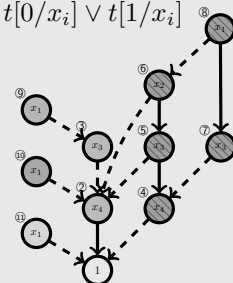


Manipulation

- ▶ **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 $build(\varphi)$
- ▶ **Op. bool.:** application récursive « synchronisée » avec $apply_{\circ}$
- ▶ **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:** $I \subseteq \llbracket \Phi \rrbracket \Leftrightarrow I \cap \llbracket \bar{\Phi} \rrbracket = \emptyset \Leftrightarrow apply_{\wedge}(u_I, u_{\llbracket \bar{\Phi} \rrbracket}) = 0$



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 $\{p \xrightarrow{a \rightarrow u} q, \dots\}$
- ▶ **But:** décrire un ensemble de piles (et non accepter des mots)
- ▶ **Configuration:** $\langle p, w \rangle \in P \times \Gamma^* \mapsto (p) + \text{pile}$

w_1
\vdots
w_n
- ▶ **Exemple de modélisation:**

bool x ∈ {faux, vrai}

foo():

x = ¬x;

si x: foo()

sinon: bar()

retourner

bar():

si x: foo()

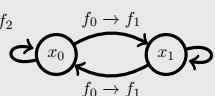
retourner

$f_1 \rightarrow b_0 f_2$

$f_2 \rightarrow \varepsilon$

$b_0 \rightarrow b_1$

$b_1 \rightarrow \varepsilon$



$f_1 \rightarrow f_0 f_2$

$f_2 \rightarrow \varepsilon$

$b_0 \rightarrow f_0 b_1$

$b_1 \rightarrow \varepsilon$

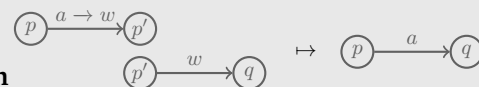
$f_0:$

$f_1:$

$f_2:$

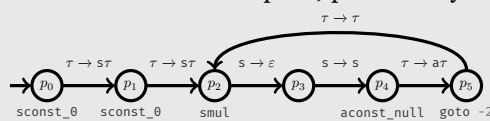
Calcul de prédécesseurs/successeurs

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



Vérification

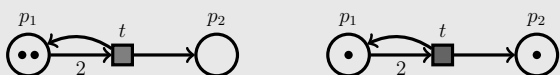
- ▶ **Approche:** système \mapsto sys. à pile, spécification \mapsto \mathcal{P} -automate, vérification: par $Pre^*/Post^*/$ automate de Büchi (LTL)
- ▶ **Applications:** raisonnement sur piles, par ex. « bytecode »



9. Systèmes infinis

Réseaux de Petri

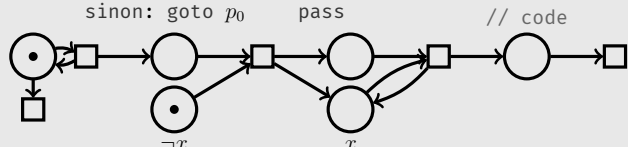
- ▶ **Déf.:** places et transitions reliées par arcs pondérés
- ▶ **Marquage:** nombre de jetons par place
- ▶ **Déclenchement:** si assez de jetons pour chaque arc entrant, les retirer, et en ajouter de nouveaux selon les arcs sortants
- ▶ **Successeurs:** $Post^*(m) = \{m' \in \mathbb{N}^P : m \xrightarrow{*}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



Modélisation

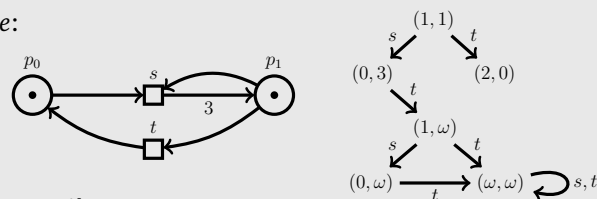
- ▶ **Processus:** comptés par les places

- ▶ **Exemple:** si $\neg x$: x = vrai tant que $\neg x$:
sinon: goto p0 pass



Graphes de couverture

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

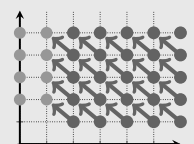


Algorithme arrière

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

Accessibilité

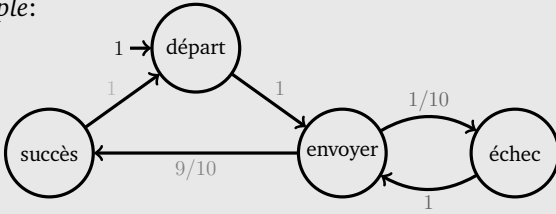
- ▶ **Problème:** tester si $m' \in Post^*(m)$
- ▶ **Décidable** mais plus compliqué



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 \mathbf{P} et vecteur **init**
- ▶ *Exemple*:



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

Accessibilité

- ▶ *Accessibilité*: événement de la forme $A \cup B$
- ▶ *Partition*: $S_0 := \llbracket \neg \exists (A \cup B) \rrbracket$ (prob. nulle), $S_1 := B$ (prob. = 1), $S_2 := S \setminus (S_0 \cup S_1)$ (prob. à déterminer)
- ▶ *Approche*: $\mathbf{A} := \mathbf{P}$ sur S_2 ; $\mathbf{b}(s) :=$ proba. d'aller de s vers S_1 ; $\mathbf{x}(s) = \mathbb{P}(s \models A \cup B)$ est la solution de $(\mathbf{I} - \mathbf{A}) \cdot \mathbf{x} = \mathbf{b}$
- ▶ *Approx.*: $A \cup^{\leq n} B$ obtenu par $f^n(\mathbf{0})$ où $f(\mathbf{y}) := \mathbf{A} \cdot \mathbf{y} + \mathbf{b}$

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 $U^{\leq n}$
- ▶ $\mathcal{P}_I(\varphi)$: proba. de φ dans intervalle I ?
- ▶ $U^{\leq n}$: côté droit satisfait en $\leq n$ étapes?
- ▶ *Vérification*: calcul récursif + éval. proba. d'accessibilité