Automates et logique temporelle LTL

Souffan Nathan Bouarah Romain

Supervisé par François Laroussinie

8 juin 2021

Table des matières

1 Introduction

Pour vérifier la validité d'un programme ou d'un système il peut être intéressant de connaître quelles propriétés doivent être vraies et à quels moments. La logique traditionnelle ne permet de qualifier ce genre d'évènements et il est nécessaire de s'intéresser à une logique dépendant du temps. Avec celle ci on peut ainsi vérifier la validité d'un programme grâce au *model checking* dont on ne parlera pas ici mais qui serait la suite logique de ce document.

2 Logique temporelle

Si pour la logique une propriété ne peut être que vraie ou fausse, celle ci peut être vraie à un certain moment puis fausse par la suite en logique temporelle. Il existe plusieurs représentations du temps pour la logique temporelle : arborescente, continue, linéaire, ... Pour une représentation linéaire, on peut prendre le temps comme étant $\mathbb{Z}$ pour considérer le passé, ou encore comme ce sera le cas par la suite $\mathbb{N}$. Celle ci s'appelle la LTL (Logique Temporelle de temps Linéaire[1]).

2.1 LTL

On se place donc dans le cas d'une logique temporelle de temps linéaire. On définit AP l'ensemble des propositions atomiques. Il y a alors une valuation des AP pour chaque n dans $\mathbb{N}$.

2.1.1 Syntaxe

On définit l'ensemble des formules de LTL comme l'ensemble engendré inductivement et librement par les règles de constructions suivantes :

Atomes Si $p \in AP$ alors p est une formule.
Tautologie $\top$ est une formule.
Négation Si φ est une formule propositionnelle, $\neg\varphi$ en est aussi une.
Conjonction Si φ et ψ sont des formules, $(\varphi \wedge \psi)$ est une formule.
Suivant (Next) Si φ est une formule, $X\varphi$ est une formule.
Jusqu'à (Until) Si φ et ψ sont des formules, $(\varphi U \psi)$ est une formule.

2.1.2 Sémantique

On pose $Q = \{q_1, \dots q_n\}$ un ensemble d'états Pour les formules LTL, les modèles sont des couples (p, l) où $p \in Q^\omega$ et $l : Q \to 2^{AP}$, l nous indique ainsi

1. A ne pas confondre avec la logique linéaire qui est différente.

quels atomes sont vrais pour chaque état.
Les formules sont interprétées sur une position $i \geq 0$ le long d'une exécution étiquetée (p, l).
On note ainsi $p, l, i \models \varphi$ le fait que φ est vraie en i le long de (p, l), de plus on définit l'équivalence $\equiv$ en posant : $\varphi \equiv \psi$ si $\forall p, l, i,\ [p, l, i \models \varphi \Leftrightarrow p, l, i \models \psi]$

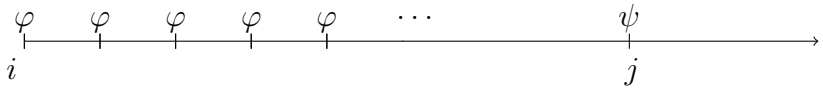
$p, l, i \models v \Leftrightarrow v \in l(p(i))$ où $v \in AP$
$p, l, i \models \top$
$p, l, i \models \varphi \wedge \psi \Leftrightarrow [(p, l, i \models \varphi)$ et $(p, l, i \models \psi)]$
$p, l, i \models \neg\varphi \Leftrightarrow p, l, i \not\models \varphi$
$p, l, i \models X\varphi \Leftrightarrow p, l, i + 1 \models \varphi$
$p, l, i \models \varphi U \psi \Leftrightarrow [\exists j \geq i$ tel que, $p, l, i \models \psi$ et $\forall i \leq j < k,\ p, l, k \models \varphi]$

On ajoute à cela plusieurs macros définis de la sorte :

$\bot = \neg\top$ (Absurde).
$\varphi \vee \psi = \neg(\neg\varphi \wedge \neg\psi)$ (Disjonction).
$(\varphi \to \psi) = (\neg\psi \vee \varphi)$ (Implication).
$F\varphi = \top U \varphi$, (Future).
$G\varphi = \neg F \neg\varphi$ (Globally).

