

Création et utilisation d'un algorithme de model-checking : à l'aide de la logique temporelle LTL

Présentation de **Valentin TOMAS-TORTONI**
(numéro de candidat : 12291)
travail réalisé avec **Tom LAHRIZIA**

1. Définitions

Model-Checking

Structure de Kripke

ω -mot et ω -automates

Logique LTL

Objectifs

2. Implémentation LTL

3. Faire l'intersection entre deux automates de Büchi

4. Annexe

Objectifs du Model-Checking

3 étapes clés :

- **Modéliser** : trouver une **bonne** modélisation pour le système pour lequel on souhaite vérifier une propriété
- **Définir** : la **propriété** que doit vérifier le modèle
- **Tester** : si le modèle vérifie la propriété

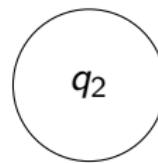
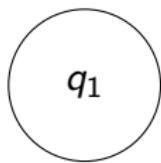
Définir

- Modèle pour décrire les systèmes
- Système formel pour décrire les propriétés

Structure de Kripke : $\mathcal{M} = (\mathcal{Q}, \mathcal{I}, \rightarrow, \mathcal{L})$

Définir

- Modèle pour décrire les systèmes
- Système formel pour décrire les propriétés

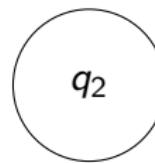
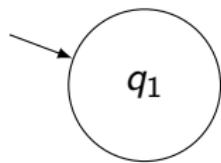


Structure de Kripke : $\mathcal{M} = (\mathcal{Q}, I, \rightarrow, \mathcal{L})$

$$\mathcal{Q} = \{q_1, q_2\}$$

Définir

- Modèle pour décrire les systèmes
- Système formel pour décrire les propriétés



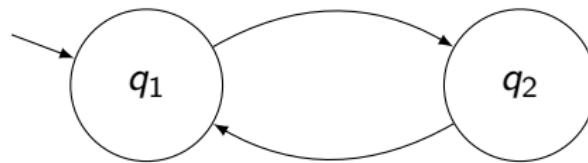
Structure de Kripke : $\mathcal{M} = (\mathcal{Q}, I, \rightarrow, \mathcal{L})$

$$\mathcal{Q} = \{q_1, q_2\}$$

$$I = \{q_1\}$$

Définir

- Modèle pour décrire les systèmes
- Système formel pour décrire les propriétés



Structure de Kripke : $\mathcal{M} = (\mathcal{Q}, \mathcal{I}, \rightarrow, \mathcal{L})$

