Méthodes Formelles pour la Conception de Logiciels, Cours M2 Mops

Partie de S. Cerrito

S. Cerrito

2012-2013

Table des matières

URL avec le support de ce cours :

`www.ibisc.univ-evry.fr/~serena}`

(suivre le lien Current Teaching).

0.1 Problèmatique

Vérification (formelle) de propriétés de systèmes informatiques

S : programme ou système dont l'exécution ne termine pas (par ex. système d'exploitation). Deux approches possibles à la vérification de S :

1. Approche fondée sur la déduction : S est décrit (spécifié) par une formule $\mathcal{S}$ d'une logique, les propriétés souhaitées pour S sont exprimées par une formule P de cette logique. On cherche une déduction de $\mathcal{S} \to P$.
2. Approche fondée sur les modèles (*model checking*) : une modélisation de S (un système de transitions) est vue comme une interprétation $\mathcal{I}_S$ d'une logique, les propriétés sont exprimées par une formule P de cette logique. On teste si P est vraie pour $\mathcal{I}_S$.

Logiques illustréess dans ce cours : surtout la **Linear Temporal Logic** (**LTL**), mais on étudiera aussi des notions de la **Alternating Temporal Logic** (**ATL**).

Le cours est centré surtout sur la première approche, mais donne aussi des bases pour comprendre les principes du logiciel SPIN qui fait du model checking avec LTL.

SPIN a été développé à partir de 1980 aux Bell Labs (organisme de recherche actuellement de Alcatel-Lucent et, avant, de : American Telephone & Telegraph Company (AT&T)). C'est un logiciel libre depuis 1991.

0.2 La logique LTL

Linear Temporal Logic : Linear Temporal Logic (Logique Temporelle Linéaire).

Un élément de la famille des logiques dites modales, où une interprétation est un graphe d'interprétations classiques. Selon les hypothèses que l'on fait sur la structure de ce graphe, on obtient une logique modale différente.

Dans le cas de LTL, le graphe est linéaire.

0.2.1 LTL : Syntaxe et Sémantique

Principes :

- Fondée sur une modélisation discrète et linéaire du temps.
- Conçue par Z. Manna et A. Pnueli (1983).
- Ajoute aux connecteurs booléens $\neg, \vee, \wedge$ les opérateurs $\Diamond$, $\Box$, $\mathcal{U}$, $\bigcirc$.
- Ajoute une composante "dynamique" (changement d'état) à la logique classique.
- Modélise des programmes/systèmes qui ne terminent pas (les systèmes d'exploitation, par ex.).

Syntaxe de LTL

– P : n'importe quelle variable propositionnelle (booléenne).
Déf. des formules de LTL (F) :

$$F = True \mid P \mid \neg F \mid F \wedge F \mid F \vee F \mid \Box F \mid \Diamond F \mid F \mathcal{U} F \mid \bigcirc F$$

– $F_1 \rightarrow F_2 =_{DEF} \neg F_1 \vee F_2$.
– Un atome est soit $True$ soit une variable prop. Un littéral est soit un atome soit une expression $\neg A$ où A est un atome.
– Une lecture intuitive :
$\Box F$: Maintenant et toujours dans le future F
$\Diamond F$: Soit maintenant soit à au moins un instant du future F
$F_1 \mathcal{U} F_2$: F_1 *until* F_2
$\bigcirc F$: F est vraie à l'état (ou "instant") suivant.

Sémantique Formelle de LTL

Besoin d'étendre les notions d'*interprétation*, *vérité* et *modèle* de la logique (booléenne) classique.

Idée : une interprétation de LTL est une suite infinie à droite (mais "finie à gauche") d'interprétations de la logique booléenne classique.

Les formules sont interprétés sur les éléments de $\mathbb{N} = \{0, 1, 2, ...\}$ ordonnés par $\leq$ (ou une structure isomorphe). Les éléments de $\mathbb{N}$ sont dits *états* et 0 est dit *état initial*.