Représentation

$\varphi U \psi$



Proposition 1. *Soient Q un ensemble d'états, $p \in Q^\omega$, $l : Q \to 2^{AP}$ et $i \in \mathbb{N}$, on a :*

1. $p, l, i \not\models \bot$
2. $p, l, i \models \varphi \vee \psi \Leftrightarrow [p, l, i \models \varphi$ *ou* $p, l, i \models \psi]$
3. $p, l, i \models F\varphi \Leftrightarrow p, l, i \models [\exists j \geq i$ *tel que* $p, l, j \models \varphi]$
4. $p, l, i \models G\varphi \Leftrightarrow p, l, i \models [\forall j \geq i$, *on a,* $p, l, j \models \varphi]$

Démonstration.

1. Trivial
2. Preuve similaire à celle pour la logique propositionnelle.

3.

$$\begin{aligned} p,l,i \models F\varphi &\Leftrightarrow p,l,i \models \top U\varphi \\ &\Leftrightarrow p,l,i \models [\exists j \geq i \text{ tel que } p,l,j \models \varphi \text{ et } \forall i \leq j < k,\ p,l,k \models \top] \\ &\Leftrightarrow p,l,i \models [\exists j \geq i \text{ tel que } p,l,j \models \varphi] \end{aligned}$$

4.

$$\begin{aligned} p,l,i \models G\varphi &\Leftrightarrow p,l,i \models \neg F\neg\varphi \\ &\Leftrightarrow p,l,i \not\models F\neg\varphi \\ &\Leftrightarrow p,l,i \not\models [\exists j \geq i \text{ tel que } p,l,j \models \neg\varphi] \\ &\Leftrightarrow p,l,i \not\models [\exists j \geq i \text{ tel que } p,l,j \not\models \varphi] \\ &\Leftrightarrow p,l,i \models [\forall j \geq i \text{ on a } p,l,j \models \varphi] \end{aligned}$$

□

Exemple 1.
$a, b \in AP$

- GFa : (toujours(futur a)) ce qui signifie il y a une infinité de positions où a est vrai.
- $aU(Gb)$: a est vrai tant que b est faux, dès que a est faux, b est toujours vrai par la suite

Les propriétés sur les opérateurs de la logique usuelle reste vraies dans la logique temporelle de temps linéaire. On peut ajouter des propriétés sur les opérateurs de la logiques temporelle.

Proposition 2.

1. $\neg(X\varphi) \equiv X(\neg\varphi)$
2. $\neg(G\varphi) \equiv F(\neg\varphi)$
3. $\neg(F\varphi) \equiv G(\neg\varphi)$
4. $X(\varphi \vee \psi) \equiv (X\varphi) \vee (X\psi)$
5. $X(\varphi \wedge \psi) \equiv (X\varphi) \wedge (X\psi)$
6. $X(\varphi U\psi) \equiv (X\varphi)U(X\psi)$
7. $F(\varphi \vee \psi) \equiv (F\varphi) \vee (F\psi)$
8. $G(\varphi \wedge \psi) \equiv (G\varphi) \wedge (G\psi)$
9. $\xi U(\varphi \vee \psi) \equiv (\xi U\varphi) \vee (\xi U\psi)$
10. $(\varphi \wedge \psi)U\xi \equiv (\varphi U\xi) \wedge (\psi U\xi)$

11. $F\varphi \equiv FF\varphi$
12. $G\varphi \equiv GG\varphi$
13. $\varphi U\psi \equiv \varphi U(\varphi U\psi)$
14. $\psi \vee (\varphi \wedge X(\varphi U\psi)) \equiv \varphi U\psi$
15. $G\varphi \equiv \varphi \wedge X(G\varphi)$
16. $F\varphi \equiv \varphi \vee X(F\varphi)$

Démonstration.

2.

$$\begin{aligned} p,l,i \models \neg(G\varphi) &\Leftrightarrow p,l,i \not\models G\varphi \\ &\Leftrightarrow \neg(\forall j \geq i \text{ tel que } p,l,j \models \varphi) \\ &\Leftrightarrow \exists j \geq i \text{ tel que } p,l,j \not\models \varphi \\ &\Leftrightarrow \exists j \geq i \text{ tel que } p,l,j \models \neg\varphi \\ &\Leftrightarrow p,l,i \models F(\neg\varphi) \end{aligned}$$

Toutes les preuves reposent sur ce type de démonstration, on laisse les autres en exercices.