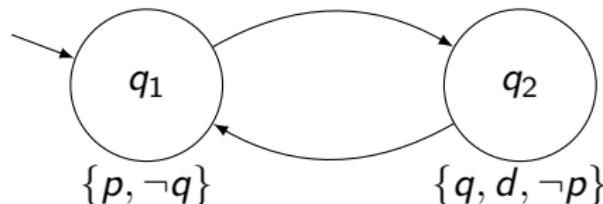
$$\mathcal{Q} = \{q_1, q_2\}$$

$$\mathcal{I} = \{q_1\}$$

$$\rightarrow = \{(q_1, q_2), (q_2, q_1)\}$$

Définir

- Modèle pour décrire les systèmes
- Système formel pour décrire les propriétés



Structure de Kripke : $\mathcal{M} = (\mathcal{Q}, I, \rightarrow, \mathcal{L})$

$$\mathcal{Q} = \{q_1, q_2\}$$

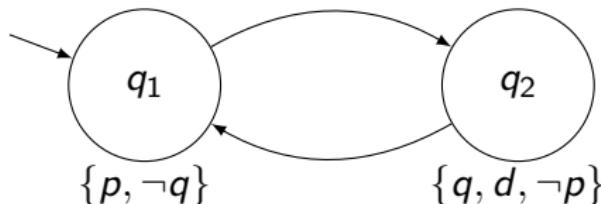
$$I = \{q_1\}$$

$$\rightarrow = \{(q_1, q_2), (q_2, q_1)\}$$

$$\mathcal{L} = \{(q_1, \{p, \neg q\}), (q_2, \{q, d, \neg p\})\}$$

Définir

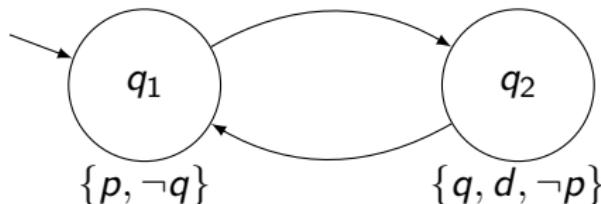
- Modèle pour décrire les systèmes
- Système formel pour décrire les propriétés



Chemin S d'étiquette L sur $\mathcal{M} = (\mathcal{Q}, \mathcal{I}, \rightarrow, \mathcal{L})$

Définir

- Modèle pour décrire les systèmes
- Système formel pour décrire les propriétés

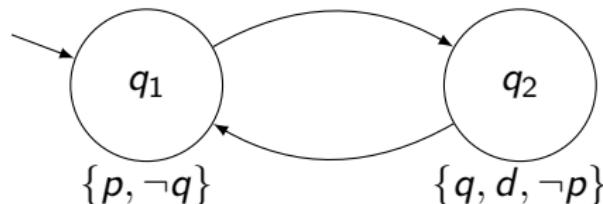


Chemin S d'étiquette L sur $\mathcal{M} = (\mathcal{Q}, I, \rightarrow, \mathcal{L})$

$S = (s_n)_{n \in \mathbb{N}} \in \mathcal{Q}^{\mathbb{N}}$ tel que $\forall i \in \mathbb{N}, (s_i, s_{i+1}) \in \rightarrow$

Définir

- Modèle pour décrire les systèmes
- Système formel pour décrire les propriétés



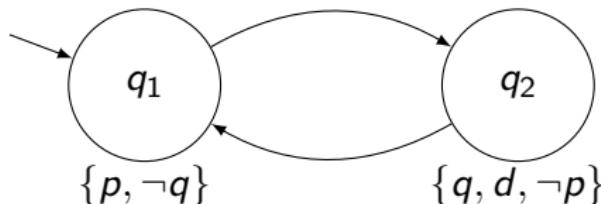
Chemin S d'étiquette L sur $\mathcal{M} = (\mathcal{Q}, I, \rightarrow, \mathcal{L})$

$S = (s_n)_{n \in \mathbb{N}} \in \mathcal{Q}^{\mathbb{N}}$ tel que $\forall i \in \mathbb{N}, (s_i, s_{i+1}) \in \rightarrow$

$L = \mathcal{L}(s_1) \bullet \mathcal{L}(s_2) \bullet \mathcal{L}(s_3) \dots$

Définir

- Modèle pour décrire les systèmes
- Système formel pour décrire les propriétés



Chemin S d'étiquette L sur $\mathcal{M} = (\mathcal{Q}, I, \rightarrow, \mathcal{L})$

$S = (s_n)_{n \in \mathbb{N}} \in \mathcal{Q}^{\mathbb{N}}$ tel que $\forall i \in \mathbb{N}, (s_i, s_{i+1}) \in \rightarrow$

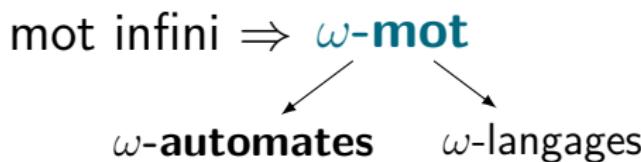
$L = \mathcal{L}(s_1) \bullet \mathcal{L}(s_2) \bullet \mathcal{L}(s_3) \dots$



mot infini

Définir

- Modèle pour décrire les systèmes
- Système formel pour décrire les propriétés



Définir

- Modèle pour décrire les systèmes
- Système formel pour décrire les propriétés

mot infini $\Rightarrow \omega\text{-mot}$



automates de Büchi généralisés étiquetés

$\mathcal{B} = (\mathcal{Q}, I, F, \rightarrow, L)$

exécution acceptante

$F = \{F_1, F_2, \dots, F_n\} \subseteq (\mathcal{P}(\mathcal{Q}))^n \quad \exists i, \text{ on passe par } F_i \text{ infiniment souvent}$

