Langages de spécification – cours 3

Introduction en logique temporelle

Catalin Dima

Spécification des propriétés temporelles

- Les automates permettent de décrire des ensembles de comportements.
  - “Langage de spécification” visuel, facile à suivre pour des exemples “pédagogiques”.
  - Mais très vite dépassant les capacités de compréhension des humains lorsqu’on modélise un vrai problème !
- L’algorithmique des automates est facile à implémenter car provenant de l’algorithmique de graphes :
  - Essentiellement atteignabilité et recherche de composants fortement connexes.
  - Plus des constructions de graphes plus grands à partir de graphes plus petits.
- Recherche d’un moyen de spécification plus facile d’écrire et de lire !
  - Equivalent avec les automates finis.
  - Mais bien plus compacte que les automates !
  - Dérivé de la logique.
  - Capable de spécifier de manière simple des propriétés simples :
    - La feuille est toujours dirigée par un insecte.
    - Les trois insectes peuvent franchir l’autre rive.
    - La barrière est baissé jusqu’au moment où le train va dépasser le passage à niveau.
- Solution (possible) : **Logique temporelle linéaire**.

LTL

Extension de la logique propositionnelle :

- Propositions atomiques : l, s, flag, etc..
  - Par extension, on considérera que des expressions comme $\mathit{pc}1 = 4$ sont des propositions atomiques – on les notera souvent aussi $p_{\mathit{pc}1=4}$.
- Tous les opérateurs booléens sont présents : $\wedge, \vee, \neg, \Rightarrow, \rightarrow$.
- Opérateurs temporels du futur :
  - Demain (angl. Next) : $X\phi$ or $\bigcirc p$.
  - Jusqu'à (angl. Until) : $\phi \mathcal{U} \psi$.
  - Toujours (angl. Globally) : $G\phi$ or $\Box \phi$.
  - À l'avenir (angl. Forward) : $F\phi$ or $\Diamond \phi$.

LTL

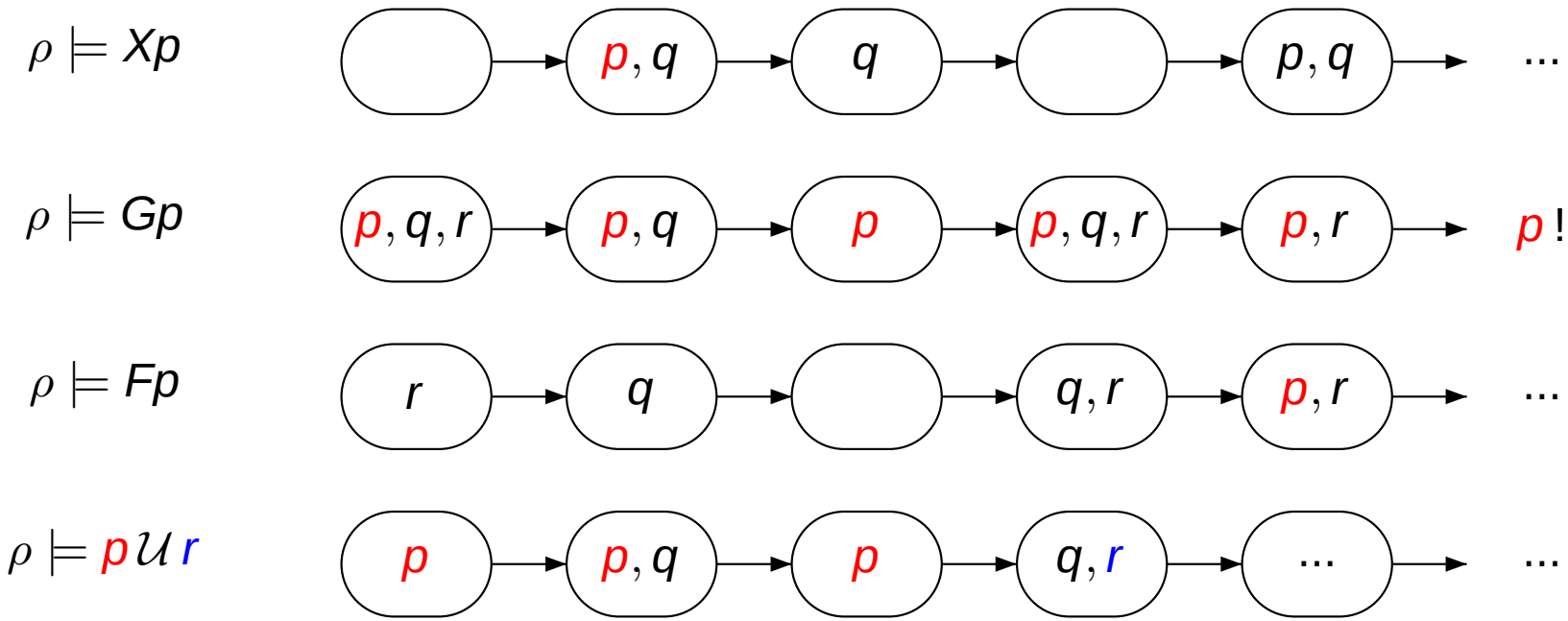
Extension de la logique propositionnelle :