□

3 Automate de Büchi

Les automates de Büchi sont un type particulier d'automate sur les mots infinis. Les automates sur les mots infinis (ou ω-automates) sont des automates finis qui acceptent des mots infinis.

Définition 1 (Automate de Büchi)**.** Un automate de Büchi est un quintuplet $\mathcal{A} = (\Sigma, Q, Q_I, \Delta, \mathscr{F})$ où :

— Σ est un ensemble fini appelé alphabet de $\mathcal{A}$.
— Q est un ensemble fini. Les éléments de Q sont les états de $\mathcal{A}$.
— $Q_I \subseteq Q$ est l'ensemble des états initiaux.
— $\Delta \subset Q \times \Sigma \times Q$ est l'ensemble des transitions.
— $\mathscr{F} \subseteq Q$ est l'ensemble des états finaux (ou états acceptants). Un mot w est accepté s'il existe une exécution acceptante de $\mathcal{A}$ sur w.

Définition 2 (Exécution)**.** Soient $w \in \Sigma^\omega$ un mot infini et $\mathcal{A} = (\Sigma, Q, Q_I, \Delta, \mathscr{F})$ un automate de Büchi. Une exécution de $\mathcal{A}$ sur w est une suite infinie $\rho = q_0q_1q_2\cdots \in Q^\omega$ telle que :

$$\forall i \geq 0 \quad (q_i, w_i, q_{i+1}) \in \Delta$$

Définition 3 (Exécution acceptante)**.** Soient $\mathcal{A} = (\Sigma, Q, Q_I, \Delta, \mathscr{F})$ un automate de Büchi et $\rho \in Q^\omega$ une exécution de $\mathcal{A}$. On dit que ρ est une exécution acceptante si :

$$Etats_{\#\infty}(\rho) \cap \mathscr{F} \neq \varnothing$$

où $Etats_{\#\infty}(\rho)$ est l'ensemble des états apparaissants une infinité de fois dans ρ.

Définition 4 (Langage reconnu)**.** Soit $\mathcal{A}$ un automate. Le langage reconnu par l'automate, noté $\mathcal{L}(\mathcal{A})$, est l'ensemble des mots w tel qu'il existe une exécution acceptante de w sur $\mathcal{A}$.

Exemple 2. Soit $\mathcal{A} = (\{a, b\}, \{q_0, q_1\}, \{q_0\}, \Delta, \{q_1\})$ un automate de Büchi. L'ensemble des transitions Δ est donné dans la figure.

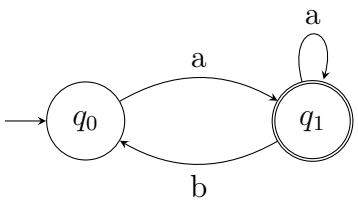


FIGURE 1 – Automate de Büchi

Cet automate reconnaît le mot $w = aaa\ldots$ car pour l'exécution $\rho = q_0 q_1^\omega$ (c'est la seule) de $\mathcal{A}$ sur w, l'état q_1 apparaît une infinité de fois dans ρ. En fait, le langage reconnu (ou accepté) est $a^\omega | a(a^*ba)^\omega | a(a^*ba)^*a^\omega$.

Les automates de Büchi généralisés sont une variante des automates de Büchi. La différence se situe sur la condition d'acceptation.

Définition 5 (Automate de Büchi généralisé)**.** Un automate de Büchi généralisé est un quintuplet $\mathcal{A} = (\Sigma, Q, Q_I, \Delta, \mathscr{F})$ où :

— Σ, Q, Q_I, Δ sont comme précédemment.

— $\mathscr{F} \subseteq \mathcal{P}(Q)$ est la condition d'acceptation. $\mathscr{F}$ est un ensemble d'ensembles finaux. De même, un mot w est accepté s'il existe une exécution acceptante de $\mathcal{A}$ sur w.

Pour un automate de Büchi généralisé, une exécution ρ de $\mathcal{A}$ est acceptante si :

$$\forall F \in \mathscr{F} \quad Etats_{\#\infty}(\rho) \cap F \neq \varnothing$$

4 Traduction de formule LTL en automate de Büchi

On souhaite "traduire" une formule LTL en un automate de Büchi. Plus précisément, on veut construire un automate qui reconnait les modèles de φ.