Définir

- Modèle pour décrire les systèmes
- Système formel pour décrire les propriétés

Définir

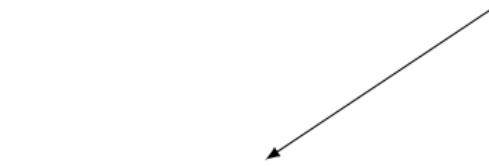
- Modèle pour décrire les systèmes
- Système formel pour décrire les propriétés

Linear Temporal Logic (LTL)

Définir

- Modèle pour décrire les systèmes
- Système formel pour décrire les propriétés

Linear Temporal Logic (LTL)



Logique propositionnelle

\perp T $\neg\varphi$ $\varphi_1 \wedge \varphi_2$

p $\varphi_1 \vee \varphi_2$

Définir

- Modèle pour décrire les systèmes
- Système formel pour décrire les propriétés

Linear Temporal Logic (LTL)

Logique propositionnelle + Opérateurs temporels

\perp T $\neg\varphi$ $\varphi_1 \wedge \varphi_2$

p $\varphi_1 \vee \varphi_2$

$X\varphi$ $F\varphi$ $G\varphi$ $\varphi_1 U \varphi_2$

$\varphi_1 R \varphi_2$

Définir

- Modèle pour décrire les systèmes
- Système formel pour décrire les propriétés

Linear Temporal Logic (LTL)

Logique propositionnelle + Opérateurs temporels

\perp T $\neg\varphi$ $\varphi_1 \wedge \varphi_2$

p $\varphi_1 \vee \varphi_2$

$X\varphi$ $F\varphi$ $G\varphi$ $\varphi_1 U \varphi_2$

$\varphi_1 R \varphi_2$

Définir

- Modèle pour décrire les systèmes
- Système formel pour décrire les propriétés

Linear Temporal Logic (LTL)

Logique propositionnelle + Opérateurs temporels

\perp T $\neg\varphi$ $\varphi_1 \wedge \varphi_2$

p $\varphi_1 \vee \varphi_2$

$X\varphi$ $F\varphi$ $G\varphi$ $\varphi_1 U \varphi_2$

$\varphi_1 R \varphi_2$

Définir

- Modèle pour décrire les systèmes
- Système formel pour décrire les propriétés

Linear Temporal Logic (LTL)

Logique propositionnelle + Opérateurs temporels

\perp T $\neg\varphi$ $\varphi_1 \wedge \varphi_2$

p $\varphi_1 \vee \varphi_2$

$X\varphi$ $F\varphi$ $G\varphi$ $\varphi_1 U \varphi_2$

$\varphi_1 R \varphi_2$

Définir

- Modèle pour décrire les systèmes
- Système formel pour décrire les propriétés

Linear Temporal Logic (LTL)

Logique propositionnelle + Opérateurs temporels

\perp T $\neg\varphi$ $\varphi_1 \wedge \varphi_2$

p $\varphi_1 \vee \varphi_2$

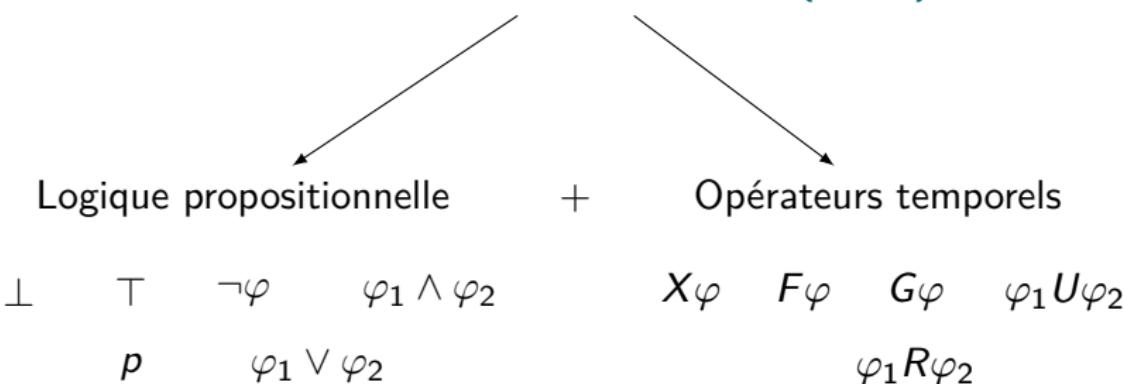
$X\varphi$ $F\varphi$ $G\varphi$ $\varphi_1 U \varphi_2$