- Propositions atomiques : l, s, flag, etc..
  - Par extension, on considérera que des expressions comme $pc1 = 4$ sont des propositions atomiques – on les notera souvent aussi $p_{pc1=4}$.
- Tous les opérateurs booléens sont présents : $\wedge, \vee, \neg, \Rightarrow, \rightarrow$.
- Opérateurs temporels du futur :
  - Demain (angl. Next) : $X\phi$ or $\bigcirc p$.
  - Jusqu'à (angl. Until) : $\phi \mathcal{U} \psi$.
  - Toujours (angl. Globally) : $G\phi$ or $\Box \phi$.
  - À l'avenir (angl. Forward) : $F\phi$ or $\Diamond \phi$.
- Opérateurs du passé :
  - Hier (angl. Yesterday) ϕ held : $Y\phi$ or $\bullet\phi$.
  - Depuis (angl. Since) : $\phi \mathcal{S} \psi$.
  - Toujours (angl. Historically) : $H\phi$ or $\blacksquare\phi$.
  - Une fois (angl. Once) : $O\phi$ or $\blacklozenge\phi$.

Cadre sémantique

- Toute formule s'interprète sur une *trace* $\rho : \mathbb{N} \longrightarrow 2^{AP}$.
  - Ou, de manière équivalente, sur un mot ***infini*** sur l'alphabet 2^{AP}.
- Sur un tel mot ***infini***, chaque variable propositionnelle a une valeur de vérité à chaque instant :
  - $l \in \rho(0)$ veut dire $l = \mathit{vrai}$ à l'instant 0.
  - $p_{pc1=4} \in \rho(251)$ veut dire que le programme 1 est juste avant l'instruction 4 à l'instant 251 dans la trace.
- Chaque formule sera interprétée aussi *à chaque instant dans la trace* !

Sémantique intuitive



Semantique formelle

$(\rho, i) \models p$	if $p \in \rho(i)$
$(\rho, i) \models \phi_1 \wedge \phi_2$	if $(\rho, i) \models \phi_1$ and $(\rho, i) \models \phi_2$
$(\rho, i) \models \neg\phi$	if $(\rho, i) \not\models \phi$
$(\rho, i) \models X\phi$	if $(\rho, i+1) \models \phi$
$(\rho, i) \models \phi_1 \mathcal{U} \phi_2$	s'il existe $j \geq i$ tel que $(\rho, j) \models \phi_2$ et pour tout $i \leq k < j, (\rho, k) \models \phi_1$
$(\rho, i) \models F\phi$	s'il existe $j \in \mathbb{N}$ tel que $(\rho, j) \models \phi$
$(\rho, i) \models G\phi$	si pour tout $j \in \mathbb{N}, (\rho, j) \models \phi$

- Mais les règles sémantiques pour $\mathcal{U}$ suffisent, car :

$$\Diamond \phi = \text{true} \mathcal{U} \phi$$
$$\Box \phi = \neg \Diamond \neg \phi$$

- Sémantique similaire pour les opérateurs du passé.
- Plus précisément....