Pour une formule LTL φ, on peut voir un modèle (ρ, l) de φ (où $\rho \in Q^\omega$ et $l : Q \to 2^{AP}$) comme un mot infini sur l'alphabet 2^{AP}. Pour cela, il suffit simplement de considérer (ρ, l) comme le mot $l(\rho(1))l(\rho(2))\dots$

Définition 6. On note $SubF(\varphi)$ l'ensemble des sous formules de φ et leur négation.

Exemple 3. Si $\varphi = aUb$ alors $SubF(\varphi) = \{a, \neg a, b, \neg b, aUb, \neg(aUb)\}$.

Définition 7 (Sous-ensemble cohérent)**.** $q \in 2^{SubF(\varphi)}$ est cohérent si toutes les conditions suivantes sont vérifiées :

(i) Si $\psi_1 \wedge \psi_2 \in q$ alors $\psi_1 \in q$ et $\psi_2 \in q$.

(ii) Si $\psi_1 \vee \psi_2 \in q$ alors $\psi_1 \in q$ ou $\psi_2 \in q$.

(iii) $\psi \in q \iff \neg\psi \notin q$.

Remarque 1. En utilisant les lois de Morgan, on en déduit aussi que :

(i) Si $\neg(\psi_1 \wedge \psi_2) \in q$ alors $\psi_1 \notin q$ ou $\psi_2 \notin q$.

(ii) Si $\neg(\psi_1 \vee \psi_2) \in q$ alors $\psi_1 \notin q$ et $\psi_2 \notin q$.

Définition 8 (Sous-ensemble maximal)**.** $q \in 2^{SubF(\varphi)}$ est maximal si pour tout $\psi \in SubF(\varphi)$ on a soit $\psi \in q$ soit $\neg\psi \in q$.

Définition 9 (Sous-ensemble conforme à la sémantique de LTL)**.** $q \in 2^{SubF(\varphi)}$ est conforme à la sémantique de LTL :

(i) Si $\psi_1 U \psi_2 \in q$ alors on a soit $\psi_1 \in q$ soit $\psi_2 \in q$.

(ii) $\forall \psi_1 U \psi_2 \in SubF(\varphi)$ si $\psi_2 \in q$ alors $\psi_1 U \psi_2 \in q$.

Exemple 4. Soit $\varphi = aU(Xb)$ alors

$$SubF(\varphi) = \{a, \neg a, b, \neg b, Xb, \neg(Xb), aU(Xb), \neg(aU(Xb))\}$$

1. $q_1 = \{\neg a, b, Xb, aU(Xb)\}$ est un sous-ensemble cohérent, maximal et conforme à la sémantique de LTL.
2. $q_2 = \{\neg a, b, Xb, \neg(aU(Xb))\}$ est un sous-ensemble cohérent, maximal mais non conforme à la sémantique de LTL car on a Xb et $\neg(aU(Xb))$.

Définition 10. Soient AP l'ensemble des propositions atomiques et φ une formule LTL sur AP. L'automate de Büchi généralisé pour φ sur AP est donné par $\mathcal{A}_\varphi = (2^{AP}, Q, Q_I, \Delta, \mathscr{F})$ où :

- $Q \subseteq 2^{SubF(\varphi)}$ contient tout les sous-ensembles de $SubF(\varphi)$ qui sont cohérents, maximaux et conformes à la sémantique de LTL.
- $Q_I = \{q \in Q | \varphi \in q\}$. Autrement dit, tous les états contenant exactement notre formule de départ φ sont des états initiaux.
- Δ est l'ensemble des transitions (q, a, q') avec $q, q' \in Q$ et $a \in 2^{AP}$ vérifiant :
  - (i) $\forall p \in AP \quad p \in q \iff p \in a$ (i.e. a possède toutes les propositions atomiques de q)
  - (ii) $\forall X\psi \in SubF(\varphi) \quad X\psi \in q \iff \psi \in q'$
  - (iii) $\forall \psi_1 U \psi_2 \in SubF(\varphi) \quad \psi_1 U \psi_2 \in q \iff (\psi_2 \in q \vee (\psi_1 \in q \wedge \psi_1 U \psi_2 \in q'))$
- $\mathscr{F} = \{F_{\psi_1 U \psi_2} | \psi_1 U \psi_2 \in SubF(\varphi)\}$ où

$$F_{\psi_1 U \psi_2} = \{q \in Q | \psi_1 U \psi_2 \notin q \vee \psi_2 \in q\}$$

L'idée dérrière cette construction est de faire en sorte que l'automate devine quelles sont les sous-formules de φ qui sont vraies lorsqu'on lit un mot. Ce sont les états qui indiquent quelles sont les sous formules vraies.