Sémantique Formelle de LTL, suite

- Une interprétation d'une formule de LTL dont l'ensemble des variables propositionnelles est $Prop$: une fonction $\mathcal{I} : \mathbb{N} \rightarrow 2^{Prop}$
  ($p \in \mathcal{I}(i)$: p est vraie à l'état i).
  Modélise **une** exécution d'un programme qui démarre à l'état 0.
  Exemple simple au tableau.
- $F=$ formule quelconque de LTL. Notation générale pour F est vraie à l'état i par rapport à l'interprétation $\mathcal{I}$:

$$\mathcal{I}, i \models F$$

Définition formelle de $\models$

$\mathcal{I}$ = interprétation, i = etat.

- Si At est un atome, $\mathcal{I}, i \models At$ ssi soit At est $True$ soit $At \in \mathcal{I}(i)$.
- $\mathcal{I}, i \models \neg F_1$ ssi $\mathcal{I}, i \not\models F_1$.
- $\mathcal{I}, i \models F_1 \wedge F_2$ ssi $\mathcal{I}, i \models F_1$ et $\mathcal{I}, i \models F_2$.
- $\mathcal{I}, i \models F_1 \vee F_2$ ssi $\mathcal{I}, i \models F_1$ ou $\mathcal{I}, i \models F_2$ (ou non-exclusif).
- $\mathcal{I}, i \models \Box F_1$ ssi quelque soit j tel que $i \leq j$ on a $\mathcal{I}, j \models F_1$.
- $\mathcal{I}, i \models \Diamond F_1$ ssi $\exists\, j$ tel que $i \leq j$ et $\mathcal{I}, j \models F_1$.
- $\mathcal{I}, i \models \bigcirc F_1$ ssi $\mathcal{I}, i+1 \models F_1$.
- $\mathcal{I}, i \models F_1 \mathcal{U} F_2$ ssi $\exists\, j$ tel que : $i \leq j$, et $\mathcal{I}, j \models F_2$ et, quelque soit n in $i, ..., j-1$, on a $\mathcal{I}, n \models F_1$.

Dessin au tableau pour $\mathcal{I}, i \models F_1 \mathcal{U} F_2$

F est vraie par rapport à une interprétation $\mathcal{I}$ ssi $\mathcal{I}, 0 \models F$. On dit aussi que $\mathcal{I}$ est un modèle de F.

F est valide si toute interprétation de F est un modèle de F ; elle est satisfiable s'il existe au moins une $\mathcal{I}$ qui est un modèle de F.
Un ensemble de formules $\{F_1, ..., F_n\}$ est satisfiable si la formule $F_1 \wedge \cdots \wedge F_n$ l'est.

F_2 est une conséquence logique de F_1 ssi, qque soit l'interprétation $\mathcal{I}$ de F_1 et quelque soit i, si $\mathcal{I}, i \models F_1$ alors $\mathcal{I}, i \models F_2$. Notation : $F_1 \models F_2$.

F_1 est logiquement équivalente à F_2, ce que l'on note $F_1 \equiv F_2$, ssi $F_1 \models F_2$ et $F_2 \models F1$.

Exemples d'évaluation de formules de LTL sur des interprétations : au tableau.

Un exemples de propriétés de systèmes exprimées en LTL

Exemple d'un feu qui règle la circulation.

Variables booléennes : vert, rouge, jaune (abrégées en : v, r et j)
Ordre du changement de couleur : $v \rightsquigarrow j \rightsquigarrow r \rightsquigarrow v$
Hyp. Le feu continue toujours à travailler.

1. A tout moment, le feu a exactement une des 3 couleurs :

$$\Box(\neg(v \wedge j) \wedge \neg(r \wedge j) \wedge \neg(r \wedge v) \wedge (v \vee j \vee r)))$$

2. Si le feu est dans un état où la couleur est verte, cette couleur persiste jusqu'à quand on passe à jaune :

$$\Box(v \rightarrow (v \mathcal{U} j))$$

3. Description générale de l'ordre du changement de couleurs du feu :

$$\Box((v \mathcal{U} j) \vee (j \mathcal{U} r) \vee (r \mathcal{U} v))$$

0.2.2 Déduction automatique pour LTL : méthode des "Tableaux"

Utilité ?