Semantique, suite

- D'autres opérateurs du temps futur possibles :
  - $\phi_1 \mathcal{W} \phi_2$: ϕ_1 doit être vrai *sauf à partir du moment où* ("*weakly until*") ϕ_2 est vrai.
  - $\phi_1 \mathcal{R} \phi_2$: ϕ_2 *relève* (*releases*) ϕ_1.
- Semantique :

$$\phi_1 \mathcal{W} \phi_2 = \phi_1 \mathcal{U} \phi_2 \vee \Box \phi_1$$
$$\phi_1 \mathcal{R} \phi_2 = \neg(\neg\phi_1 \mathcal{U} \neg\phi_2) = \phi_2 \mathcal{W}(\phi_1 \wedge \phi_2)$$

Quelques formules

... et leur correspondant en langage naturel

- Exclusion mutuelle : $G\neg(pc_1 = 3 \wedge pc_2 = 3)$.
  - Jamais les deux programmes ne se trouveront en même temps dans leur section critique.
- Atteignabilité : $F(chass \wedge loup \wedge chevre \wedge chou)$.
  - Le chasseur amènera sur la rive gauche le loup, la chèvre et le chou.
- Convergence : $FG(Voyager - atteint - Alpha - Centauri)$.
  - Voyager va arriver et va rester jusqu'à la fin de l'Universe dans le système solaire d'Alpha Centauri.
- Recurrence : $GF(pc_1 = sect\_crit)$.
  - Le premier programme entrera une infinité de fois dans sa section critique.
- Honneteté (fairness) : $G((pc_1 = 2) \longrightarrow F(pc_1 = 3))$.
  - Si le premier programme arrive une infinité de fois à l'entrée de sa section critique, il va entrer une infinité de fois dans cette section critique.
  - ... et donc il ne sera jamais "*enfamé*".

Construction de formules en LTL à partir des énoncés en langage naturel

Supposons un feu tricolore de circulation, avec trois variables r, o, v.

- Un seul feu de couleur est allumé à tout instant : .....
- Le feu ne peut pas passer directement de vert à rouge : ...
- Si le feu est vert à un instant quelconque, il redeviendra vert dans trois instants de temps (3s, 3 ticks d'horloge,...).
- Le feu doit rester rouge jusqu'à un moment où il doit passer vert.
- Quand le feu est rouge, alors il va passer au vert dans le futur, après avoir été jaune pendant un certain temps.

Construction de formules en LTL à partir d'énoncés en langage naturel

Fausse solution d'exclusion mutuelle :

```
while (true) {
   flag1 := true ;
   wait ( !flag2)
      section critique 1
   flag1 := false  ;
}
```

```
while (true) {
   flag2 := true ;
   wait ( !flag1)
      section critique 2
   flag2 := false  ;
}
```

- À chaque fois que le programme 1 n'avance pas, la valeur de la variable *flag*1 n'est pas modifiée.
- Lorsque le programme 1 exécute l'instruction 3, la variable *flag*2 doit être fausse.
- La variable *flag*2 garde sa valeur jusqu'à l'instant où on exécute l'instruction 2 du programme 2 (sans garantie quant à l'existence de cet instant !)
- Les deux programmes se trouvent, à l'avenir, en état d'interblocage.

Sémantique de LTL

- Tautologie : formule qui est vraie indépendamment des affectations de valeurs de vérité aux propositions atomiques.
- Exemples :

$$\neg \bigcirc p \Leftrightarrow \bigcirc \neg p$$
$$\bigcirc p \Rightarrow \Diamond p$$
$$\Diamond \Diamond p \Rightarrow \Diamond p$$
$$\Box(p \wedge q) \Leftrightarrow \Box p \wedge \Box q$$
$$(\Diamond p \Rightarrow \Diamond q) \Rightarrow \Diamond(p \Rightarrow q)$$

- Formules que ne sont pas des tautologies :

$$\Diamond(p \wedge q) \Leftrightarrow \Diamond p \wedge \Diamond q$$
$$p \,\mathcal{U}\, (q \,\mathcal{U}\, r) \Leftrightarrow (p \,\mathcal{U}\, q) \,\mathcal{U}\, r$$

- Pour prouver qu'elles ne sont pas des tautologies, donner un contre-exemple !