Si l'automate devine que $\neg p$, avec $p \in AP$, est vraie cela revient à vérifier que $p \notin a$. Pour $X\psi$, l'automate vérifie qu'à l'état suivant ψ est bien vraie. Enfin, si l'on a $\psi_1 U \psi_2$ vérifier ψ_2 qu'au prochain état n'est pas suffisant car ψ_2 peut se réaliser bien plus loin. Dans ces cas là, la solution consiste à conserver $\psi_1 U \psi_2$ pour se rappeler qu'on doit encore vérifier cette formule. Finalement, c'est la condition d'acceptation qui nous permet de dire si la vérification $\psi_1 U \psi_2$ est repoussée indéfiniment ou non.

Exemple 5. Avec $\varphi = Xa$ on obtient l'automate suivant :

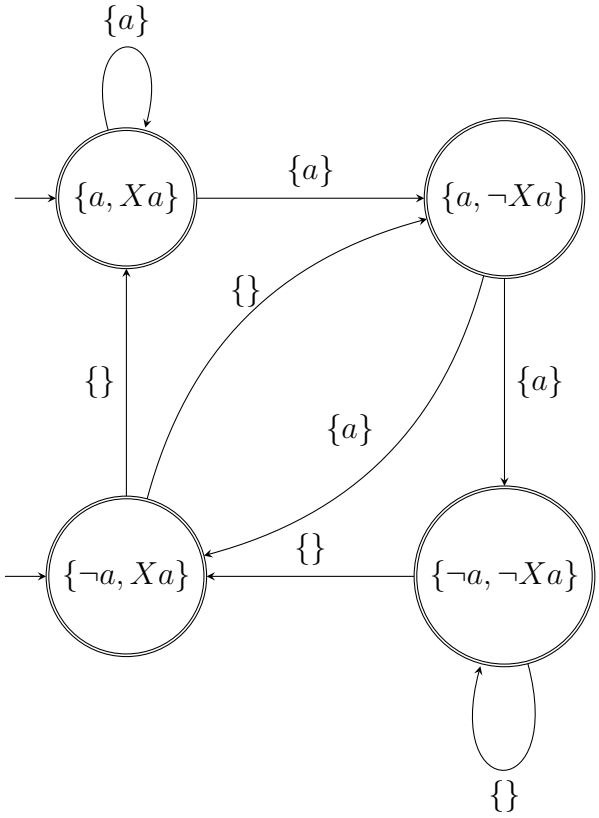


FIGURE 2 – l'automate de Büchi généralisé pour Xa sur $\{a\}$

Sur cet exemple, une transition $\{\}$ signifie qu'on a $\neg a$, autrement dit l'absence d'une variable dans une transition signifie que l'on a sa négation.

On remarque que l'automate peut atteindre une taille exponentielle dans la taille de la formule φ. Si on a $Card(SubF(\varphi)) = n$ alors l'automate peut avoir au plus 2^n états.

Théorème 1. *Soient :*
- *AP un ensemble de propositions atomiques.*
- *φ une formule LTL sur AP.*
- *$w \in (2^{AP})^\omega$ un mot infini sur l'alphabet 2^{AP} tel que $w, 0 \models \varphi$.*
- *$\mathcal{A}_\varphi$ l'automate de Büchi généralisé pour φ sur AP.*

Alors $w \in \mathcal{L}(\mathcal{A}_\varphi)$.

Démonstration. On veut montrer que $w \in \mathcal{L}(\mathcal{A}_\varphi)$. D'après la définition 3, cela revient à montrer qu'il existe une exécution acceptante de w sur $\mathcal{A}_\varphi$.

On pose $\forall i \geq 0 \quad q_i = \{\psi \in SubF(\varphi) | w, i \models \psi\}$ et $\rho = q_0 q_1 q_2 \dots$ Montrons que ρ est une exécution acceptante sur w dans $\mathcal{A}_\varphi$.