$\varphi_1 R \varphi_2$

Définir

- Modèle pour décrire les systèmes
- Système formel pour décrire les propriétés

Linear Temporal Logic (LTL)



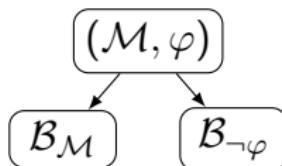
Tester

But du Model-Checking : \mathcal{M} satisfait φ ?

$$(\mathcal{M}, \varphi)$$

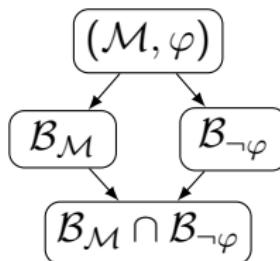
Tester

But du Model-Checking : \mathcal{M} satisfait φ ?



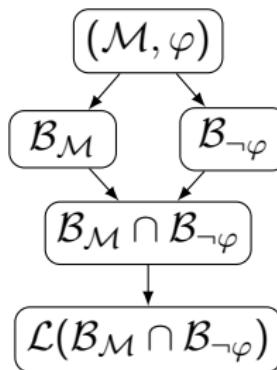
Tester

But du Model-Checking : \mathcal{M} satisfait φ ?



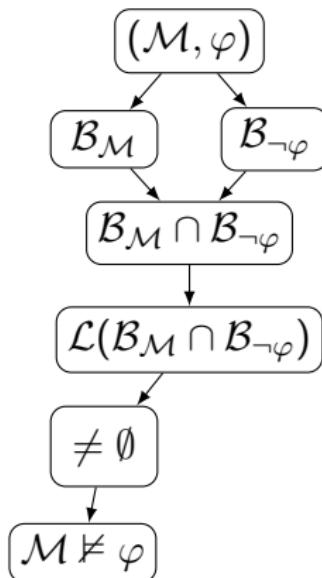
Tester

But du Model-Checking : \mathcal{M} satisfait φ ?



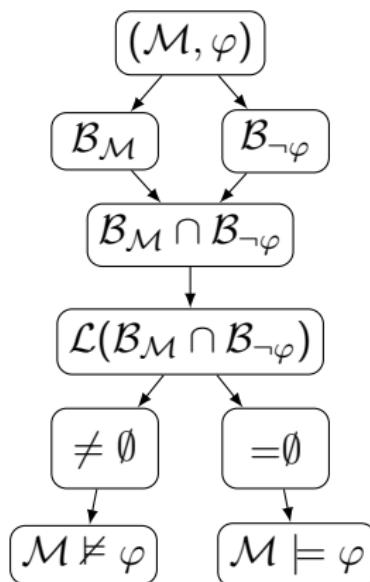
Tester

But du Model-Checking : \mathcal{M} satisfait φ ?



Tester

But du Model-Checking : \mathcal{M} satisfait φ ?