Axiomatization

- Axiomes et règles de déduction pour la partie propositionnelle (n'importe quel système déductif).
- Axiomes et règles pour X and $\mathcal{U}$:
  - Distributivité : $X\phi \wedge X(\phi \longrightarrow \psi) \longrightarrow X\psi$.
  - Temps linéaire : $\neg X\phi \Leftrightarrow X\neg\phi$.
  - Axiome de point fixe pour "until" : $\phi\mathcal{U}\psi \Leftrightarrow \psi \vee (\phi \wedge X(\phi\mathcal{U}\psi))$.
  - Règle d'inférence pour "toujours" : à partir de ϕ déduire $G\phi$.
  - Règle d'inférence pour "until" : à partir de $\phi' \longrightarrow \neg\psi \wedge X\phi'$ déduire $\phi' \longrightarrow \neg(\phi\mathcal{U}\psi)$.

Systèmes de transitions et logique temporelle linéaire

- Un système de transitions $A = (Q, \Pi, Q_0, \delta, \lambda)$ définit un ensemble de traces (infinies) : celles qui représentent la concaténation des étiquettes des chemins infinis partant d'un état initial.
  - Pour rappel, $\delta \subseteq Q \times Q$ et $\lambda : Q \longrightarrow 2^{\Pi}$.
  - ... et on peut noter $Runs^{\omega}(A)$ l'ensemble des traces définies par un système de transitions.
- Une formule LTL définit aussi un ensemble de traces : celles qui satisfont la formule.
- Peut-on faire un lien entre ces deux modes de spécification des ensembles de traces ?

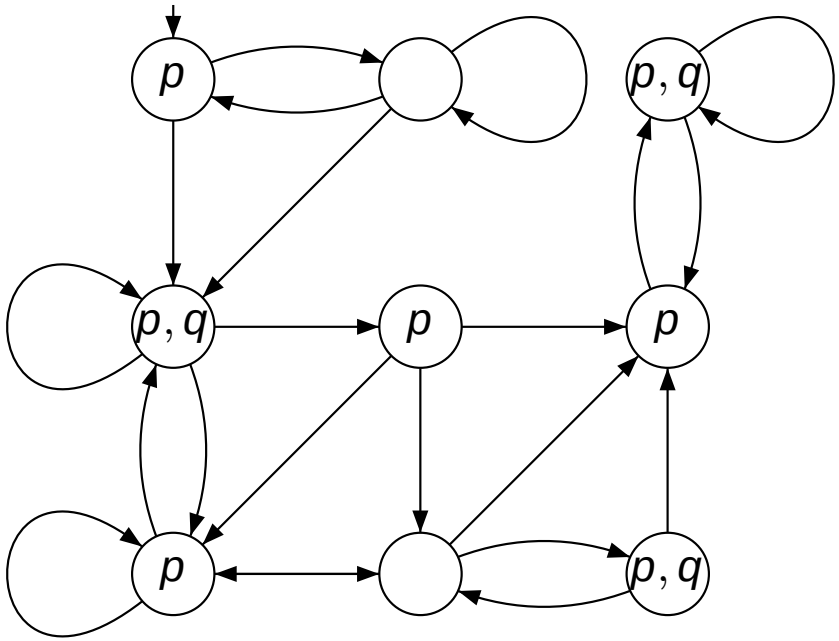
Problème du model-checking

- Étant un système de transitions $A = (Q, \Pi, Q_0, \delta, \lambda)$ et une formule ϕ, est-ce que toutes les traces définies par T satisfont ϕ ?

$$\forall \rho \in \mathit{Runs}^{\omega}(T), (\rho, 0) \models \phi ?$$

- Problème plus général : étant donné un système A, un état q et une formule ϕ, est-il le cas que pour toute trace ρ qui part de q, $(\rho, 0) \models \phi$?

