Spécifications et Vérifications Formelles

M1 Informatique UPSAY

Serenella Cerrito

Laboratoire IBISC et Département d'Informatique – Université d'Évry

second semestre 2016-2017

Plan du cours

1. Introduction : motivations
2. Automates pour des mots infinis.
3. Spécification de propriétes : Syntaxe et Sémantique de la Logique LTL
4. Cohérence d'une spécification et Décision de la satisfiabilité en LTL : Tableaux pour LTL
5. Automates de Büchi
6. Des tableaux aux automates
7. Vérification : *Model Checking* avec LTL
8. Logique CTL
9. Vérification : *Model Checking* avec CTL
10. Cohérence d'une spécification et Décision de la satisfiabilité en CTL

Informations sur le cours

- Diapositives de cours disponibles par e-media.
- Modalités de contrôle des connaissances : Un DS et un Examen.

$$NoteFinale = Max(Examen, \frac{2 \times DS + Examen}{3})$$

Introduction

Pourquoi utiliser des méthodes formelles (systèmes de transitions, automates, logiques,...) pour vérifier qu'un systéme informatique satisfait sa spécification, ?

- Développement : spécification formelle $\Rightarrow$ description non ambigüe partagée par des groupes differents ;
- Possibilité de vérifier automatiquement la coherence des spécifications des propriétés ;
- Possibilité de chercher systèmatiquement et automatiquement des exécutions contre-exemples à une spécification (*model checking*) ;
- Parfois, possibilité de conversion (semi-) automatique de propriétes specifiées en code ;
- Aide à la réalisation de logiciels et/ou systèmes fiables.

Introduction

Pour appliquer des techniques de vérification, il faut modéliser.

Besoin d'abstraction :

- focus sur les propriétes essentielles ;
- simplification ;
- indépendance de langages de programmation différents, de systèmes d'exploitations différents, etc. ;
- possibilité de tester certaines propriétés avant la réalisation finale

Systémes où l'interaction entre composantes ou avec l'environment est importante : systèmes *reactifs* (par ex : systèmes d'exploitation).
On suppose que les éxécutions ne terminent pas.

Plusieurs formalismes de modélisation et spécification sont possibles

Introduction

Propriétes souhaitées pour un formalisme de spécification :

- Possibilité de vérifier automatiquement qu'un système satifait la spécification. De façon efficace, si possible.
  On peut avoir besoin de vérifier :
  que la spécification est cohérente ;
  que l'implementation est correcte par rapport à la spécification.
- Expressivité. Mais compromis entre expressivité et complexité (il faut troquer).
- Possibilité de passage automatique de la spécification au prototypage.

Un formalisme unique, parfait, n'existe pas.

Dans ce cours : automates permettant de lire des éxécutions (mots) infinies pour modéliser des systèmes réactifs, logiques témporelles (LTL, CTL) pour spécifier leur propriétés

Modélisation de systèmes informatiques et Systèmes de Transitions

Une première modélisation d'un système : un graphe d'états, dirigé et ayant des racines, où chaque chemin modélise une exécution du système.

$+$ précisement : un *système de transitions avec étiquettes* est un triplet $\langle S, \hookrightarrow L \rangle$ où : S est un ensemble d'états, $\hookrightarrow$, la relation de transition, est un sous-ensemble de $S \times S$, et L est une fonction d'étiquette, qui associe aux arcs déterminés par T (et/ou les sommets) des expressions (ou des ensembles d'expressions) d'un langage formel.

Exemple de modélisation

Exemple de modélisation d'un ressort par un système de transitions avec étiquettes :

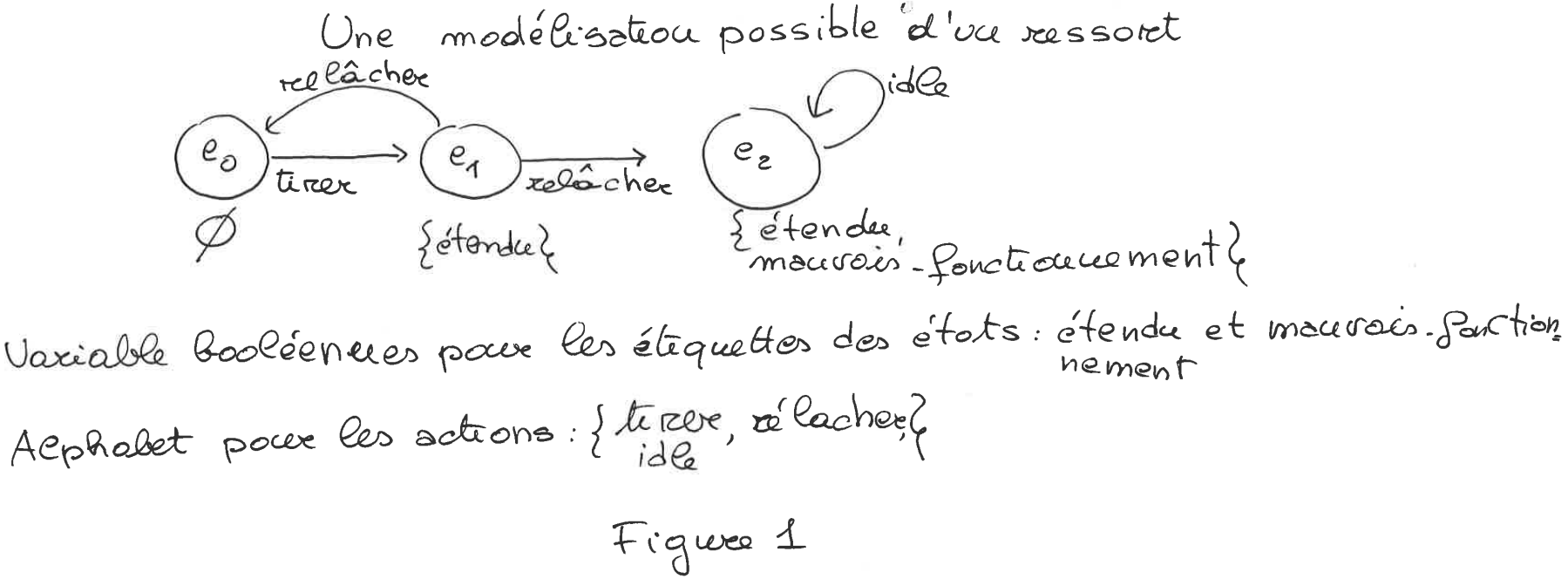


Figure 1

Modélisation de systèmes informatiques et Automates

On peut aussi voir la modélisation du ressort comme un automate, ayant un seul état initial, s_0.
NB : On a etiqueté les états **et** les arcs $\hookrightarrow$.

Un mot lu par cet automate code une exécution et il est infini : Ω-automates.

Conditions d'acceptation d'un mot infini ?
Plus sur les Ω-automates après.

Spécifier des propriètés d'un système informatique

Ici : expression les propriétés d'un système informatique. par des formules de la logique LTL :
Linear Temporal Logic (Logique Temporelle Linéaire).

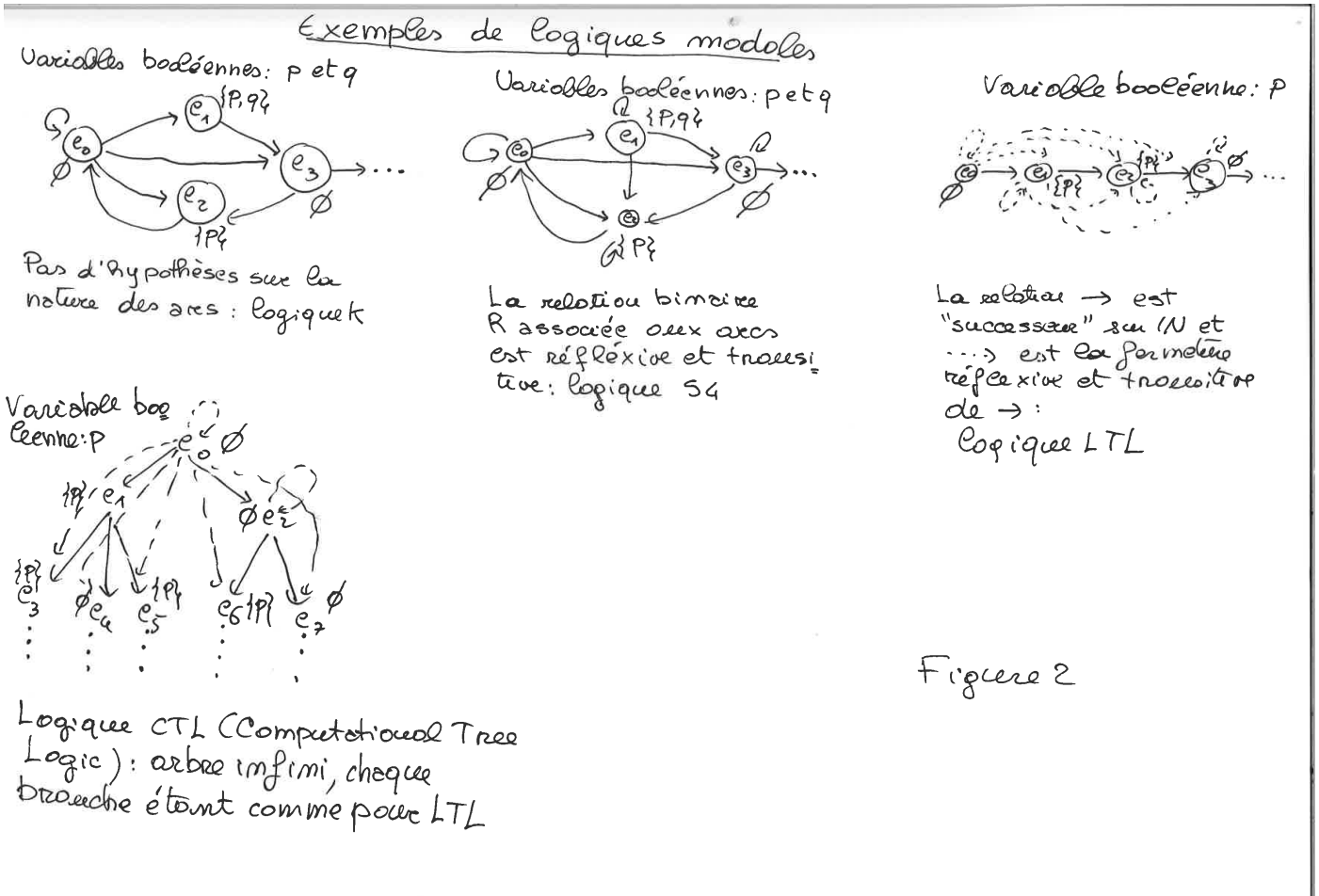
Conçue par Z. Manna et A. Pnueli (1983).

Principe de LTL : modélisation discrète et linéaire du temps.

Logiques Modales

LTL appartient à la famille des logiques dites *modales*, où une interprétation est un graphe d'interprétations classiques.

Hypothèses sur la structure des graphes $\Rightarrow$ logiques modales différentes.



Logique LTL

LTL : Lest états sont indexés par des éléments de $\mathbb{N}$ (les instants du temps). Le temps est discrète et linéaire (un instant a exactement un instant successeur).

LTL

Langage de LTL (propositionnelle) :
operateurs booléens $\neg, \vee$ et $\wedge$ + les opérateurs temporels :
$\bigcirc, \Box, \Diamond$ et U.

Logique classique + aspet dynamique

Modélise des programmes/systèmes reactifs et qui ne terminent pas (systèmes d'exploitation, par ex.).

Syntaxe de LTL

Soit P un ensemble de variables booléennes.