- $P1, P2$: formules de LTL qui expriment des propriétés d'un système S.
  Question Q : Si $P1$ est vraie pour S, est-il vrai que $P2$ est forcement vraie ?
- Une reformulation Q' de Q : $P1 \rightarrow P2$ est valide ?
- Déduction automatique : possibilité d'utiliser un *programme* qui donne la réponse à Q'.

Préliminaires sur les *tableaux* (peu importe la logique)

- Des systèmes de preuve bien adaptés à la déduction automatique.
- Systèmes de réfutation.
- **Intuition** : Pour prouver F, on essaie systématiquement de construire un modèle de $\neg F$. Si on échoue alors F est valide, sinon F n'est pas valide, car on a trouvé une interprétation (situation) contre-exemple.
- En effet, un calcul par tableaux répond directement à la question : la formule F (l'ensemble de formules E) est satisfiable ? Mais, puisque F est valide ssi $\neg F$ est insatisfiable, il peut répondre aussi à la question : F est valide ?

Preliminaires, suite

Une formule est dite en forme normale de négation (f.n.n.) si toute occurrence de $\neg$ s'applique à une formule atomique.

Toute formule booléenne classique F peut se réécrire en un formule F' telle que $F \equiv F'$ et F' est en f.n.n. En fait, pour toute formule A et B :

$\neg\neg A \equiv A$	
$\neg(A \wedge B) \equiv (\neg A) \vee (\neg B)$	$\neg(A \vee B) \equiv (\neg A) \wedge (\neg B)$

Tableaux de P. Wolper (1985) pour LTL

Ici : formules en f.n.n. Pour récrire toute formule de LTL en f.n.n. on exploite :

$\neg\Diamond A \equiv \Box\neg A$	$\neg\Box A \equiv \Diamond\neg A$
$\neg\bigcirc A \equiv \bigcirc\neg A$	$\neg(A\mathcal{U}B) \equiv \Box\neg B \vee (\neg B\mathcal{U}(\neg A \wedge \neg B))$

Un tableau pour LTL = un graphe orienté avec une racine, dont les noeuds sont des ensembles de formules de LTL en f.n.n. et les fils d'un noeud sont engendrés en appliquant une règle d'expansion.

– **Règles d'expansion** :

$\frac{E,A\wedge B}{E,(A\wedge B)^*,A,B}(\alpha)$	$\frac{E,A\vee B}{E,(A\vee B)^*,A \mid E,(A\vee B)^*,B}(\beta)$
$\frac{E,\Diamond A}{E,(\Diamond A)^*,A \mid E,(\Diamond A)^*,\bigcirc\Diamond A}(\Diamond)$	$\frac{E,\Box A}{E,(\Box A)^*,A,\bigcirc\Box A}(\Box)$
$\frac{E,A\mathcal{U}B}{E,(A\mathcal{U}B)^*,B \mid E,(A\mathcal{U}B)^*,A,\bigcirc A\mathcal{U}B}(\mathcal{U})$	$\frac{L,\bigcirc A_1,\cdots\bigcirc A_n,B_1*,\cdots,B_k^*}{A_1,\cdots,A_n}(\bigcirc)$

Dans la règle pour $\bigcirc$, L = ensemble de littéraux. Si n=0, écrire juste $True$ dans l'expansion. * est un **marquage** d'une formule. Un noeud auquel on applique la règle $\bigcirc$ est dit état.

– Les règles d'expansion sont fondées sur les équivalences :

$$\Diamond A \equiv A \vee \bigcirc \Diamond A \qquad \Box A \equiv A \wedge \bigcirc \Box A \qquad A \mathcal{U} B \equiv B \vee (A \wedge \bigcirc (A \mathcal{U} B))$$

– Un noeud est dit clos (ou contradictoire) s'il contient la formule $\neg True$ ou bien une paire de formules p, $\neg p$ pour quelque variable booléenne p.