Problème du model-checking



Quels sont les états pour lesquels *toutes les traces* qui quittent l'état respectif satisfont :

- Gp ?
- $p \mathcal{U} q$?

Convention : si un état n'est pas étiquetté par une variable a, c'est que cette variable est fausse dans l'état respectif !

Formules LTL et systèmes de transition

- $\Pi = \{p\}$.
- Est-ce qu'il existe un système de transitions dont l'ensemble de traces (infinies) soit exactement l'ensemble des traces satisfaisant FGp ?

Formules LTL et systèmes de transition

- $\Pi = \{p\}$.
- Est-ce qu'il existe un système de transitions dont l'ensemble de traces (infinies) soit exactement l'ensemble des traces satisfaisant *FGp* ?
- Non !
- Les systèmes de transitions ont besoin d'un critère plus puissant permettant de *filtrer* certaines traces !

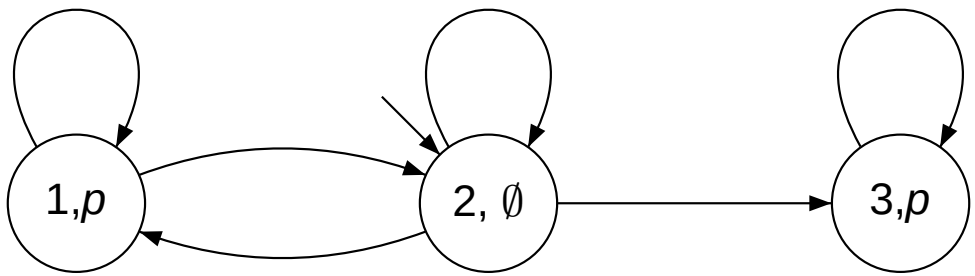
Mots infinis et états *répétitifs*

- Un automate de Büchi est un système de transition $A = (Q, \Pi, Q_0, \delta, \lambda, \mathcal{R})$,...
- ... dans lequel on a rajouté une composante $\mathcal{R} \subseteq Q$ définissant des états répétitifs.

Condition d'acceptation de Büchi

Une trace est acceptante si elle passe infiniment souvent par un des états de $\mathcal{R}$.

- Exemple pour *FGp* :



Avec $\mathcal{R} = \{3\}$.

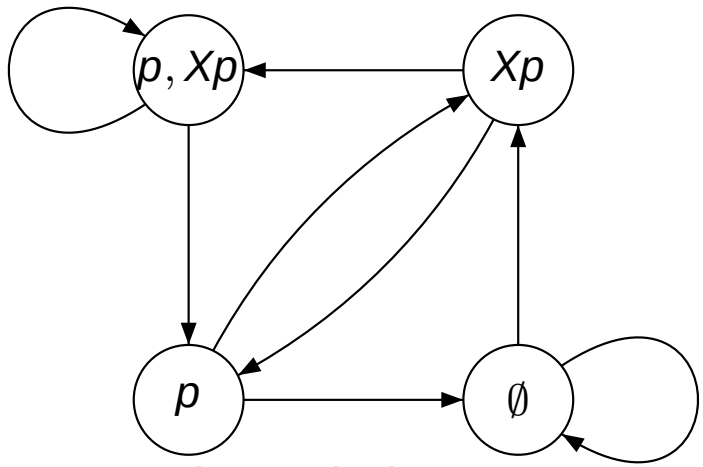
Convention : si un état n'est pas étiquetté par une variable a, c'est que cette variable est fausse dans l'état respectif !