La grammaire des formules (F) de LTL est :
$F :=$
$p \mid True \mid \neg F \mid F \vee F \mid F \wedge F \mid \bigcirc F \mid \Box F \mid F \mathsf{U} F \mid \Diamond F$
où $p \in P$.

$F_1 \rightarrow F_2 =_{DEF} \neg F_1 \vee F_2$

Un *atome* est soit True soit un élément de P. Un *littéral* est soit un atome soit une expression $\neg A$ où A est un atome.

- $\bigcirc F$: F est vraie à l'instant suivant
- $\Box F$: F est vraie maintenant et toujours dans le future
- $\Diamond F$: F est vraie mainteinant ou bien à quelque instant du future
- $F_1 \mathsf{U} F_2$: F_1 until F_2

Syntaxe de LTL

Sous-formules

Si F est une formule de LTL, une sous-formule de F est un sous-mot du mot F qui est lui même une formule.
NB : F lui même est considéree sous-formule de F.

Calcul de $sousf(F)$, l'ensemble des sous-formules de F, recursivement :

- Si F est un atome, alors $sousf(F) = \{F\}$.
- Si $F = op\ F_1$ où $* \in \{\neg, \Box, \bigcirc, \Diamond\}$ alors $sousf(F) = \{F\} \cup \{F' \mid F' \in sousf(F_1)\}$
- Si $* \in \{\wedge, \vee, \mathsf{U}\}$ et $F = F_1 * F_2$ alors $sousf(F) = \{F\} \cup \{F' \mid F' \in sousf(F_1)\} \cup \{F'' \mid F'' \in sousf(F_2)\}$.

Vers la sémantique de LTL

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

Idée : une inteprétation de LTL est une suite infinie d'inteprétations de la logique booléennes qui peut être mise en bijection avec le nombres naturels : $s_0, s_1, s_2, ...$

Chaque interprétation classique s_i est un état. Un même état peut appparaître plusieurs fois dans la suite.

Sémantique formelle de LTL

Une interprétation d'une formule de LTL dont l'ensemble des variables propositionnelles est inclus dans P est une fonction $\mathcal{I} : \mathbb{N} \to 2^P$.

Intuition : c'est une suite d'états et, pour $i \in \mathbb{N}$, $p \in \mathcal{I}(i)$ signifie que p est vraie à l'instant i. On dit que $\mathcal{I}(i)$ est l'état s_i. NB : donc chaque état est une interprétation booléenne classique.

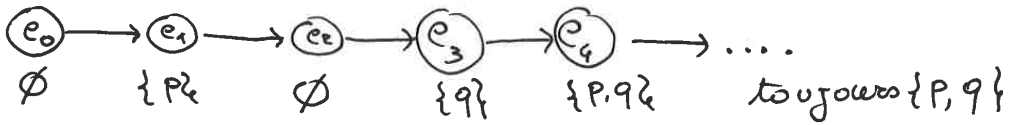
Si P est fini, forcement au moins un état se répete un nombre infini de fois : $s_i = s_{i+k_1} = s_{i+k_2} = \ldots$. Pourquoi ? Combien d'états différents peut contenir $\mathcal{I}$, si P a n éléments ?

Mais c'est faux qu'on peut réprésenter toute inteprétation par un graphe fini avec des cycles : $\rightsquigarrow$ Exemples : Figure 3.

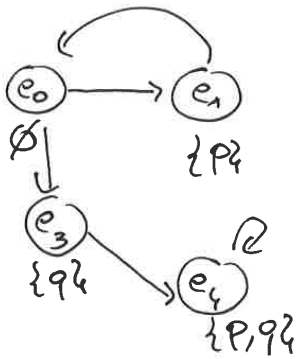
Exemples

Exemple 1

Variables booléennes : p et q



Si on représente par un graphe avec 4 états:



Ci dessus : à e_0 est vrai qu'à l'état suivant $p \wedge q$ est vraie
à e_2 " " " $\neg p \wedge q$ est vraie

A gauche : pas clair ce qui est vrai à e_0

Exemple 2 : variable booléenne p

$I(0)=\{p\}$, $I(1)=\emptyset$, $I(2)=\{p\}$, $I(3)=I(4)=\emptyset$, $I(5)=\{p\}$, $I(6)=I(7)=I(8)=\emptyset$, $I(9)=\{p\}$, ...
etc.
On augmente toujours de 1 le nombre de $\emptyset$ entre deux $\{p\}$.
C'est clair que c'est impossible de représenter I avec un graphe à 2 états !

Figure 3

Sémantique formelle de LTL

Une interprétation $\mathcal{I}$ modélise une exécution (infinie) d'un système, qui démarre à l'instant 0.

Si $\mathcal{I} = s_0, s_1, s_2, \ldots$ et $i \geq 0$, avec $\mathcal{I}_i$ on note le suffixe de $\mathcal{I}$ dont le premier état est s_i : $s_i, s_{i+1}, s_{i+2}, \ldots$.

Notation pour « la formule F est vraie à l'instant i par rapport à l'intéprétation $\mathcal{I}$ » : $\mathcal{I}_i \models F$.

La définition de $\mathcal{I}_i \models F$ est par récurrence sur la formule F :

Sémantique formelle de LTL

- Si At est un atome, $\mathcal{I}_i \models At$ ssi soit At est $True$ soit $At \in \mathcal{I}(i)$.
- $\mathcal{I}_i \models \neg F$ ssi $\mathcal{I}_i \not\models F$.
- $\mathcal{I}_i \models F_1 \wedge F_2$ 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 $\mathcal{I}_i \models F_2$.
- $\mathcal{I}_i \models \bigcirc F$ ssi $\mathcal{I}_{i+1} \models F$.
- $\mathcal{I}_i \models \Box F$ ssi quelque soit $j \geq i$ $\mathcal{I}_j \models F$.
- $\mathcal{I}_i \models \Diamond F$ ssi il existe $j \geq i$ tel que $\mathcal{I}_j \models F$.
- $\mathcal{I}_i \models F_1 \mathsf{U} F_2$ ssi il existe $j \geq i$ tel que $\mathcal{I}_j \models F_2$ et, quelque soit $n \in [i, j-1]$, $\mathcal{I}_n \models F_1$.

Figure 4 : signification de U

Deux cas pour $F_1 \mathrm{U} F_2$ vrai à l'état i :

1. $j > i$:

$$\underbrace{i \quad \ldots \quad j-1}_{F_1}, \underbrace{j}_{F_2} \ldots$$

2. $j = i$:

$$\underbrace{i}_{F_2} \ldots$$

Sémantique formelle de LTL

Une formule 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 dite satisfiable ssi il existe au moins un modèle de F. Un ensemble fini de formules $\{F_1, ..., F_n\}$ est satisfiable ssi $F_1 \wedge .... \wedge F_n$ l'est. On dit qu'une formule (ou un ensemble de formules) est insatisfiable quand ce n'est pas satisfiable.

F est dite valide ssi toute inteprétation (des variables booléennes de F) est un modèle de F.

F_2 est dite consequence logique de F_1 ssi, quelque soit l'interprétation $\mathcal{I}$ et quelque soit $i \in \mathbb{N}$, si $\mathcal{I}_i \models F_1$ alors $\mathcal{I}_i \models F_2$. On note : $F_1 \models F_2$.

F_1 et F_2 sont dites logiquement équivalentes ssi $F_1 \models F_2$ et $F_2 \models F_1$.

Exemples

$\Box(p \vee \neg p)$ et $\Box p \rightarrow \Diamond p$ sont valides.

La formule $\neg\Box(p \vee \neg p)$ est insatisfiable et l'ensemble de formules $\{\Box p, \Diamond\neg p\}$ aussi.

La formule $\Box p$ est satisfiable mais pas valide.

$\Box p \models \Diamond p$ mais $\Diamond p \not\models \Box p$: $\Box p \not\equiv \Diamond p$

$\bigcirc p \models \Diamond p$ mais $\Diamond p \not\models \bigcirc p$: $\bigcirc p \not\equiv \Diamond p$

$p\mathrm{U}q \models \Diamond q$ mais $\Diamond q \not\models p\mathrm{U}q$: $p\mathrm{U}q \not\equiv \Diamond q$

$T\mathrm{U}p \models \Diamond p$ et $\Diamond p \models T\mathrm{U}p$: $T\mathrm{U}p \equiv \Diamond p$

$\Diamond p \models \neg\Box\neg p$ et $\neg\Box\neg p \models \Diamond p$: $\Diamond p \equiv \neg\Box\neg p$

Exemple 1 : formalisation en LTL des propriétés d'un feu qui règle la circulation

Hypothèses : le feu travaille toujours ; l'ordre de changement de couleurs est : *vert* $\rightsquigarrow$ *jaune* $\rightsquigarrow$ *rouge* $\rightsquigarrow$ *vert*
Variables booléennes : v (vert), r (rouge), j (jaune).

- *A tout moment le feu a exactement une des trois couleurs*
  $\Box((v \vee j \vee r) \wedge \neg(v \wedge j) \wedge \neg(v \wedge r) \wedge \neg(r \wedge j))$
- *Si le feu est dans un état où la couleur est verte, cette couleur persiste jusqu'à quand on passe au jaune*
  $\Box(v \rightarrow (v \mathrm{U} j))$
- Expression de l'ordre de changement de couleurs
  $\Box((v \mathrm{U} j) \vee (j \mathrm{U} r) \vee (r \mathrm{U} v))$

Un des modèles possibles de la $\wedge$ des trois formules : la $\mathcal{I}$ telle que $\mathcal{I}(n) = \{v\}$ si $n = 0$ *mod* 3, $\mathcal{I}(n) = \{j\}$ si $n = 1$ *mod* 3 et $\mathcal{I}(n) = \{r\}$ si $n = 2$ *mod* 3

Exemple 2 : formalisation en LTL des propriétés du ressort

Hypothèses : le comportement du ressort est celui de la Figure 1.

Variables booléennes :*etendu*, *mauvais_fonctionnement*

- Interprétation LTL correspondante à l'éxécution où le ressort est d'abord tiré, puis laché, puis tiré, puis laché etc., et il fonctionne toujours :

  $\mathcal{I} : \mathcal{I}(n) = \emptyset$ si n est pair, $\mathcal{I}(n) = \{etendu\}$ si n est impair.

- Interprétation LTL correspondante à l'éxécution où le ressort est d'abord tiré, puis laché et il ne fonctionne plus :

  $\mathcal{I}' : \mathcal{I}'(0) = \emptyset$, $\mathcal{I}'(1) = \{etendu\}$,
  $\mathcal{I}'(n) = \{etendu, mauvais\_fonctionnement\}$ si $n > 1$.

Exemple 2 : formalisation en LTL des propriétés du ressort, suite

- Interprétation LTL correspondante à l'éxécution où le ressort est d'abord tiré, puis laché, puis tiré à nouveau, puis laché et il ne fonctionne plus : $\mathcal{I}'' : \mathcal{I}''(0) = \emptyset$, $\mathcal{I}''(1) = \{etendu\}$, $\mathcal{I}''(2) = \emptyset$, $\mathcal{I}''(3) = \{etendu\}$, $\mathcal{I}''(n) = \{etendu, mauvais\_fonctionnement\}$ si $n \geq 4$.

Exemple : le ressort, suite

On évalue des formules par rapport à $\mathcal{I}''$ (la dernière interprétation)

- $\mathcal{I}''_0 \models \neg etendu$
- $\mathcal{I}''_0 \models \bigcirc etendu$
- $\mathcal{I}''_0 \models \bigcirc \bigcirc etendu$
- $\mathcal{I}''_0 \models \Diamond \bigcirc etendu$
- $\mathcal{I}''_0 \models \neg \Box etendu$
- $\mathcal{I}''_0 \models \Diamond \Box etendu$
- $\mathcal{I}''_0 \models \neg((\neg etendu) \mathrm{U} mauvais\_fonctionnement)$
- $\mathcal{I}''_0 \models \Box((\neg etendu) \rightarrow \bigcirc etendu)$
- $\mathcal{I}''_0 \models \neg\Box(etendu \rightarrow \bigcirc \neg etendu)$
- $\mathcal{I}''_0 \models \Diamond(etendu \wedge \bigcirc etendu)$

Les deux dernières formules sont équivalentes ; pourquoi ?