Comment on applique les règles ?

– On applique une règle à un noeud seulement si ce noeud n'est pas clos.
– Une formule F est traitée seulement si elle n'est pas marquée avec $*$.
– En développant F, on crée un nouveau noeud avec une expansion Ex de F seulement si un noeud avec même étiquette Ex n'existe pas déjà. (Deux étiquettes sont identiques si elles contiennent exactement les mêmes formules, avec les mêmes marquages.)
  Sinon, on crée un arc vers ce vieux noeud.
– Appliquer au noeud $True$ la règle $\bigcirc$ une seule fois !

Un tableau de racine $\{A_1, ..., A_n\}$ est fini car le nombre d'ensembles de sous-formules distincts pouvant etiquetter un noeud est fini (pourquoi ?).

Exemple 1

Construction d'un tableau de racine $\{\Box p \wedge \Diamond \neg p\}$.

(Dans la suite, on fera un abus de notation, et on écrira, par exemple, "un tableau de racine $\Box p \wedge \Diamond \neg p$).

Comment tester la satisfiabilité de l'ensemble de formules à la racine ?

Procédure d'élagage pour les tableaux pour LTL

Répeter les instructions suivantes (dans l'ordre) jusqu'a quand le tableau reste stable :

1. Effacer tout sommet clos (c'est forcement une feuille).
2. Effacer tout sommet dont on a effacé tous les fils.
3. Effacer tout sommet s ayant comme élément une formule existentielle (*eventuality*) - $\Diamond F_2$ ou $F_1 \mathcal{U} F_2$ - telle qu'il n'existe pas de chemin, dans le graphe, conduisant de s à un sommet s' qui contient F_2.
   N.B. Une *eventuality* peut être marquée par *.

Si le tableau résultat est $\emptyset$, alors la racine est insatisfiable. Sinon, les chemins qui restent peuvent donner des modèles de la racine

Exemple 1, Suite

Élagage du tableau de racine

$$\Box p \wedge \Diamond \neg p$$

Que peut-on conclure ?

Exemple 2

Construction et élagage d'un tableau de racine

$$\Box\Diamond p$$

Que peut-on conclure ?

Toute règle du calcul sauf ($\bigcirc$) est dite *statique*. La règle ($\bigcirc$) est dite *dynamique*.

Quel est le sens intuitif de cette terminologie ? Et pourquoi on appele “état” un noeud au quel on a appliqué la règle ($\bigcirc$) ?

Si on déplie le graphe (avant l'élagage) on obtient un arbre pouvant être infini, et t.q. :

- une branche qui sera éliminée par l'élagage : une branche qui ne décrit pas un modèle de la racine.

- une branche qui ne sera pas éliminée : une branche ouverte, qui pourrait décrire un modèle (ou plus) de la racine. Comment construire un tel modèle ?

1. Ne garder que les "états" (comme définis dans la diapositive avec les règles d'expansion). Ce seront les états de l'interprétation.
2. Pour chaque état m, déclarer vrais les littéraux l tels que $l \in m$.
   N.B : description partielle d'une inteprétation, car, pour une p donnée, il se peut que ni $p \in m$ ni $\neg p \in m$. Dans ce cas, m peut décrire 2 situations : celle où p est vrai et celle où p est fausse.

Suite des exemples

Dépliage et étude des modèles possibles pour les deux exemples de tableaux de racine, resp. $\Box p \wedge \Diamond \neg p$ et $\Box \Diamond p$.

Est-il vrai que l'on obtient toujours un modèle à partir de toute branche qui ne sera pas éliminée par la procédure d'élagage du tableau ? Si oui, pourquoi ? Sinon, pourquoi ?

En fait :

Soit E une *eventuality* de la forme $\Diamond F_2$ ou $F_1 \mathcal{U} F_2$. Les formules E, F_1 et F_2 peuvent être marquées.

$$f_E = \{noeud\ s\ du\ tableau \mid E \notin s\ ou\ F_2 \in s\}$$

Déf. Une branche satisfait E ssi elle contient des noeuds de f_E infiniment souvent.