— $w, 0 \models \varphi$ donc $\varphi \in q_0$ donc q_0 est bien un état initial.

— Il y a bien des transition (q_i, w_i, q_{i+1}) dans $\mathcal{A}_\varphi$.

— ρ vérifie bien la condition d'acceptation. En effet, si $\psi_1 U \psi_2 \in q_i$, alors $w, i \models \psi_1 U \psi_2$ (par construction des q_i) donc $\exists j \geq i$ tel que $w, j \models \psi_2$ et $\forall k, i \leq k \leq j \quad w, k \models \psi_1$. Enfin, d'après la définition 9, on a aussi $\psi_1 U \psi_2 \in q_j$ et il existe un chemin valide jusqu'à q_j ainsi ρ passe infiniment souvent par $F_{\psi_1 U \psi_2}$.

□

Théorème 2. *Soit $\omega \in (2^{AP})^\omega$ et $p = q_0 q_1 \dots$ une exécution acceptante de $\mathcal{A}_\varphi$ du mot ω alors :*
$\forall i \geq 0, \forall \psi \in SubF(\varphi) : \ (\psi \in q_i \Leftrightarrow \omega_i \models \varphi)$

Démonstration. La preuve s'effectue par induction structurelle sur ψ

— $\psi = v \in AP$
- Si $v \in q_i$, alors par construction de $\mathcal{A}_\varphi$ on sait que $v \in \omega_i$, donc $\omega, i \models v$
- Inversement si $\omega, i \models v$ alors $v \in \omega_i$ et donc $v \in q_i$ par construction de l'automate.

— $\psi = \psi_1 \wedge \psi_2$
- Supposons $\psi_1 \wedge \psi_2 \in q_i$, alors par construction de $\mathcal{A}_\varphi$ on a $\psi_1, \psi_2 \in q_i$ et donc par hypothèse d'induction on a $\omega, i \models \psi_1$ et $\omega, i \models \psi_2$. Ainsi par définition de la LTL on a $\omega, i \models \psi_1 \wedge \psi_2$.
- Inversement, supposons $\omega, i \models \psi_1 \wedge \psi_2$, alors $\omega, i \models \psi_1$ et $\omega \models \psi_2$ donc par hypothèse d'induction, $\psi_1, \psi_2 \in q_i$. On en conclus par construction de $\mathcal{A}_\varphi$ que $\psi_1 \wedge \psi_2 \in q_i$

— $\psi = \neg\psi_1$
- Si $\neg\psi_1 \in q_i$ alors par construction de l'automate $\psi_1 \notin q_i$. Par hypothèse d'induction on obtient $\omega, i \not\models \psi_1$. Ainsi $\omega, i \models \neg\psi_1$
- Inversement, si $\omega, i \models \neg\psi_1$. Alors $\omega, i \not\models \psi_1$. Par hypothèse d'induction on obtient donc $\psi_1 \notin q_i$

— $\psi = X\psi_1$
- Supposons $\psi \in q_i$, alors on a $\psi_1 \in q_{i+1}$ par construction de $\mathcal{A}_\varphi$, par l'hypothèse d'induction on a donc $\omega, i+1 \models \psi_1$. Ainsi on a bien par définition $\omega, i \models X\psi_1$
- Inversement, si $\omega, i \models X\psi_1$ alors $\omega, i+1 \models \psi_1$ et donc par hypothèse d'induction $\psi_1 \in q_{i+1}$. Ainsi par construction de l'automate on a bien $X\psi_1 \in q_i$.