Des équivalences importantes

1. $\neg \bigcirc F \equiv \bigcirc \neg F$
2. $\neg \Box F \equiv \Diamond \neg F$
3. $\neg \Diamond F \equiv \Box \neg F$
4. $\neg (F \mathrm{U} G) \equiv (\Box \neg G) \vee ((\neg G) \mathrm{U} (\neg F \wedge \neg G))$
5. $\Box F \equiv F \wedge \bigcirc \Box F$
6. $\Diamond F \equiv F \vee \bigcirc \Diamond F$
7. $F \mathrm{U} G \equiv G \vee (F \wedge \bigcirc (F \mathrm{U} G))$

Les trois dernières équivalences, dites *de point fixe*, donnent une caractérisation récursive des opérateurs $\Box$, $\Diamond$ et U.

Réécriture en forme normale de négation en LTL

Une formule est dite *en forme normale de négation* (fnn) si la négation s'applique exclusivement à des atomes. Par exemple, $\neg(p \vee \Box\neg q)$ n'est pas en fnn, mais $\neg p \wedge \Diamond q$ si.

Les équivalences booléennes de double négation et de De Morgan, avec les équivalences 1-4 permettent de réecrire toute formule F en une formule équivalente qui est en fnn.

Réécriture en forme normale de négation en LTL : Exemples

Réécriture en fnn de $\neg(p \vee \Box\neg q)$,
puis de $\neg((\bigcirc p)\mathrm{U}(q \wedge r))$.

- $\neg(p \vee \Box\neg q) \underbrace{\equiv}_{\textit{loi de De Morgan}} \neg p \wedge \neg(\Box\neg q) \equiv \neg p \wedge \Diamond q$
  (équivalence 3, et loi de double négation).
- $\neg((\bigcirc p)\ \mathrm{U}\ (q \wedge r))$
  $\underbrace{\equiv}_{\textit{par la} \equiv 4} \Box(\neg(q \wedge r)) \vee ((\neg(q \wedge r))\ \mathrm{U}\ [(\neg \bigcirc p) \wedge \neg(q \wedge r)])$
  $\underbrace{\equiv}_{\textit{loi de De Morgan et la } 1}$
  $\Box(\neg q \vee \neg r) \vee ((\neg q \vee \neg r)\ \mathrm{U}\ [(\bigcirc\neg p) \wedge (\neg q \wedge \neg r)])$

Réécriture en forme normale de négation en LTL

La réécriture en fnn est utile pour décider si une formule est satisfiable, et pour lui associer un Ω-automate : voir après.

Contraintes d'équité en LTL

Une contrainte d'équité (*fairness constraint*) pose des restrictions sur les exécutions admises.

Exemples d'expression de contraintes d'équités en LTL

Variables booléennes : $\mathit{enabled}_\alpha$ pour dire : « la transition α est autorisée » et $\mathit{executed}_\alpha$ pour dire : « la transition α est executée ».

- A partit d'un instant donné, la transition α est exécutée toujours : $\Diamond\Box \mathit{executed}_\alpha$.
- La transition α est exécutée infiniment souvent : $\Box\Diamond \mathit{executed}_\alpha$.
- Equité faible pour la transaction α : $(\Diamond\Box \mathit{enabled}_\alpha) \rightarrow \Diamond\Box \mathit{executed}_\alpha$
- Equité forte pour la transaction α : $(\Box\Diamond \mathit{enabled}_\alpha) \rightarrow \Diamond\Box \mathit{executed}_\alpha$

Equité forte et faible

NB : L'équité faible est une conséquence logique de la forte, mais la reciproque est fausse.
Preuve de : *L'équité faible est une conséquence logique de la forte*.

Soit $\mathcal{I}$ une interprétation quelconque, i un de ses états.
Supposons :
1) $\mathcal{I}_i \models (\Box\Diamond \mathit{enabled}_\alpha) \rightarrow \Diamond\Box \mathit{executed}_\alpha$ (équité forte).
Supposons l'antecedent de l'équité faible vrai à i :
2) $\mathcal{I}_i \models (\Diamond\Box \mathit{enabled}_\alpha)$.
En général, $\Diamond\Box F \models \Box\Diamond F$. Donc :
3) $\mathcal{I}_i \models \Box\Diamond \mathit{enabled}_\alpha$.
Par (1), nous obtenons :
4) $\mathcal{I}, s \models \Diamond\Box \mathit{executed}_\alpha$.
CQFD

Le problème de la cohérence d'une spécification LTL

On a décrit des propriétes sohuaitées pour un système avec une formule LTL. On a écrit une *spécification formelle* S.

Problème : la spécification S est cohérente ou pas ?

Autre formulation : la formule S est satisfiable ou pas ?

Dans la suite : une méthode constructive pour tester si S est satisfiable ou pas.
Si elle est, la méthode nous en fournit un modèle.

C'est la méthode des tableaux de P. Wolper (1985).

NB : Elle a été étendue, après, à des logiques temporelles plus complexes : avec plusieurs futures possibles, avec plusieurs agents qui peuvent former des coalitions, etc. Mais elle est la base.

Tableaux pour LTL

Les tableaux pour LTL ne sont pas des choses comme ça :

Ce sont des structures permettant de faire de la déduction automatique.

Préliminaires sur les méthodes de tableaux

Peu importe la logique, la philosophies des méthodes par tableaux qui testent la satisfiabilité d'une formule F (ou d'un ensemble de formules E) est la même :

- L'entrée est F (ou E) ;
- On essaye de construire un modèle de F (ou E) de façon systématique ;
- Si échec, alors F (ou E) n'est pas satisfiable. Sinon, le tableau donnera au moins un modèle de F (ou de E).

Préliminaires sur les méthodes de tableaux

Peu importe la logique :

- Les tableaux sont des systèmes de raisonnement adaptés à la déduction automatique ;
- Faits pour tester la satisfiabilité, mais il peuvent aussi prouver la validité d'une formule F, par réfutation :
  - On essaie de montrer que $\neg F$ est satisfiable, en essayant de façon systématique d'en construire un modèle ;
  - Si échec, alors F est valide. Sinon, elle n'est pas valide, car $\neg F$ peut être vraie.
  - On exploite le principe : F est valide ssi $\neg F$ est insatisfiable.

Préliminaires sur les méthodes de tableaux

Pour aider l'intuition, voyons comment les tableaux pour la logique booléenne marchent, en étudiant trois cas :

1. $((p \to q) \wedge p$ est satisfiable ? Si oui, en fournir un modèle (un'intérprétation booléenne classique où elle est vraie).
2. $((p \to q) \wedge \neg q) \wedge p$ est satisfiable ? Si oui, en fournir un modèle.
3. $((p \to q) \wedge \neg q) \to p$ est valide ?

Au tableau (l'autre tableau :-))

Tableaux pour LTL

Dans la formulation qu'on verra ici, ils utilisent des formules en fnn.

On a vu que toute formule peut être réécrite en fnn. On utilisera seulement des formules en fnn.

Convention pour la suite :
si on sait que E est un 'ensemble de formules, et F une formule, et on écrit : E, F alors on signifie l'ensemble de formules : $E \cup \{F\}$;

Tableaux pour LTL

Un tableau pour LTL qui teste la satisfiabilité d'un ensemble de formules E est un graphe orienté avec une racine tel que :

- Chaque sommet a un ensemble de formules (en fnn) comme étiquette ;
- L'étiquette de la racine est E (et d'habitude on omet les accolades) ;
- Les successeurs d'un sommet sont obtenus en lui appliquant une *règle d'expansion* (voir après) ;
- Formes générales d'une règle d'expansion :
  $\frac{E_1}{E_2}$ où E_1 etE_2 sont des ensembles de formules ; on crée un seul successeur ;
  $\frac{E_1}{E_2 | E_3}$ où E_1, E_2 et E_3 sont des ensembles de formules. On crée deux successeurs) ;

Tableaux pour LTL

Possibilité de cycles : si un règle d'expansion appliquée à un sommet s_i créerait un sommet ayant comme étiquette un ensemble X qui est déjà l'étiquette d'un sommet existant s_j, pas de création de nouveau sommet : s_j devient successeurs de s_i (*loop check*).

Si une règle d'expansion a la forme $\frac{E_1}{E_2}$, on dit que E_2 est l'*expansion* de E'.

Si une règle d'expansion a la forme $\frac{E_1}{E_2|E_3}$, on dit que E_2 et E_3 sont les deux *expansions* de E_1.

Intuition : Si un sommet s ayant E_1 comme étiquette a été créé, des successeurs de s dont les étiquettes sont les expansions de E_1 (un seul dans le premier cas, deux dans le deuxième) peuvent être crées (sauf le cas de *loop* déjà cité).

Règles d'expansion pour les tableaux LTL

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

Dans toute règle : $*$ est un marquage d'une formule.

Règle $(\bigcirc)$:

- L = ensemble de littéraux.
- Si $n = 0$, écrire juste *True* dans l'expansion.
- Un noeud auquel on applique $(\bigcirc)$ est dit état (du tableau).

Intuition sur les règles

Voyons la signification intuitive de l'application de règles à partir de la formule $\Box p \wedge \Diamond \neg p$, puis de la formule $p \mathrm{U} q$: au tableau (noir).

Sur les règles d'expansion

Les règles d'expansion ($\Diamond$), ($\Box$) et (U) sont fondéés sur les équivalences de point fixe (4, 5 et 6) déjà vues.

Un sommet est dit *clos* (ou *contradictoire*) s'il contient la formule $\neg \mathit{True}$ ou bien une paire $p, \neg p$ pour quelque variable booléenne p.
Intuition : il représente une situation (état) impossible.

Comment appliquer les règles d'expansion

- On aplique une règle à un sommet seulement s'il n'est pas clos ;
- Une formule F, dont l'opérateur principal est $op \in \{\wedge, \vee, \Diamond, \Box, \mathsf{U}, \bigcirc\}$, est traitée, en appliquant la règle (op), à un sommet qui la contient, seulement si elle n'est pas marquée avec * ;
- On crée un nouveau sommet s' successeur de s_i seulement si l'étiquette de s' dictée par la règle utilisée n'est pas l'étiquette d'un sommet s_j déjà existant ; sinon, s_j devient successeurs de s_i (*loop check*).
  NB : Deux étiquettes sont idéntiques si elles contiennent exactement
  les mêmes formules, avec les mêmes marquages.
- *Appliquer à un sommet marqué par* $\{True\}$ *la règle* $\bigcirc$ *exactement une fois !*

Sur les règles d'expansion

- NB. Appliquer (op) à un sommet marqué par $\{True\}$ crée une boucle.
- Un tableau de racine $\{A_1, ..., A_n\}$ est forcement fini. Pourquoi ?

Exemples de tableau

Construction d'un tableau de racine $\{\Box p \wedge \Diamond \neg p\}$,
puis d'un tableau de racine $\{p \mathrm{U} q,\ \neg q,\ \bigcirc \neg q\}$: voir les figures 4 et 5.

Dans la suite, on n'écrira jamais les accolades, et on écrira, par exemple, par abus de language : un tableau de racine $\Box p \wedge \Diamond \neg p$.