Soit $\mathcal{B}$ une branche infinie qui ne sera pas éliminée. Une interprétation décrite (partiellement) par $\mathcal{B}$ est un modèle de la racine ssi $\mathcal{B}$ satisfait toute *eventuality* qui est une sous-formule de la racine.

Comment tester si une branche a cette propriété, et décrit donc un modèle ? Voir après : “Automates de Büchi et Tableaux’

La construction et l'élagage des tableaux donnent un algorithme de décision pour le problème de la satisfiabilité (resp. validité) d'une formule F de LTL :

Thm (Correction du système de Wolper par rapport à l'insatisfiabilité) :
Si un tableau de racine F est "clos", au sens que, après élagage, on reste avec le tableau vide, alors F est insatisfiable.

Thm (Complétude du système de Wolper par rapport à l'insatisfiabilité) :
Si F est insatisfiable alors tout tableau de racine A est "clos" (c.à.d. : après élagage, on reste avec le tableau vide).

Complexité en temps et en espace de cet algorithme qui teste la satisfiabilité de F (donc aussi la validité de $\neg F$) : exponentielle par rapport à $\mid F \mid$, où $\mid F \mid$ est la taille de F.

NB : La satisfiabilité de PTL est un problème $PSPACE$-complet.

0.2.3 Tableaux pour LTL et Automates de Büchi

- Les Automates finis : des machines à un nombre fini d'états. Ils lisent des mots finis.
- Les ω-automates : des automates avec un nombre fini d'états, mais capables de lire et accepter/refuser un mot infini. Utilisation : description des exécutions infinies d'un programme/système (si nécessaire : une transition "bidon" no_op).
- Pour tester si une formule F de LTL est satisfiable ($\neg F$ est valide) on peut associer à F un ω-automate $\mathcal{A}_F$ tq :
  mot accepté = description d'un modèle de F
  mot refusé = description d'une interprétation où F est fausse.
  F est satisfiable ssi le langage accepté par $\mathcal{A}_F$ n'est pas vide.

- Langages acceptés par les ω-automates : ω-réguliers.
- On peut les décrire par des expressions contenant les symboles de l'alphabet Σ et les opérateurs :

  $+$ (union)

  $.$ (concaténation)

  Souvent, si L_1 et L_2 sont deux ensembles de mots, on note L_1L_2 à la place de L_1
- D'habitude les automates ont des étiquettes sur les transitions : $\delta(1, a) = 2$ ou, graphiquement, $1 \xrightarrow{a} 2$.
- Ici, étiquettes sur les états : $\overset{a}{1} \rightarrow 2$.

Automates de Büchi

– La classe la plus simple de ω-automates.

– Un automate de Büchi (BA) est :

$$\mathcal{A} = \langle Q, I, \delta, L, \Sigma, \mathcal{F} \rangle$$

où :

– Q = ensemble fini d'*états*

– $I \subseteq Q$ = ensemble des états *initiales*

– $\delta \subseteq Q \times Q$= *relation de transition*

– Σ : *alphabet* fini

– $L : Q \to \Sigma$: fonction qui associe un élément de Σ à chaque $q \in Q$.

– $\mathcal{F} \subseteq Q$ = états *d'acceptation*.

Exemple de BA

Automate $\mathcal{A}_1$ où :

$$\Sigma = \{\alpha, \beta\} \qquad Q = \{e_1, e_2\} \qquad I = \{e_1, e_2\}$$
$$L(e_1) = \alpha, L(e_2) = \beta \qquad \delta = \{\langle e_1, e_1\rangle, \langle e_1, e_2\rangle, \langle e_2, e_1\rangle, \langle e_2, e_2\rangle\} \qquad \mathcal{F} = \{e_1\}$$

(FIGURE au tableau)

Exécutions

– Soit $m \in \Sigma^\omega$ un mot infini sur l'alphabet Σ.