— $\psi = \psi_1 U \psi_2$

- Supposons $\psi_1 U \psi_2 \in q_i$, il y a alors deux possibilités.
  $\psi_2 \in q_i$: on a alors tout de suite le résultat voulu car $\omega, i \models \psi_2$ et donc $\omega, i \models \psi_1 U \psi_2$.
  $\psi_1 \in q_i$ et $\psi_1 U \psi_2 \in q_{i+1}$. Puisque p est une exécution acceptante, il existe une certain q_j avec $j \geq i+1$ et $\psi_2 \in q_j$ (Sinon on aurait $\psi_1 U \psi_2$ dans tous les états q_j où $j \geq i$ mais alors l'exécution ne serait pas bonne), de plus on a $\psi_1 \in q_i, q_{i+1}, \ldots, q_{j-1}$. Ainsi on en déduit que $\forall i \leq k < j$, $\omega, k \models \psi_1$ et $\omega, j \models \psi_2$. Par définition de la LTL on en conclus que $\omega, i \models \psi_1 U \psi_2$
- On suppose maintenant $\omega, i \models \psi_1 U \psi_2$. Alors par définition on a :
  $\exists j \geq i$ tel que $\omega, j \models \psi_2$ et $\forall i \leq k < j$, $\omega, k \models \psi_1$
  On a alors $\psi_2 \in q_j$ et $\psi_1 \in q_k$ $\forall i \leq k < j$.
  Ainsi on en conclus que $\psi_1 U \psi_2 \in q_i, q_{i+1} \ldots, q_j$.
  On suppose maintenant $\omega, i \models \psi_1 U \psi_2$. Alors par définition on a :
  $\exists j \geq i$ tel que $\omega, j \models \psi_2$ et $\forall i \leq k < j$, $\omega, k \models \psi_1$
  On a alors $\psi_2 \in q_j$ et $\psi_1 \in q_k$ $\forall i \leq k < j$.
  Ainsi on en conclus que $\psi_1 U \psi_2 \in q_i, q_{i+1} \ldots, q_j$.

Ainsi le théorème est démontré par induction structurelle. □

Corollaire 3. *Soit φ une formule LTL sur AP. On a, $\mathcal{L}(\mathcal{A}_\varphi) = mod(\varphi)$ où $mod(\varphi)$ est l'ensemble des modèles reconnues par φ.*

Démonstration. Par double inclusion en utilisant théorème 1 et théorème 2 □

5 Implémentation

5.1 Premières définitions des types

```
type ltlFormula =
  | Const of bool
  | Var of string
  | Or of ltlFormula * ltlFormula
  | And of ltlFormula * ltlFormula
  | Not of ltlFormula
  | Next of ltlFormula
  | Until of ltlFormula * ltlFormula

type state = ltlFormula list;;

type buchi = {
    (* Ensemble fini représentant l'alphabet *)
```

```
    alphabet : SetString.t;
    eval : (state, (state, SetString.t) Hashtbl.t) Hashtbl.t;

    (* Ensemble fini contenant tous les états *)
    states : state list;
    final_states : state list;
    init_states : state list;
  }
```

5.2 Création des états

```
let rec add_to_all_states f states = match f with
  | Const _ -> add_const_to_states states
  | Var v -> add_var_to_states v states
  | Or (o1, o2) ->
    let states = add_to_all_states o2 (add_to_all_states o1 states) in
    add_or_to_states f states
  | And (a1, a2) ->
    let states = add_to_all_states a2 (add_to_all_states a1 states) in
    add_and_to_states f states
  | Not n -> add_to_all_states n states
  | Next n -> add_next_to_states f (add_to_all_states n states)
  | Until (u1, u2) ->
    let states = add_to_all_states u2 (add_to_all_states u1 states) in
    add_until_to_states f states
```

Les sous parties détaillent une à une comment chaque cas est géré.

5.2.1 Ajout des constante

```
let add_const_to_states states = match states with
| [] -> [[Const true]]
| s :: _ -> if is_in_state s (Const true)
            then states
            else List.map (fun l -> (Const true) :: l) states
```

5.2.2 Ajout des variables

```
let add_var_to_states v states =
  let rec add_var_to_states_aux v states acc = match states with
    | [] -> acc
    | s :: states' ->
```

```
      if is_in_state s (Var v) then states
      else if is_in_state s (Not (Var v)) then states
      else let acc = (Var v :: s) :: ((Not (Var v)) :: s) :: acc in
        add_var_to_states_aux v states' acc
  in if states = [] then [[Var v]; [Not (Var v)]]
  else add_var_to_states_aux v states []
```

5.2.3 Ajout des OU

```
let add_or_to_states o states =
  let rec add_or_to_states_aux o states acc = match states with
    | [] -> acc
    | s :: states' ->
      if is_in_state s o then states
      else if is_in_state s (Not o) then states
      else match o with
        | Or (o1, o2) ->
          let acc = if is_in_state s o1 || is_in_state s o2
            then (o :: s) :: acc
            else ((Not o) :: s) :: acc
          in add_or_to_states_aux o states' acc
        | _ -> raise (Invalid_argument "Impossible")
  in add_or_to_states_aux o states []
```