Tableau pour $\{\Box p \wedge \Diamond \neg p\}$

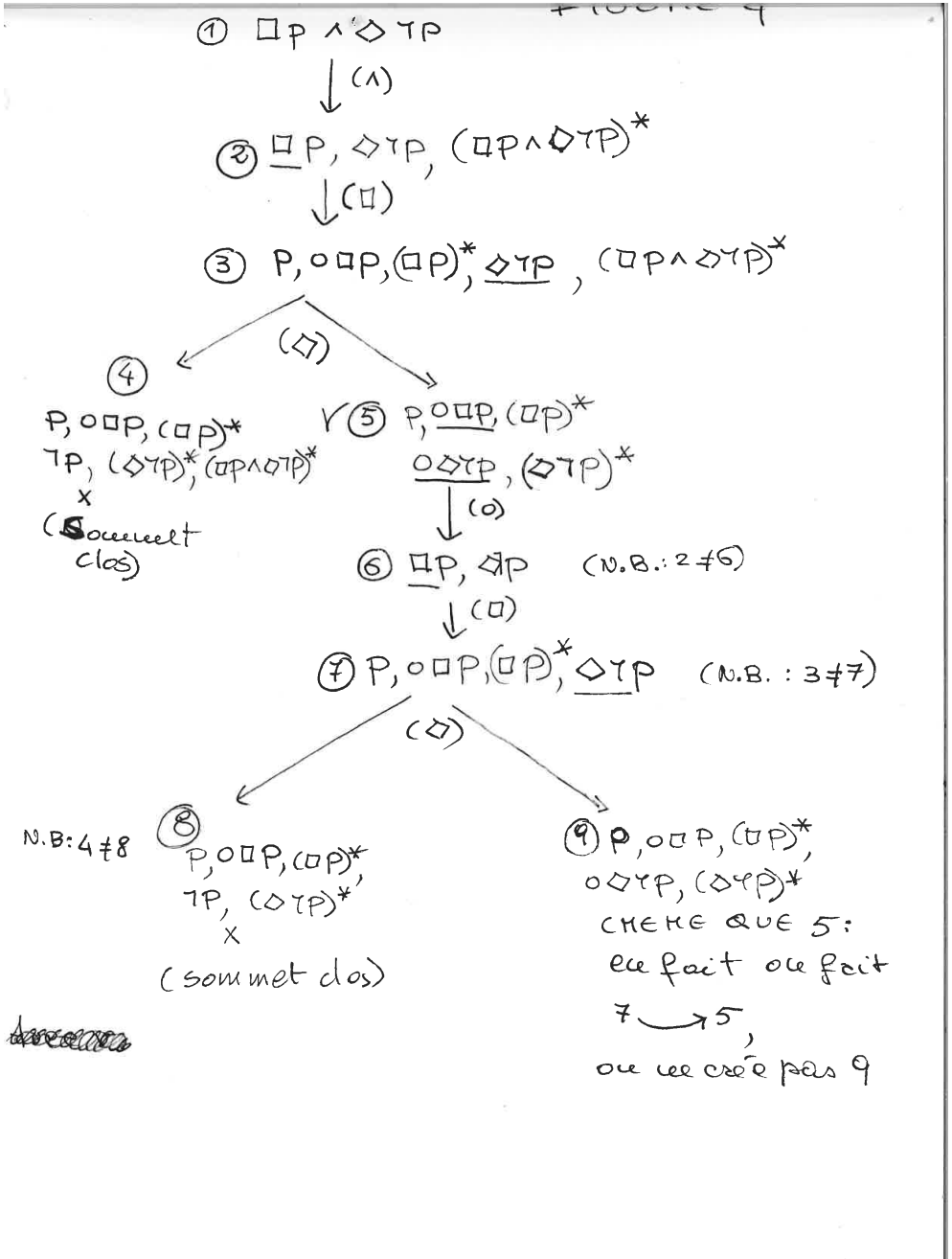
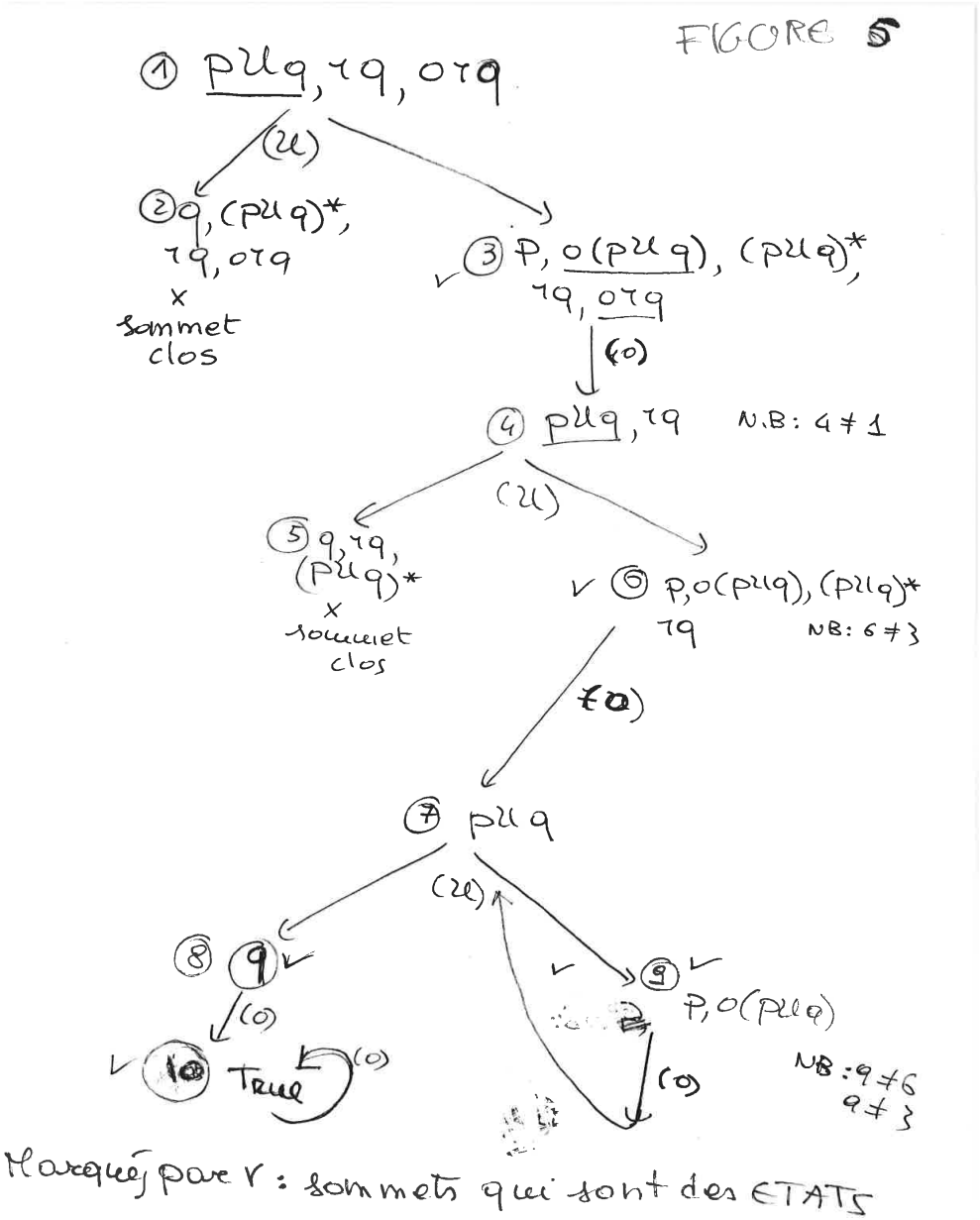


Tableau pour $\{pUq,\ \neg q,\ \bigcirc\neg q\}$



Formules dites *Eventualities*

Une formule de la forme $\Diamond F_2$ ou de la forme $F_1 \mathrm{U} F_2$ est dite *existentielle* ou *eventuality* : elle « promet » que F_2 sera vraie, tôt ou tard.

Formules dites *Eventualities*

Une formule de la forme $\Diamond F_2$ ou de la forme $F_1 \mathrm{U} F_2$ est dite *existentielle* ou *eventuality* : elle « promet » que F_2 sera vraie, tôt ou tard.

Elagage d'un tableau

Une fois construit un tableau, on procéde à l'élimination des mauvais sommets : on élague le graphe obtenu.

Répéter les instructions suivantes, dans l'ordre, jusqu'à quand le tableau reste stable :

1. Supprimer tout sommet clos (c'est forcement une feuille) ;
2. Supprimer tout sommet dont on a déjà supprimé tous les successeurs ;
3. Supprimer tout sommet s dont l'étiquette contient une formule existentielle (*eventuality*) – $\Diamond F_2$ ou $F_1 \mathrm{U} F_2$ –telle qu'il n'existe pas de chemin, dans le tableau, allant de s à un sommet ayant F_2 comme élément de son étiquette.
   NB : l'éventuality, et/ou F_2 pourront être marquées ou pas : ça ne change en rien l'istruction. Quand on dit « contient une formule » on fait abstraction du marquage de la formule.

Tableau ouvert ou clos et décision de la satisfiabilité

- Si le tableau qui reste après l'élagage est vide, alors on dit que le le tableau est clos et on déclare la racine insatisfiable.

  Ce qui s'est passé, si le tableau est clos : tout essai de construire un modèle de la racine a échoué.
- Si le tableau qui reste après l'élagage est n'est pas vide, on dit que le le tableau est ouvert et on déclare la racine satisfiable.

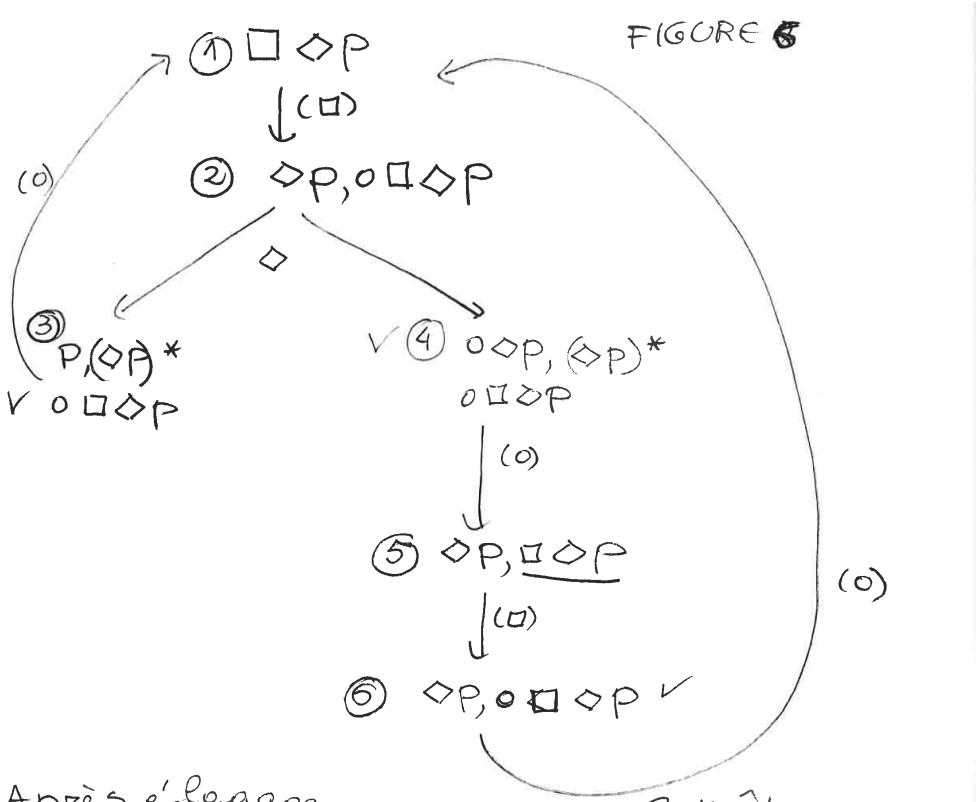
  En effet, certains des chémins qui restent dans le tableau élagué ouvert décrivent des modèles de la racine.