Exécution ρ d'un BA $\mathcal{A}$ sur m : un chemin infini dans $\mathcal{A}$ (graphe), dont la racine est un état initial et les noeuds sont étiquetés "en accord avec" m.

Formellement :

m est une fonction : $\mathbb{N} \to \Sigma$

$$m = m(0)\ m(1)\ m(2)...$$

Une exécution d'un BA $\mathcal{A}$ sur m est une fonction $\rho : \mathbb{N} \to Q$ tq :

1. $\rho(0) \in I$
2. Qque soit $i \geq 0$, $\langle \rho(i), \rho(i+1) \rangle \in \delta$
3. Qque soit $i \geq 0$, $L(\rho(i)) = m(i)$.

– S'il existe une exécution d'un BA $\mathcal{A}$ sur m on dit que $\mathcal{A}$ lit m.

Exemples d'éxecutions.

Automate $\mathcal{A}_1$ de l'exemple précédent.

- Mot $m_1 = (\alpha)^\omega$. Exécution ρ_1 lisant m_1 : $e_1\ e_1\ e_1$.... Exécution ρ_2 ne lisant pas m_1 : $e_1\ e_1$ et puis toujours e_2. L'exécution ρ_2 lit $m_2 = \alpha\alpha(\beta)^\omega$!
- Mot $m_3 = (\alpha\beta)^\omega$. Exécution ρ_3 lisant m_3 : $e_1\ e_2\ e_1\ e_2\ e_1\ e_2$...
- Etc.

Exécutions qui acceptent

$$\mathcal{A} = \langle Q, I, \delta, L, \Sigma, \mathcal{F} \rangle$$

- Une exécution ρ de $\mathcal{A}$ sur m accepte un mot m ssi $\exists$ au moins un état $e \in \mathcal{F}$ tq e apparaît infiniment souvent dans ρ.
- L'automate $\mathcal{A}$ accepte un mot m ssi $\exists$ au moins une éxécution ρ sur m qui accepte m.
- Le langage $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^\omega$ de $\mathcal{A}$ est l'ensemble des mots acceptés par $\mathcal{A}$.

Suite de l'exemple de l'automate $\mathcal{A}_1$:

$m_1 = (\alpha)^\omega$ accepté par $\mathcal{A}_1$, $m_2 = (\alpha\beta)^\omega$ accepté, $m_3 = \alpha\alpha(\beta)^\omega$ non accepté.
$\mathcal{L}(\mathcal{A}_1)$ caractérisé par l'expression ω-régulière $(\beta^*\alpha)^\omega$.

Automates déterministes et non-déterministes

- Un BA déterministe : fixé un mot m, $\exists$ au max. une éxécution ρ qui lit m.
- Un BA non-déterministe : fixé un mot m, on peut avoir plus d'une une éxécution ρ qui lit m, donc au moins deux états e_1 et e_2 tq :
  1. $L(e_1) = L(e_2)$
  2. soit $e_1, e_2 \in I$ soit $\exists\, e'$ tq $\langle e', e_1\rangle$, $\langle e', e_2\rangle \in \delta$.

NB : A la différence de ce qui se passe avec les automates finis, c'est faux que quelque soit l'automate non-déterministe $\mathcal{A}$ il existe un automate déterministe $\mathcal{A}'$ tel que $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$. Par exemple, le langage des mots sur l'alphabet $\{a, b\}$ contenant seulement un nombre fini de a est réconnu seulement par un automate non-déterministe.

Ici, on considère la classe d'automatimates la plus générale : les non-déterministes.

Exécution d'un BA et interprétations de LTL

Soit P un ensemble de formules atomique. Soit un automate $\mathcal{A}$ où l'alphabet Σ est l'ensemble des sous-ensembles de P :

$$\langle Q, I, \delta, L, 2^P, \mathcal{F} \rangle$$

- Mots lus : suites infinies de sous ensembles de P : $X_1, X_2, ...$ où $X_i \subseteq P$.
- Chaque X_i : une interprétation classique de la logique booléenne :
  Si $p \in X_i$ alors p est vraie dans l'interprétation classique correspondante.
  Sinon, p est fausse.
- Un mot m lu par $\mathcal{A}$: une interprétation $\mathcal{I}$ de LTL.