Algorithmes d'automates de Büchi

- "Langage" vide ?
  - Vérifier s'il existe un état dans $\mathcal{R}$ qui est : atteignable et fait partie d'un circuit.
  - C.à.d. $\mathcal{R}$ appartient à une composante fortement connexe non-triviale et atteignable.
- Constructions pour l'union similaire aux automates finis.
- Intersection : il faut adapter les états répétitifs !
  - Petit exemple...

Automates de Büchi à partir de formules LTL

- Construction pour Xp and $\neg Xp$:

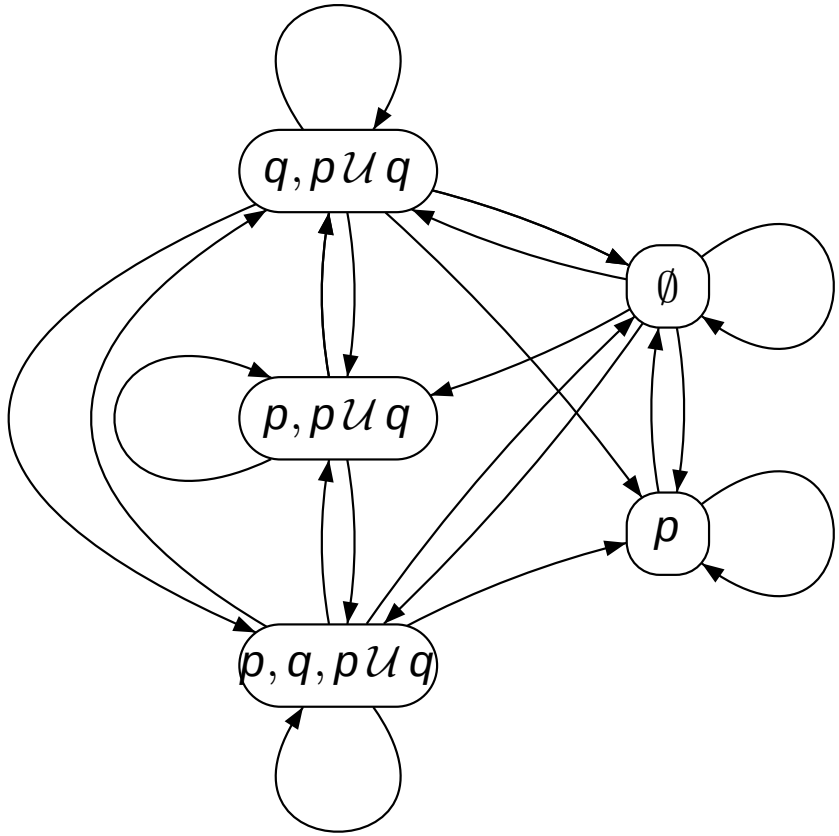


- Pas besoin de spécifier des états répétitifs !

Convention : si un état n'est pas étiquetté par une variable a, c'est que cette variable est fausse dans l'état respectif !

Automates de Büchi à partir de formules LTL

- Construction pour $p \mathcal{U} q$ and $\neg(p \mathcal{U} q)$.



- Cette fois-ci il faut spécifier des états répétitifs :
  - On ne peut pas rester à l'infini dans l'état $p, p \mathcal{U} q$!

Convention : si un état n'est pas étiquetté par une variable a, c'est que cette variable est fausse dans l'état respectif !

Algorithme de model-checking

- Construire l'automate de Büchi A_ϕ correspondant à $\neg\phi$.
- Vérifier que $A \cap A_\phi = \emptyset$.
  - Si $A \cap A_\phi = \emptyset$ alors aucune trace de A (partant d'un état initial) ne satisfait $\neg\phi$,
  - ... donc toutes les traces de A satisfont ϕ !

Vérification des propriétés LTL en NuSMV

Commande `check_ltlspec` :

- Option `-p "formule"` : vérifier que le système de transition défini dans le fichier chargé satisfait la formule indiquée.
- Des formules (spécifications) LTL peuvent être déclarées dans le code avec le mot-clé `LTLSPEC`, et vérifiées avec `check_ltlspec -n numero`, où `numero` est le numéro de la formule dans la liste des `LTLSPEC`s.