Exemples

Elagage du tableau de racine $\{\Box p \wedge \Diamond \neg p\}$ déjà construit.
Que peut-on conclure ? Que la formule racine est insatisfiable.

Construction puis élagage d'un tableau pour $\Box \Diamond p$.
Que peut-on conclure ? Voir la figure 6.

Tableau pour $\Box\Diamond p$.



Après enlogage on reste avec le même tableau, ici : la formule racine est donc déclarée satisfiable.

Pourquoi il reste le même? seule éventualité: $\Diamond p$, contenue dans tout sommet sauf 1. Et de chaque sommet part une chemin qui va à ③, qui contient p.

Etats d'un tableau

Pourquoi on appele « état » un sommet du tableau au quel on peut appliquer (◯) ? On a explicité tout ce qui est vrai à ce sommet, qui représente alors un état d'un modèle (candidat)

Toute règle d'expansion sauf (◯) est dite statique, tandis que (◯) est dite dynamique.
Pourquoi ? La règle (◯) crée un nouvel état dans le modèle.

Tableau ouvert et modèles de la racine

- Si on déplie un tableau avant de l'élaguer, on obtient un arbre de profondeur infinie ;
- Une branche qu'on supprime avec l'élagage : une branche qui ne décrit pas un modèle de la racine ;
- Une branche qui reste dans le tableau élagué est dite *branche ouverte* et peut décrire un modèle de la racine.

  Comment ?

Tableau ouvert et modèles de la racine

Comment construire un modèle de la racine à partir d'une branche ouverte $\mathcal{B}$ de l'arbre obtenu en dépliant un tableau ?

- Ne garder que les états du tableau de $\mathcal{B}$. On obtient une suite infinie $e_0, e_1, e_2, \ldots$ Chaque e_i sera un état d'une interprétation possible ;
- Pour chaque e_i, declarer vrais exactement les littéraux qui sont dans son étiquette ;
  Dit autrement, on associe à e_i l'interprétation booléenne $\{p \mid p \text{ appartient à l'étiquette de } e_i\}$.

  NB. Il se peut que, pour une variable booléenne p donnée, ni p ni $\neg p$ soient éléments de l'étiquette de e_i : on a obtenu une inteprétation partielle : à l'instant i, p peut être ou vraie où fausse. Dans ce cas, $\mathcal{B}$ décrit un ensemble de modèles de la racine.

Tableau ouvert et modèles de la racine

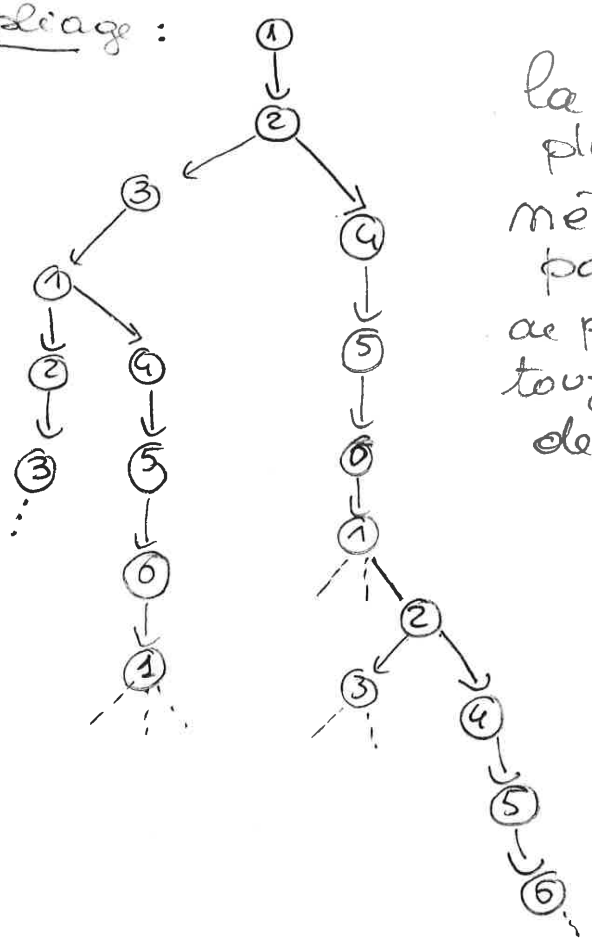
- Ce n'est pas vrai que toute branche ouverte décrit un modèle de la racine ! Etudier les branches d'un tableau pour $\Box\Diamond p$ (voir la Figure 7).
- En fait : soit Ev une formule existentielle (*eventuality*) de a forme $\Diamond F_2$ ou $F_1 \mathrm{U} F_2$.
  - Définition : $f_{Ev} = \{s$ est un sommet du tableau $\mid Ev \notin$ étiquette de s ou bien $F_2 \in$ étiquette de $s\}$.
    NB : dans cette définition on fait à nouveau abstraction du marquage.
  - Définition : on dit qu'une *branche ouverte satisfait* Ev ssi elle contients des sommets éléments de f_{Ev} infiniment souvent.
  - Résultat : *Une interprétation (partielle) décrite par une branche ouverte est un modèle de la racine si et seulement si elle saisfait toute formule existentielle qui est une sous-formule d'une formule de la racine.*

Etude d'un tableau pour $\Box\Diamond p$.

Figure 7

Etude des tableaux de la fig. 7 (qui après élagage reste le même.

Dépliage :



la boucle la plus à droite ne décrit pas un modèle : on procrastine toujours l'obtention de p

Propriétés des tableaux

On a les théorèmes suivants :

- Terminaison : Toute construction d'un tableau termine.
- Correction par rapport à l'insatisfiabilité : Si un tableau est clos, alors sa racine est insatisfiable.
- Completude par rapport à l'insatisfiabilité : Si E est un ensemble de formules insatisfiable, alors tout tableau de racine E est clos.

Ces trois propriétés permettent d'utiliser la méthode des tableaux comme algorithme de décision de la satisfiabilité de formules de LTL.

Comment prouver les propriétés des tableaux ?

Terminaison

- Les seules formules qui peuvent apparaître dans un tableau dont la racine est un ensemble E de formules sont des sous-formules d'éléments de E ou des $\bigcirc$ de ces sous-formules : au max $2n$, où n est le nombre de symboles dans E.
- Donc un tableau pour E contient au max $2^{(2n)}$ sommets distincts si E contient n symboles/.
- La construction du tableau termine, au max, avec $2^{(2n)}$ sommets : temps exponentiel.

NB : le problème de la décision *SAT ?* pour LTL (propositionnelle) est PSPACE-complet.

Rappel : $P \subseteq NP \subseteq PSPACE \subseteq EXPTIME \subseteq EXSPACE$
On sait : PSPACE=NPSPACE, mais on ne sait pas si $P \subset NP$ et si $P \subset PSPACE$.

Comment prouver les propriétés des tableaux ?

Correction par rapport à l'insatisfiabilité

On prouve la réciproque : Si l'ensemble de formules racine est satisfiable, alors tout tableau pour E est ouvert.

Idée de la preuve

- On montre que si on applique une règle d'expansion à un sommet s dont l'étiquette est E, et tout élément de E est vrai à $\mathcal{I}, s$, alors ceci reste vrai pour au moins une des deux (éventuelles) expansions si la règle n'est pas ($\bigcirc$), et tout élément de E est vrai à $\mathcal{I}, s'$, où s' est le successeur de s, sinon.
- Quand on élague, on élimine exclusivement des sommets ayant des étiquettes insatisfiables.
- Donc, si la racine est satisfiable, on n'élimine pas tous les sommets, et le tableau final est ouvert.

Comment prouver les propriétés des tableaux ?

Complétude par rapport à l'insatisfiabilité

On prouve la réciproque : Si un tableau pour E est ouvert, alors il existe un modèle de E.

Idée de la preuve

- Déjà vu comment construire une interprétation (partielle) à partir d'une branche ouverte, à condition que cette branche satisfasse toute formule existentielle (*eventuality*) qui est une dous-formule de la racine.
- Il reste donc à montrer que si le tableau est ouvert, alors au moins une branche de ce type existe, et c'est un modèle de la racine E.
- La difficulté : plusieurs formules existentielles dans un sommet. Par exemple : $E = \Diamond p, \Diamond \neg p$: il faudra montrer qu'on pourra toujours ordonnancer les formules existentielles à satisfaire.

Un exemple de tableau plus complexe

Pour prouver que l'équité faible d'une transaction est une conséquence logique de l'équite forte à l'aide des tableaux, on peut prouver la validité de la formule :
$((\Box\Diamond \mathit{enabled}_\alpha) \rightarrow \Diamond\Box \mathit{executed}_\alpha)$
$\rightarrow$
$((\Diamond\Box \mathit{enabled}_\alpha) \rightarrow \Diamond\Box \mathit{executed}_\alpha)$

c'est à dire l'insatisfiabilité de l'ensemble de formules :
$\{\Diamond\Box\neg \mathit{enabled}_\alpha) \vee \Diamond\Box \mathit{executed}_\alpha,$
$((\Diamond\Box \mathit{enabled}_\alpha)\ \Box\Diamond\neg \mathit{executed}_\alpha\}$
Pourquoi ?

Il suffit (!) de construire un tableau clos ayant comme racine cet ensemble. Avec une machine, c'est facile. A la main...

Ω-Automates

- Déjà vu qu'un système informatique peut se modéliser avec un système de transitions (exemple du ressort)
- Ce dernièr peut être réecrit comme un Ω-automate, qui reconnait des mots de longueur infinie. Mot infini lu : code une éxécution du système.
- Le type le plus simple de Ω-automates : *automates de Büchi*. Et il y a un lien fort entre ces automates et les formules LTL, et même avec les tableaux pour LTL : à voir dans la suite.
- Expressions lues par un automate de Büchi : expressions Ω-regulières.

Expressions Ω-regulières : Exp_ω

Σ : un alphabet,

Exp abrège : ensemble des expressions regulières (celles des automates finis standard) et Exp_Ω abrège : ensemble des expressions Ω-regulières.

$Exp := s \mid \epsilon \mid Exp + Exp \mid Exp.Exp \mid Exp^+ \mid Exp^*$
ou $s \in \Sigma$.

$Exp_\Omega := (Exp)^\Omega \mid Exp.(Exp)^\Omega \mid Exp_\Omega + Exp_\Omega$

ϵ est le mot vide, $+$ est l'union, . est la concatenation (souvent on omet le .), * construit une suite finie d'expressions, $+$ construit une suite finie et non-vide d'expressions (opérateur reguliers)

NB : On peut simuler Exp^+ par $ExpExp^*$.

Ω construit une suite infinie d'expressions ; par exemple, si $a \in \Sigma$, $a^\Omega = aaaaa...$

Expressions Ω-regulières : Exp_ω

Exemple d'expression Ω-régulière pour $\Sigma = \{\alpha, \beta\}$:
$(\beta^*\alpha)^\Omega$ qui décrit le langage contenant un nombre infini de α.

Par exemple, ce langage contient les mots :
$\alpha\alpha\alpha\alpha...$, car on le voit comme : $(\epsilon\alpha)(\epsilon\alpha)(\epsilon\alpha)...$

$\alpha\beta\alpha\beta\alpha\beta\alpha\beta...$ (le motif $\alpha\beta$ se répéte systématiquement dans la suite)
On peut le voir comme : $(\epsilon\alpha)(\beta\alpha)(\beta\alpha)(\beta\alpha)...$

$\beta\alpha\beta\beta\alpha\beta\beta\beta\alpha....$ (le nombre de β entre deux α augmente toujours de 1)
On peut le voir comme : $(\beta\alpha)(\beta\beta\alpha)(\beta\beta\beta\alpha)...$

etc.