Exemple d'automate $\mathcal{A}_2$ dont les mots acceptées sont les modèles d'une formule de LTL : au tableau.

Représentation compacte de BA avec étiquettes en 2^P

- Dans la pratique : étiqueter chaque état avec exactement une interprétation (classique) n'est pas efficace.
- Si $e_1, \cdots, e_n$ ont les mêmes successeurs et les mêmes predecesseurs (et ils sont ou bien tous d'acceptation, ou bien aucun) $\rightsquigarrow$ "fusion" dans un seul état e, dont l'étiquette est une formule vraie exactement dans toutes les interprétations (classiques) étiquettes de $e_1, \cdots, e_n$.
- N.B : la formule est juste une représentation compacte d'une classe d'interprétations (classiques). La notion d'automate ne change pas ; c'est juste la notation qui est plus synthétique.

Exemple d'un BA $\mathcal{A}_3$ et de sa représentation compacte : au tableau.

Représentation compacte de BA avec étiquettes en 2^P, suite

- Il reste vrai qu'un mot lu est une suite infinie d'interprétation classiques = une interprétation $\mathcal{I}$ de LTL.
  Il reste vrai que : mot m = suite infinie d'interprétations classiques = fonction : $\mathbb{N} \to 2^P$

$$m = m(0)\ m(1)\ m(2)...$$

- Mais, grâce à la représentation compacte :
  Une exécution d'un BA $\mathcal{A}$ sur m est une fonction $\rho : \mathbb{N} \to Q$ tq :
  1. $\rho(0) \in I$
  2. Qque soit $i \geq 0$, $\langle \rho(i), \rho(i+1) \rangle \in \delta$
  3. Qque soit $i \geq 0$, $m(i) \models L(\rho_i)$. (ici $\models$, pas $=$!)
- **NB** : La notion d'éxécution n'a pas vraiment changé !
- **NB** : Si $L(\rho_i) = True$, alors, qque soit $m(i)$:

$$m(i) \models L(\rho_i)$$

Pour l'automate $\mathcal{A}_3$ déjà vu **et** pour sa représentation compacte.

– Exemple de mot accepté :

$$\{q\}\{p,q\}\{q\}\{p\}\{q\}\emptyset\{q\}(\{p\}\{q\})^\omega$$

– Exemple de mot non accepté :

$$\{q\}(\{p\})^\omega$$

– Toute interprétation $\mathcal{I}$ de LTL pour laquelle la formule

$$\neg p \wedge q \wedge \Box[((\neg p \wedge q) \rightarrow \bigcirc(p \vee \neg q)) \wedge ((p \vee \neg q) \rightarrow \bigcirc(\neg p \wedge q))]$$

est vraie est un mot accepté par $\mathcal{A}_3$ **et** par sa représentation compacte.

Remarque

Soit $P = \{q\}$ et supposons que, quand on passe d'un automate $\mathcal{A}$ à sa représentation compacte on fusionne dans un seul état e' 2 états e_1 et e_2 tels que $L(e_1) = \{q\}$ et $L(e_2) = \emptyset$. Alors $L(e') = q \vee \neg q$.

Mais $q \vee \neg q \equiv True$, donc on peut poser $L(e') = True$.

Une façon de voir : la formule associée à l'état e' "laisse passer" toute interprétation.

Automates généralisé de Büchi (GBA)

- Une généralisation des BA utile (par ex. dans la transformation : tableau de racine $B \Rightarrow$ Automate $\mathcal{A}_B$ dont le langage accepté est l'ensemble des modèles de B).
- Pas d'ajout de pouvoir expressif : tout GBA peut être traduit en un BA acceptant le même langage.

Automates généralisés de Büchi (GBA), Déf.