1. Définitions

2. Implémentation LTL

LTL en ocaml

Mettre une formule sous forme normale négative

Notions d'ensembles en Ocaml

Transformer f en LGBA

3. Faire l'intersection entre deux automates de Büchi

4. Annexe

Définition formelle de la logique LTL (linear temporal logic)

LTL :

- ▶ \top
- ▶ \perp
- ▶ $p, p \in AP$
- ▶ $\neg \phi, \phi \in LTL$
- ▶ $\phi \wedge \psi, (\phi, \psi) \in LTL \times LTL$
- ▶ $\phi \vee \psi, (\phi, \psi) \in LTL \times LTL$
- ▶ $X \phi, \phi \in LTL$ (**N**ext)
- ▶ $G \phi, \phi \in LTL$ (**G**lobally)
- ▶ $F \phi, \phi \in LTL$ (**F**inally)
- ▶ $\phi U \psi, (\phi, \psi) \in LTL \times LTL$ (**U**ntil)
- ▶ $\phi R \psi, (\phi, \psi) \in LTL \times LTL$ (**R**elease)

```

1  type ltl =
2  | Top
3  | Bot
4  | Atom of string
5  | Not of ltl
6  | Or of ltl*ltl
7  | And of ltl*ltl
8  | X of ltl
9  | G of ltl
10 | F of ltl
11 | U of ltl*ltl
12 | R of ltl*ltl

```

Définition d'une fnn (forme normale négative)

Definition (Forme normale négative pour la LTL)

Une formule LTL est dite sous forme normale négative lorsque :

- L'opérateur \neg est appliqué uniquement aux propositions atomiques
- Seules les opérateurs : \vee , \wedge , X , U , R peuvent apparaître devant les littéraux (On abandonne G , F)

Les règles utilisées

| Loi de De Morgan | |
|---------------------------|----------------------------------|
| Formule initiale | Formule équivalente |
| $\neg (\phi \wedge \psi)$ | $(\neg \phi) \vee (\neg \psi)$ |
| $\neg (\phi \vee \psi)$ | $(\neg \phi) \wedge (\neg \psi)$ |

Les règles utilisées

| Loi de De Morgan | |
|---------------------------|----------------------------------|
| Formule initiale | Formule équivalente |
| $\neg (\phi \wedge \psi)$ | $(\neg \phi) \vee (\neg \psi)$ |
| $\neg (\phi \vee \psi)$ | $(\neg \phi) \wedge (\neg \psi)$ |

| Pour "pousser" la négation | |
|----------------------------|-----------------------------|
| Formule initiale | Formule équivalente |
| $\neg (X \phi)$ | $(X \neg \phi)$ |
| $\neg (\phi U \psi)$ | $(\neg \phi) R (\neg \psi)$ |
| $\neg (\phi R \psi)$ | $(\neg \phi) U (\neg \psi)$ |

Les règles utilisées

| Loi de De Morgan | |
|---------------------------|----------------------------------|
| Formule initiale | Formule équivalente |
| $\neg (\phi \wedge \psi)$ | $(\neg \phi) \vee (\neg \psi)$ |
| $\neg (\phi \vee \psi)$ | $(\neg \phi) \wedge (\neg \psi)$ |

| Pour "pousser" la négation | |
|----------------------------|-----------------------------|
| Formule initiale | Formule équivalente |
| $\neg (X \phi)$ | $(X \neg \phi)$ |
| $\neg (\phi U \psi)$ | $(\neg \phi) R (\neg \psi)$ |
| $\neg (\phi R \psi)$ | $(\neg \phi) U (\neg \psi)$ |

| Pour supprimer F et G | |
|-----------------------|---------------------|
| Formule initiale | Formule équivalente |
| $F \phi$ | $\top U \phi$ |
| $G \phi$ | $\perp R \phi$ |

Les règles utilisées

| Loi de De Morgan | |
|--------------------------|----------------------------------|
| Formule initiale | Formule équivalente |
| $\neg(\phi \wedge \psi)$ | $(\neg \phi) \vee (\neg \psi)$ |
| $\neg(\phi \vee \psi)$ | $(\neg \phi) \wedge (\neg \psi)$ |

| Pour "pousser" la négation | |
|----------------------------|-----------------------------|
| Formule initiale | Formule équivalente |
| $\neg(X \phi)$ | $(X \neg \phi)$ |
| $\neg(\phi U \psi)$ | $(\neg \phi) R (\neg \psi)$ |
| $\neg(\phi R \psi)$ | $(\neg \phi) U (\neg \psi)$ |

| Pour supprimer F et G | |
|-----------------------|---------------------|
| Formule initiale | Formule équivalente |
| $F \phi$ | $\top U \phi$ |
| $G \phi$ | $\perp R \phi$ |

| Pour transformer une formule en LGBA | |
|--------------------------------------|--|
| Formule initiale | Formule équivalente |
| $\phi U \psi$ | $\psi \vee (\phi \wedge X(\phi U \psi))$ |
| $\phi R \psi$ | $(\psi \wedge \phi) \vee (\psi \wedge X(\phi R \psi))$ |