Le langage décrit par $(\alpha^*\beta)^\Omega$ contient un nombre infini de mots, bien sûr, et chaque mot est lui même infini.

Automates de Büchi

Pour des raisons qui seront clarifiées dans la suite (lien avec les tableaux), on va étiquetter les états, avec les symboles à lire, pas les arcs.

Un automate de Büchi est une 6-ple $\langle \Sigma, S, \Delta, I, L, F \rangle$ où :

1. Σ est un *alphabet* fini ;
2. S est un ensemble fini d'*états* (NB. « état » : mot surchargé ! Utilisé pour les sommets de : une inteprétation $\mathcal{I}$ de LTL, d'un tableau, d'un automate.) ;
3. $\Delta \subseteq S \times S$ est la *relation de transition* ;
4. $I \subseteq S$ est l'ensemble des états *initiaux* ;
5. $L : S \to \Sigma$ est une fonction qui étiquette (*labels*) les états ;
6. $F \subseteq S$ est l'ensemble des états d'*acceptation*.

Exemple d'automate de Büchi

Appelons $\mathcal{A}_1$ l'automate suivant :

1. $\Sigma = \{\alpha, \beta\}$
2. $S = \{s_0, s_1\}$
3. $\Delta = \{\langle s_0, s_0\rangle, \langle s_0, s_1\rangle, \langle s_1, s_0\rangle, \langle s_1, s_1\rangle\}$
4. $I = S$
5. $L(s_0) = \alpha$ et $L(s_1) = \beta$
6. $F = \{s_0\}$.

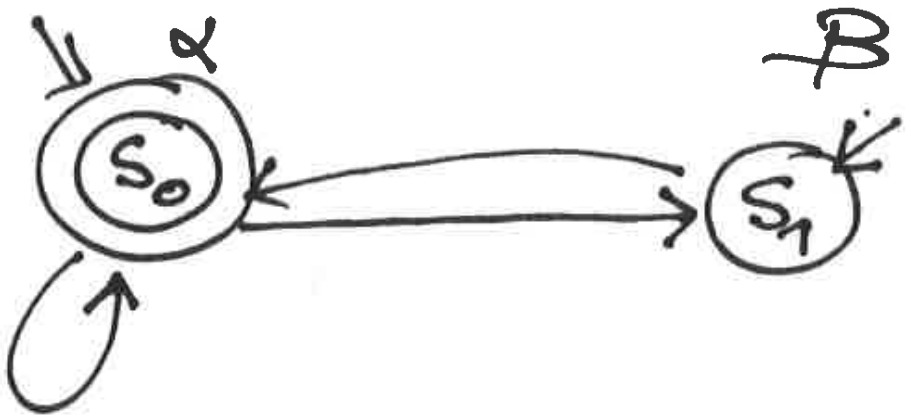
Comment visualizer cet automate ? Figure 8

Conventions de représentation : une flêche sans départ rentre dans s si $s \in I$, et s est entouré par un cercle si $s \in F$.

Exemple d'automate de Büchi

Figure 8

$\mathcal{A}_1$



Comment lire un mot (infini) ?

Un mot infini m sur l'alphabet Σ est une séquence infinie $\sigma_0, \sigma_1, \sigma_2...$ de symboles de Σ.

On peut la réprésenter comme une fonction $m : \mathbb{N} \to \Sigma$: $m(0) = \sigma_0$, $m(1) = \sigma_1$, $m(2) = \sigma_2$,...

Par exemple, le mot $m_1 = (\alpha\beta)(\alpha\beta)(\alpha\beta)...$ peut être vu comme la fonction m_1 telle que :
$m_1(n) = \alpha$ si n est pair, et $m_1(n) = \beta$ si n est impair.

C'est alors plus facile de définire proprement un *run* (exécution) d'un automate $\mathcal{A}$ sur un mot infini m, c'est à dire, intuitivement, un chemin **infini** de l'automate, à partir d'un état initial, qui lit ce mot. $\Rightarrow$

Run

Soit $\mathcal{A} = \langle \Sigma, S, \Delta, I, L, F \rangle$ un automate (de Büchi). Soit m un mot sur l'alphabet Σ (mot vu comme une fonction).

Un *run* de $\mathcal{A}$ sur l'entrée m est une application $\rho : \mathbb{N} \to S$ telle que :

1. $\rho(0) \in I$: on démarre la lecture à un état initial ;
2. Pour $i \geq 0$, $\langle \rho(i), \rho(i+1) \rangle \in \Delta$: on avance d'un état à un autre en suivant la relation de transition Δ ;
3. Pour tout $i \geq 0$, $m(i) = L(\rho(i))$: le i-ème symbole de m et égale à l'étiquette de l'état $\rho(i)$ du run ;

En particulier, pour $i = 0$, la 3ème condition dit que le premier symbole de m est lu à un état initial.

Si au moins un run de $\mathcal{A}$ sur m existe, alors on dit que $\mathcal{A}$ *lit* m. Pourquoi « au moins une » ? Definition d'automates non déterministe !

Exemples de runs

Soit $\mathcal{A}_1$ l'automate déjà vu.

Au tableau : runs pour les mots infinis suivants, que $\mathcal{A}_1$ peut lire :
$\alpha\alpha\alpha\alpha...$
$\alpha\beta\alpha\beta\alpha\beta\alpha\beta...$
$\beta\alpha\beta\beta\alpha\beta\beta\beta\alpha....$

De facto, $\mathcal{A}_1$ peut lire tout mot infini sur $\Sigma = \{\alpha, \beta\}$.
Mais il n'accepte pas tous ce mots. Comme l'on verra :
$\mathcal{A}_1$ lit $\alpha\alpha\alpha \underbrace{\beta\beta\beta...}_{\textit{toujours } \beta}$ mais il ne l'accepte pas.

Comment accepter des mots infinis ?

Soit $\mathcal{A} = \langle \Sigma, S, \Delta, I, L, F \rangle$ un automate (de Büchi).
Soit ρ un run d'un automate $\mathcal{A}$ pour un mot $m \in \Sigma^\Omega$.

On note $\inf(\rho)$ l'ensemble des états de $\mathcal{A}$ qui apparaissent infiniment souvent dans ρ.

Le run ρ *accepte* m quand $\inf(\rho) \cap F \neq \emptyset$, c'est à dire qu'un état d'acceptation apparaît infiniment souvent dans ρ.

Le langage $\mathcal{L}(\mathcal{A})$ est l'ensemble de tous les mots de Σ^Ω tels que chacun d'eux a au moins un run qui l'accepte.

NB : Selon la définition d'automate, $\mathcal{A}$ peut permettre plusieurs runs pour un même mot (non-détérminisme) ! Plus sur ce point après.

Comment accepter des mots infinis ? Suite

Soit $\mathcal{A}_1$ l'automate de Büchi déjà vu.

$\mathcal{L}(\mathcal{A}_1)$ est décrit par l'expression Ω-régulière $(\beta^*\alpha)^\Omega$:
l'ensemble des mots sur $\Sigma = \{\alpha, \beta\}$ qui contiennent, chacun,
le symbole α un nombre infini de fois.

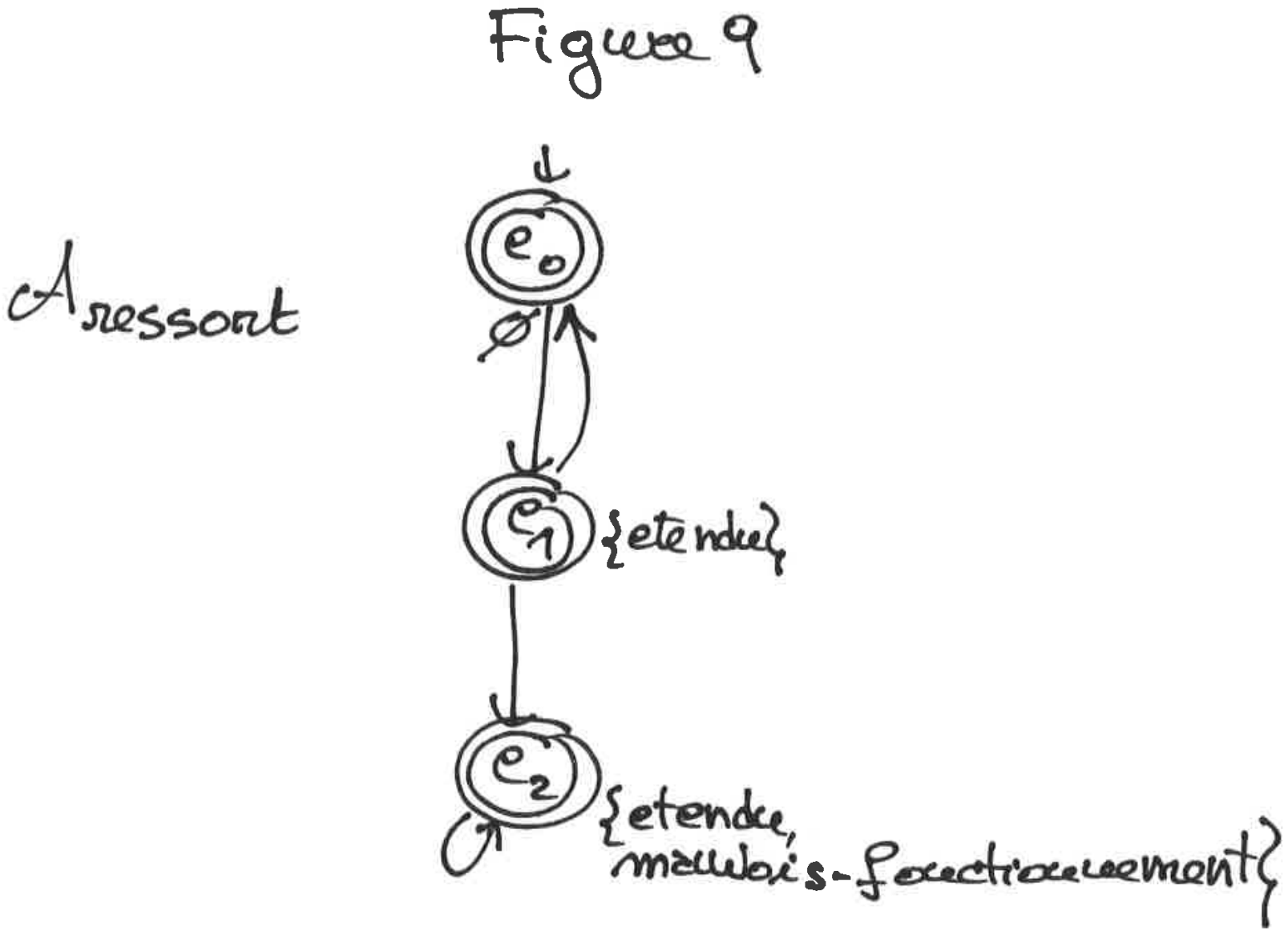
Codage de systèmes et automates de Büchi

Si $\Sigma =$ ensemble de tous les sous-ensembles d'un ensemble de variables booléennes P, noté 2^P, alors on peut coder un système informatique S par un automate, et un chemin infini code une exécution d'un système.