– $\mathcal{A} = \langle Q, I, \delta, L, \Sigma, \mathcal{F} \rangle$

Seul changements par rapports aux BA :

1. Dans un GBA, on peut avoir plusieurs ensembles d'états d'acceptation :

$$\mathcal{F} = \{E_1, \cdots, E_n\}$$

où, qque soit i, $E_i \subseteq Q$.

2. Notion d'acceptation : une éxécution ρ lisant m doit passer par chaque E_i infiniment souvent, c.à.d :
$inf(\rho) =_{def}$ ensemble des états de Q qui apparaîssent infiniment souvent dans ρ.
Une éxécution de ρ sur m accepte m ssi

$$inf(\rho) \cap E_i \neq \emptyset \text{ pour chaque } E_i \in \mathcal{F}$$

Transformer un tableau de racine F en un GBA $\mathcal{A}_F$

Soit F une formule de LTL. Un algorithme permettant de construire un GBA $\mathcal{A}_F$ tq $\mathcal{L}(\mathcal{A}_F)$= modèles de F utilise les tableaux.

1. On construit un tableau T_F de racine F.
2. On élague T_F en appliquant exclusivement :
   (a) Effacer tout sommet contenant tant p que $\neg p$ pour quelque variable propositionnelle p, ou contenant $\neg True$.
   (b) Effacer tout sommet dont on a effacé tous les fils.

   On reste avec T'_F.
3. Pour chaque eventuality E qui est une sous-formule de F, si $E = \Diamond A$ ou $B\mathcal{U}A$, on pose $f_E =_{DEF}$ ensemble de tous les noeuds m de T'_F tq. $E \notin m$ ou $A \in m$ (peu importe si E et B sont marquées ou pas).
   N.B. le noeud du tableau qui contient seulement la formule $True$ appartient à tout f_E !

Puis, $\mathcal{A}_F$ est défini par : $\rightsquigarrow$

- Ensemble d'états Q de $\mathcal{A}_F$= "états" de T'_F (selon la définition de la diapositive avec les règles d'expansion).
- $I = \{n \in Q \mid \exists$ un chemin en T'_F de la racine à n où n est le seul élément de $Q\}$.
- Relation $\delta = \{\langle n_i, n_j \rangle \mid n_i, n_j \in Q$ et $\exists$ un chemin en T'_F de n_i à n_j qui ne contient pas d'autre éléments de $Q\}$.

- Σ = ensembles de toutes les conjonctions possibles de littéraux du langage de F.
- Qque soit $n \in Q$, $L(n)$ = une formule : $l_1 \wedge \cdots \wedge l_p$ où $l_1, \cdots, l_p$ sont tous le littéraux $\in n$ dans le tableau T'_F. Cas particulier : $l_1 \wedge \cdots \wedge l_p = True$ si $p = 0$!
- Etats d'Acceptation :
  - Si la racine n'a pas de sous-formules qui soient des *eventualities*, alors $\mathcal{F} = \{Q\}$;
  - Sinon : $\mathcal{F} = \{f_{E_1} \cap Q, \cdots, f_{E_k} \cap Q\}$ où $E_1, \cdots, E_k$ sont toutes les *eventualities* sous-formules de F (la racine).

Exemple : Construction d'un tableau de racine

$$\Diamond p \wedge \Diamond \neg p$$

puis du GBA associé : au tableau.

Remarques Finales

La construction de l'automate $\mathcal{A}_F$ que l'on vient de décrire ne génére pas forcement l'automate le plus simple possible. Des techniques de simplification des GBA existent.

La méthodologie de SPIN : pour tester si un système S satisfait la propriété P :

1. On modèlise S par un automate $\mathcal{A}_S$;
2. On modèlise P par une formule F_P de LTL et on construit $\mathcal{A}_{\neg F_P}$;
3. On teste si $\mathcal{L}(\mathcal{A}_S \cap \mathcal{A}_{\neg F_P}) = \emptyset$.
   Si oui, S satisfait forcement P, sinon non. Dans le deuxième cas : tout mot de ce langage modèlise une éxécution sur S qui est un contre-exemple !