5.2.4 Ajout des ET

```
let add_and_to_states a states =
  let rec add_and_to_states_aux a states acc = match states with
    | [] -> acc
    | s :: states' ->
      if is_in_state s a then states
      else if is_in_state s (Not a) then states
      else match a with
        | And (a1, a2) ->
          let acc = if is_in_state s a1 && is_in_state s a2
            then (a :: s) :: acc
            else ((Not a) :: s) :: acc
          in add_and_to_states_aux a states' acc
        | _ -> raise (Invalid_argument "Impossible")
  in add_and_to_states_aux a states []
```

5.2.5 Ajout des Next

```
let add_next_to_states n states =
  let rec add_next_to_states_aux n states acc = match states with
    | [] -> acc
    | s :: states' ->
      if is_in_state s n then states
      else if is_in_state s (Not n) then states
      else let acc = (n :: s) :: ((Not n) :: s) :: acc in
        add_next_to_states_aux n states' acc
  in add_next_to_states_aux n states []
```

5.2.6 Ajout des Until

```
let add_until_to_states u states =
  let rec add_until_to_states_rec u states acc = match states with
    | [] -> acc
    | s :: states' ->
      if is_in_state s u then states
      else if is_in_state s u then states
      else match u with
        | Until (u1, u2) ->
          let acc = if is_in_state s u2
            then (u :: s) :: acc
            else if not (is_in_state s u1)
            then ((Not u) :: s) :: acc
            else (u :: s) :: ((Not u) :: s) :: acc
          in add_until_to_states_rec u states' acc
        | _ -> raise (Invalid_argument "Impossible")
  in add_until_to_states_rec u states []
```

5.2.7 Récupération des états initiaux

```
let get_initial_states f states =
  let rec get_initial_states_aux f states acc = match states with
    | [] -> acc
    | s :: states ->
      let acc = if is_in_state s f
        then s :: acc
        else acc in
      get_initial_states_aux f states acc in
  get_initial_states_aux f states []
```

5.2.8 Récupération des états finaux

```
let get_final_states f states =
  let untils = get_list_of_untils f in
  let rec is_final_states s l = match l with
    | [] -> true
    | (Until (_, f2) as f) :: l ->
      if is_in_state s f && (not (is_in_state s f2)) then false
      else is_final_states s l
    | _ -> failwith "Impossible" in
  let rec get_final_states_aux states acc = match states with
    | [] -> acc
    | s :: states ->
      if is_final_states s untils
      then get_final_states_aux states (s :: acc)
      else get_final_states_aux states acc in
  get_final_states_aux states []
```

5.3 Création des transitions

Cette fonction vérifie si l'on peut créer une transition de from_state à to_state.

```
let can_create_transition from_state to_state =
  let rec check ltl =
    match ltl with
    | Next f -> is_in_state to_state f
    | Until (f1, f2) as f ->
      (is_in_state from_state f2) ||
      ((is_in_state from_state f1) && (is_in_state to_state f))
    | Not (Next f) -> not (check (Next f))
    | Not (Until(f1, f2)) -> not (check (Until (f1, f2)))
    | _ -> true
  in List.for_all check from_state;;
```

On va essayer de créer toutes les transitions possibles, d'où les deux List.iter pour effectuer un produit cartésien.

```
let create_all_transitions states =
  let n = List.length states in
  let transitions = Hashtbl.create n in
  let _ = List.iter (fun from_state -> let transition' =
  ↪  Hashtbl.create n in
```

```
                    let _ = List.iter (fun to_state ->
                        if can_create_transition from_state
                        ↪  to_state
                        then Hashtbl.add transition' to_state
                            (get_variables_from_state
                            ↪  from_state)
                        else ())
                        states in
                    Hashtbl.add transitions from_state
                    ↪  transition';)
        states in
    transitions;;
```

6 Bibliographies

Références

[1] *Automata : From Logics to Algorithms*, M. Y. Vardi & T. Wilke

[2] *https ://www.irif.fr/ francoisl/m2modspec.html*, F. Laroussinie