Par ex., le graphe de transition de la figure 1 (ressort) peut être vu comme un automate de Büchi $\mathcal{A}_{ressort}$ si :

- On ignore les étiquettes de arcs ;
- $\Sigma = 2^{\{etendu, mauvais\_fonctionnement\}}$;
- $S = \{e_0, e_1, e_2\}$;
- $\Delta = \{\langle e_0, e_1\rangle,\ \langle e_1, e_0\rangle,\ \langle e_1, e_2\rangle,\ \langle e_2, e_2\rangle\}$
- $I = \{e_0\}$;
- $L(e_0) = \emptyset$, $L(e_1) = \{etendu\}$,
  $L(e_2) = \{etendu, mauvais\_fonctionnement\}$;
- $F = S$

Automates de Büchi et contraintes sur les éxécutions : $\mathcal{A}_{ressort}$



Automates de Büchi et contraintes sur les éxécutions

Pourquoi dans $\mathcal{A}_{ressort}$ on a posé $F = S$?
On a stipulé que tout mot lu code une éxécution acceptable (légale).

Et si on imposait que chaque éxécution legale doit passer par e_1 un nombre infini de fois, quel ensemble d'états serait F ?
Il serait : $F = \{e_1\}$.
NB : On imposerait alors une forme de contrainte d'équité sur les éxécutions acceptables : les seules executions acceptables sont celles où le ressort fonctionne toujours et il est étendu infiniment souvent.

Automates de Büchi et contraintes sur les éxécutions

Imposer que chaque éxécution légale doit passer par e_1 un nombre infini de fois révient à dire que, quand on voit chacune des éxécutions légales comme une interprétation donnée $\mathcal{I}$ de LTL, on doit avoir que
$\mathcal{I}_0 \models \Box\Diamond(etendu \wedge \neg mauvais\_fonctionnement)$.

Il faut donc que l'automate réjecte certaines intérprétations, même si il peut le lire.

Le choix approprié de $F \subset S$ est la pour rejecter certaines exécutions comme exécutions légales du système modélisé.

Automates modélisant des systèmes informatique

Dans la suite, on considérera toujours des automates de Büchi où l'alphabet Σ dans le quel les mots à lire sont écrits est l'ensemble des sous-ensembles d'un ensemble de variables booléennes P, c'est-à-dire 2^P.

Réprésentation compacte d'automates

Symbole étiquette d'état = sous-ensemble de l'ensemble P des variables booléennes = interprétation classique : pas toujours pratique.

Exemple. Soit $P = \{p, q\}$. Automate $\mathcal{A}_2$, où $\Sigma = 2^P$ et :

- $S = \{e_0, s_1, s_2, s_3\}$;
- $\Delta = \{\langle e_0, s_1\rangle, \langle s_1, e_0\rangle, \langle e_0, s_2\rangle, \langle s_2, e_0\rangle, \langle e_0, s_3\rangle, \langle s_3, e_0\rangle\}$
- $I = \{e_0\} = F$;
- $L(e_0) = \{q\}$, $L(s_1) = \{p\}$, $L(s_2) = \{p, q\}$, $L(s_3) = \emptyset$.

L'étiquette de e_0 rend vraie la formule $(\neg p) \wedge q$.
Les étiquettes des états s_1, s_2 et s_3 rendent toutes vraie la formule $p \vee \neg q$. Ces états, ont exactement le même ensemble de successeurs (c'est $\{e_0\}$) et le même ensemble de prédécesseurs (c'est $\{e_0\}$).

Fusion de s_1, s_2 et s_3 dans un unique état s $\Rightarrow$

Réprésentation compacte d'automates, suite

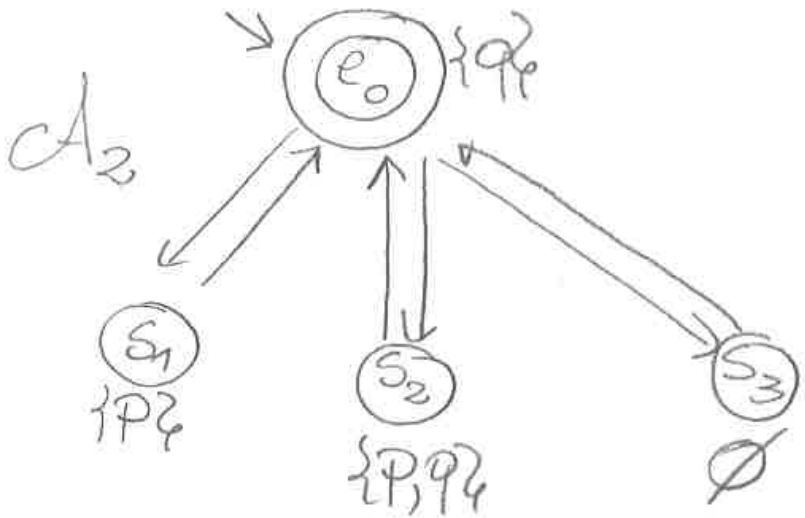
Une autre réprésentation du même automate $\mathcal{A}_2$, où Σ reste $P = \{p, q\}$:

Exemple.

- $S = \{e_0, s\}$;
- $\Delta = \{\langle e_0, s\rangle\langle s, e_0\rangle\}$
- $I = \{e_0\} = F$;
- $L(e_0) = (\neg p) \wedge q$, $L(s) = p \vee \neg q$.

Réprésentation compacte d'automates, suite

Figure 10



A_2 avec une réprésentation COMPACTE

Réprésentation compacte d'automates, suite

Que faut-il changer pour passer de la *réprésentation standard* de l'automate $\mathcal{A}_2$ à celle compacte ?
NB : l'automate ne change pas, c'est juste la façon de le *réprésenter* (écrire) qui change !

1. Avec la fonction d'étiquette L, à la place d'associer à un état un élément de $\Sigma = 2^P$, on lui associe une formule booléenne dont les variables booléénes sont dans P. NB : *True* fait partie de cette famille de formules.
2. Dans la définition de run de l'automate sur un mot m, à la place de la condition :
   « Pour tout $i \geq 0$, $m(i) = L(\rho(i))$ »
   il faudra considérer :
   Pour tout $i \geq 0$, $m(i)$ rend vraie la formule $L(\rho(i))$.

Attention : **L'automate représenté de façon compacte continue à lire seulement des mots qui sont des suites d'ensembles de variables de P !**

Exemples de lecture de mots par $\mathcal{A}_2$, selon les deux réprésentations

$\{q\}\{p\}\{q\}\{p\}\{q\}\{p\}\{q\}\{p\}\ldots$
(on commence par $\{q\}$ et le motif $\{q\}\{p\}$ se répète).
Accepté. Non compacte : un chémin acceptant est :

$e_0\ s_1\ e_0\ s_1\ e_0\ s_1\ \ldots$

$\{q\}\{p\}\{p,q\}\{q\}\emptyset\{q\}\{p\}\{p,q\}\{q\}\emptyset\{q\}\{p\}\{p,q\}\{q\}\emptyset\ldots$
(le motif $\{q\}\{p\}\{p,q\}\{q\}\emptyset$ se répète)
Accepté. Non compacte : un chémin acceptant est :

$e_0,\ s_1,\ e_0,\ s_2,\ e_0,\ s_3,\ e_0,\ s_1,\ e_0,\ s_2,\ e_0,\ s_3\ldots.$

$\{q\}\{p\}\{q\}\{q\}\ldots.$
(on commence par $\{q\}\{p\}\{q\}\{q\}$ puis peu importe).
Réfusé : on ne peut pas lire la seconde occurrence de $\{q\}$.

Automates Compacts

Dans la suite, on considérera toujours des automates de Büchi où $\Sigma = 2^P$, mais qui sont réprésentés de façon compacte.

C'est plus pratique, pour plein de raisons.

Automates Non-Déterministes

Un automate de Büchi est dit *non-déterministe* si S contient au moins deux états distincts, e_i et e_j, tels que e_i et e_j ont la même étiquette et on a au moins un de ce deux cas :

1. $e_i, e_i \in I$; (on peut commencer à lire un mot d'au moins 2 états)
2. Il existe un état e_k tel que $\langle e_k, e_i \rangle \in \Delta$ **et** $\langle e_k, e_j \rangle \in \Delta$, avec $e_i \neq e_j$ (on a au moins deux choix d'état successeur pour continuer à lire un mot à partir de e_k)

Si cela se vérifie, forcement il aura plus qu'un run qui lira un même mot m. Et il suffit un run qui accepte m, pour déclarer ce mot comme accepté par l'automate.

La définition d'automate de Büchi donnée est générale.
Êlle permet le non-déterminisme.

Automates Non-Déterministes

Attention : tout automate « standard » qui lit un mot fini, peut être toujours détérminisé. Mais ceci est faux pour les automates de Büchi : voir l'exemple suivant.

Exemple d'Automate de Büchi intrinséquement non-déterministes

$\mathcal{A}_3 = \langle \Sigma, S, \Delta, I, L, F \rangle$ où (réprésentation compacte !) :

1. $\Sigma = 2^{\{q\}}$
2. $S = \{e_0, e_1\}$
3. $\Delta = \{\langle e_0, e_0 \rangle, \langle e_0, e_1 \rangle, \langle e_1, e_1 \rangle\}$
4. $I = \{e_0, e_1\}$
5. $L(e_0) = \mathit{True}$, $L(e_1) = \{\neg q\}$
6. $F = \{e_1\}$

$\mathcal{L}(\mathcal{A}_3) =$ ensemble des suites infinies d'éléments de $2^{\{q\}}$ tels que $\{q\}$ apparaît un nombre **fini** de fois.

Exemple d'Automate de Büchi intrinséquement non-déterministes

Pourquoi non détérminisme, ici ? Par exemple, le mot $\{q\}\emptyset\emptyset\emptyset...$ est lu par le run $\rho_1 = e_0, e_0, e_0, e_0...$ et aussi par le run $\rho_2 = e_0, e_1, e_1, e_1....$ Le premier run ne l'accepte pas, le second si.

$\nexists$ un automate deterministe reconnaissant exactement le même langage que $\mathcal{A}_3$) !

Essayez d'en trouver un...

Pas possible de le trouver, et cela se **prouve** !

Associer un automate à une formule de LTL

Etant donné une formule quelconque F de LTL, il existe toujours au moins un un automate de Büchi $\mathcal{A}_F$ tel que :

1. Si F est insatisfiable, alors $\mathcal{L}(\mathcal{A}_F) = \emptyset$
2. Sinon, tout mot $m \in \mathcal{L}(\mathcal{A}_F)$ code une interprétation $\mathcal{I}$ qui est un modèle de F.

Exemples

• Si on voit α et β comme des variables booléennes, alors l'automate $\mathcal{A}_1$ déjà vu, où $\mathcal{L}(\mathcal{A}_1)$ est l'ensemble de mots décrit par $(\beta^*\alpha)^\Omega$, peut être associé à $F = \Box\Diamond\alpha$.

• L'automate $\mathcal{A}_2$ déjà vu peut être associé à la formule F :

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

Associer un automate à une formule de LTL, suite

Il existe un algorithme qui permet d'extraire un automate $\mathcal{A}_F$ à partir d'une formule F de LTL.

On peut le présenter de plusieurs façons.

Ici, nous exploiterons des choses que nous savons déjà faire avec la méthode des tableaux.

Utilité ?

Mais à quoi sert-il associer un automate $\mathcal{A}_F$ à une formule F de LTL, puisque on sait déjà utiliser les tableaux, pour tester si une formule est satisfiable ?

Un peu de patience...

Ca sert à faire de la vérification automatique des propriétés souhaitées pour un système S pour une modélisation $\mathcal{M}$ donnée (*model checking*), mais, cela, on le verra plus tard.

Soit, par exemple, $P = \{q\}$.
Supposons que, quand on passe d'un automate présenté de façon non-compacte à sa réprésentation compacte, on fusionne en un seul état e deux états e_i et e_j tels que : $L(e_i) = \{q\}$ et $L(e_i) = \emptyset$.
Alors, on peut étiquetter e par $q \vee \neg q$, ou, ce qui révient au même, par la formule *True*.

Une façon de voir e : c'est un état qui « ne filtre rien » : l'étiquette *True* « laisse passer » n'importe quel symbole lu (= intéprétation booléenne) !

Des tableaux aux automates

Une généralisation utile : les GBA

Les *automates généralisés de Büchi* (GBA) sont une généralisation des automates de Büchi.

C'est une généralisation utile, si on veut passer d'une formule F à un automate $\mathcal{A}_F$.

Toutefois, les GBA « n'ajoutent pas de pouvoir expressif » aux automates de Büchi standard, qu'on va appeler : BA.

Ce qui veut dire : tout automate GBA peut être traduit en un BA acceptant le même langage (et vice-versa).
Plus tard on le démontrera.

Des tableaux aux automates

Définition des GBA

Un *automate généralisé de Büchi* (GBA) $\mathcal{A}$ est exactement comme un automate de Büchi (BA) : $\mathcal{A} = \langle \Sigma, S, \Delta, I, L, F \rangle$, sauf que :

1. On a un ensemble d'ensembles d'états d'acceptation :

$$F = \{E_1, ..., E_n\}$$

   où $n \geq 1$.

2. La condition d'acceptation change : un run ρ qui lit un mot m l'accepte quand ρ passe par chaque E_i infiniment souvent, c'est à dire :
   Pour chaque $E_i \in F$: $inf(\rho) \cap E_i \neq \emptyset$

Si $n = 1$, *de facto*, on retombe sur les BA, car $F = \{E_1\}$, et on pourra « idéntifier » $\{E_1\}$ avec E_1.

Des tableaux aux automates

Première étape

Soit A une formule de LTL.

On construit un tableau T de racine A.

On l'élague partiellement en appliquant, exclusivement :

1. Supprimer tout sommet contenant p et $\neg p$, pour quelque variable booléénne p ;
2. Supprimer tout sommets dont on a aussi effacé tous les successeurs.

Notons T'_A le tableau temporaire pour A ainsi obtenu.

Des tableaux aux automates

Seconde Etape

Pour chaque formule existentielle (*eventuality*) Ev qui est une sous formule de A, si Ev a la forme $\Diamond F_2$ ou bien $F_1 \mathrm{U} F_2$, construire f_{Ev}. La définition de f_{Ev} a été déjà donnée, et on la rappelle :

$f_{Ev} = \{s$ est un sommet du tableau $\mid Ev \notin$ etiquette de s ou bien $F_2 \in$ étiquette de $s\}$

Cas particulier : Un sommet s dont l'étiquette est **la** formule *True* appartient à l'étiquette de tout f_{Ev}.

Des tableaux aux automates

Troisième Etape

Construire le GBA $\mathcal{A}_A = \langle \Sigma, S, \Delta, I, L, F \rangle$ ainsi :

- $\Sigma = 2^P$, où $P =$ l'ensemble des variables booléennes de F ;
- $S =$ ensemble des **états du tableau** T'_A (révoir la définition !) ;
- $I =$ ensemble de ces états s du tableau T'_A tels que il existe un chemin (éventuellement de longueur 0) qui va de la racine de T'_A à s où s est le seul élément de S ;
- $\Delta = \{\langle s, s' \rangle \mid s, s' \in S$ et il existe un chemin de T'_A qui va de s à s' et qui ne touche pas d'autre états $\}$;

Et L ? Et F ? Continuons.

Des tableaux aux automates

Troisième Etape, Suite

- L : une fonction : $S \rightarrow C$ où C est l'ensemble de toutes le conjonctions de littéraux sur Σ (littéral : un symbole $l \in P \cup \{True\}$, ou bien la négation de cela).
  Pour $s \in S$:
  - Quelque soit $s \in S$, $L(s) = l_1 \wedge ... l_q$, où chaque l_i est un littéral dans l'étiquette du sommet s dans T'_A.
    Cas particulier : cette étiquette ne contient pas de littéraux, et $q = 0$; dans ce cas, $L(s) = True$.

Des tableaux aux automates

Troisième Etape, Suite

- Si A n'a pas des sous-formules qui soient des *eventualities*, alors $F = S$.

  Sinon, soient $Ev_1, .., Ev_m$, $m \geq 1$ toutes les *eventualities* qui sont des sous-formules de A, et, pour chaque $i \geq m$, soit f_{Ev_i}.
  Alors : $F = \{f_{Ev_1} \cap S, .., .f_{Ev_m} \cap S\}$.

Fin de l'étape 3, et fin de la construction de $\mathcal{A}_A$ à partir de A, tout court.

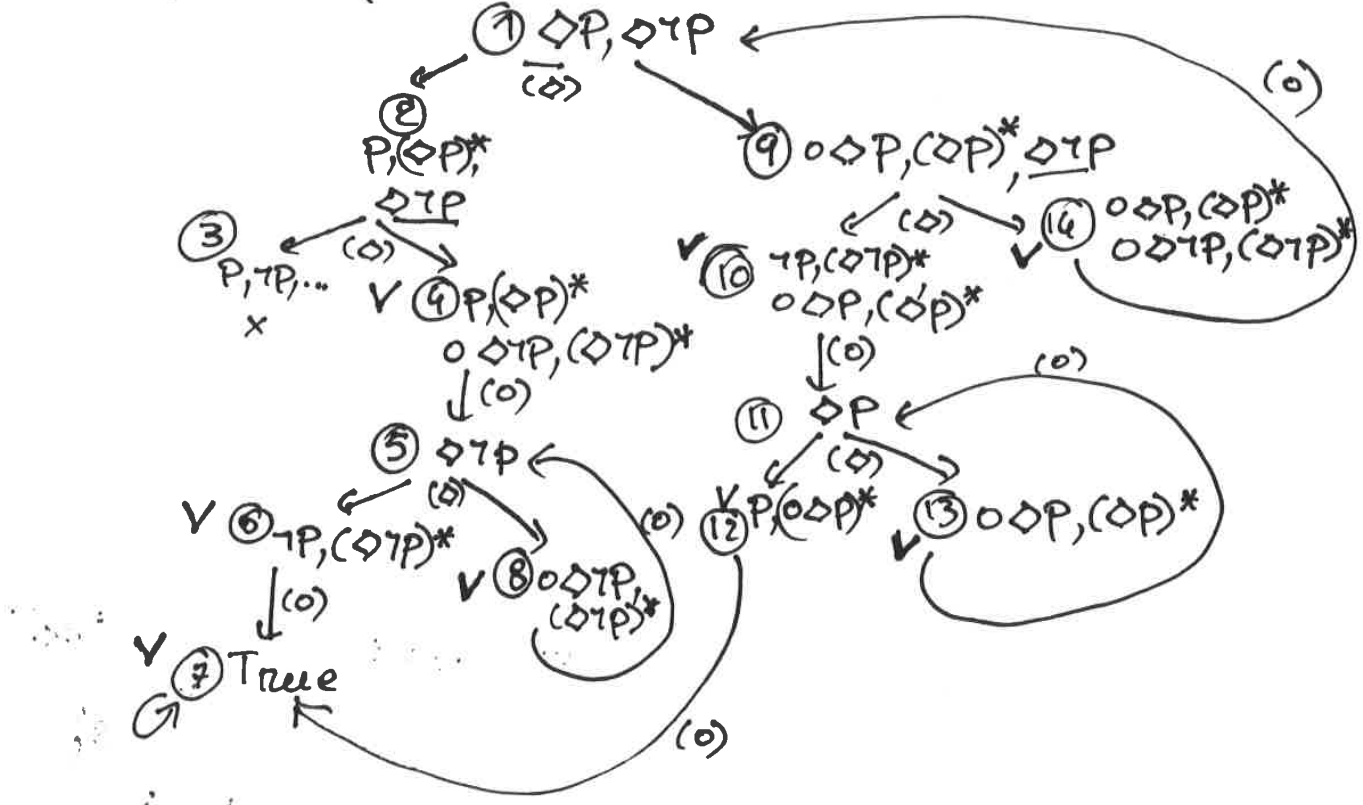
Des tableaux aux automates

Exemple

Soit $A = \Diamond p \wedge \Diamond \neg p$. Construction, selon l'algorithme, de $\mathcal{A}_A$: Figures 12 et 13

Figure 12

Pour simplifier, la racine des tableaux pour $\Diamond p \wedge \Diamond \neg p$ sera $\{\Diamond p, \Diamond \neg p\}$.



Des tableaux aux automates

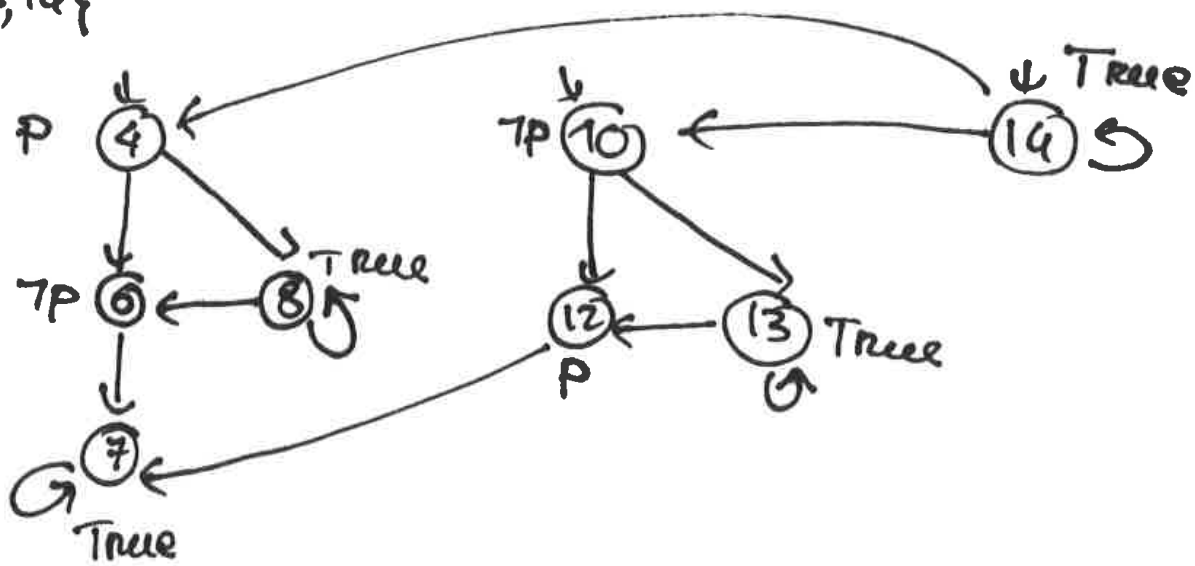
Exemple, suite

Figure 13

$S_{\Diamond p} = \{2, 4, 5, 6, 7, 8, 12\}$ $S_{\Diamond \neg p} = \{6, 7, 10, 11, 12, 13\}$

Etats : $\{\{6, 7, 8, 10, 12, 13, 14\}$

$\mathcal{A}_{\Diamond p \wedge \Diamond \neg p}$:



$F = \{\{4, 6, 7, 8\}, \{6, 7, 10, 12, 13\}\}$

Des tableaux aux automates

Exemple, suite

NB : Dans cet exemple pour $A = \Diamond p \wedge \Diamond \neg p$: dès que $Ev_1 = \Diamond p$ est « réalisée » on passe (dans le tableau et l'automate) à un état qui « ne s'occupe plus de Ev_1 ». Même chose pour $Ev_2 = \Diamond \neg p$.

Et dès que les deux sont réalisées, on passe dans un état dont l'étiquette est la formule *True*, dans lequel on reste toujours.

Comparer avec ce qui se passe pour $F = (\Box \Diamond p) \wedge (\Box \Diamond \neg p)$.