Exemple d'une transformation vers fnn

$$f := F(p \wedge (X q))$$

Exemple d'une transformation vers fnn

$$f := F(p \wedge (X q))$$

$$\neg(F(p \wedge (X q)))$$

Exemple d'une transformation vers fnn

$$f := F(p \wedge (X q))$$

$$\neg(F(p \wedge (X q)))$$



$$\neg(\top U (p \wedge (X q)))$$

Exemple d'une transformation vers fnn

$$f := F(p \wedge (X q))$$

$$\neg(F(p \wedge (X q)))$$



$$\neg(\top U (p \wedge (X q)))$$



$$(\neg\top R \neg(p \wedge (X q)))$$

Exemple d'une transformation vers fnn

$$f := F(p \wedge (X q))$$

$$\neg(F(p \wedge (X q)))$$



$$\neg(\top U (p \wedge (X q)))$$



$$(\neg\top R \neg(p \wedge (X q)))$$

$$\neg\top$$

Exemple d'une transformation vers fnn

$$f := F(p \wedge (X q))$$

$$\neg(F(p \wedge (X q)))$$



$$\neg(\top U (p \wedge (X q)))$$



$$(\neg\top R \neg(p \wedge (X q)))$$

$$\neg\top$$



$$\perp$$

Exemple d'une transformation vers fnn

$$f := F(p \wedge (X q))$$

$$\neg(F(p \wedge (X q)))$$



$$\neg(\top U (p \wedge (X q)))$$



$$(\neg\top R \neg(p \wedge (X q)))$$



Exemple d'une transformation vers fnn

$$f := F(p \wedge (X q))$$

$$\neg(F(p \wedge (X q)))$$



$$\neg(\top U (p \wedge (X q)))$$



$$(\neg\top R \neg(p \wedge (X q)))$$

$$\begin{array}{ccc} \neg\top & \xleftarrow{\quad} & R \\ \downarrow & & \downarrow \\ \perp & & \neg(p \wedge (X q)) \end{array}$$

Exemple d'une transformation vers fnn

$$f := F(p \wedge (X q))$$

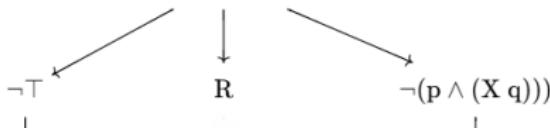
$$\neg(F(p \wedge (X q)))$$



$$\neg(\top U (p \wedge (X q)))$$



$$(\neg\top R \neg(p \wedge (X q)))$$



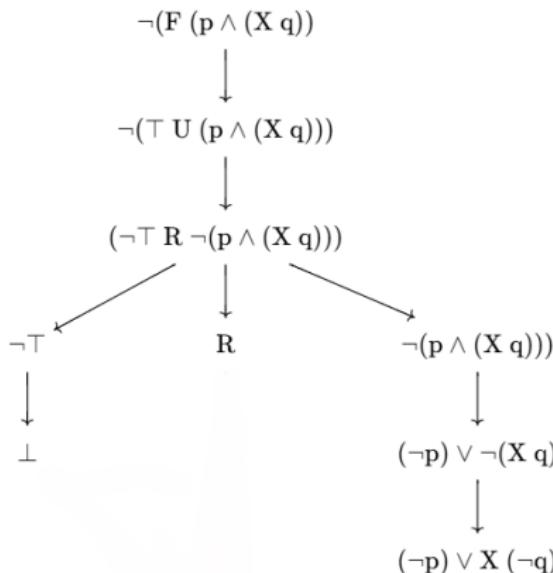
$$\perp$$



$$(\neg p) \vee \neg(X q)$$

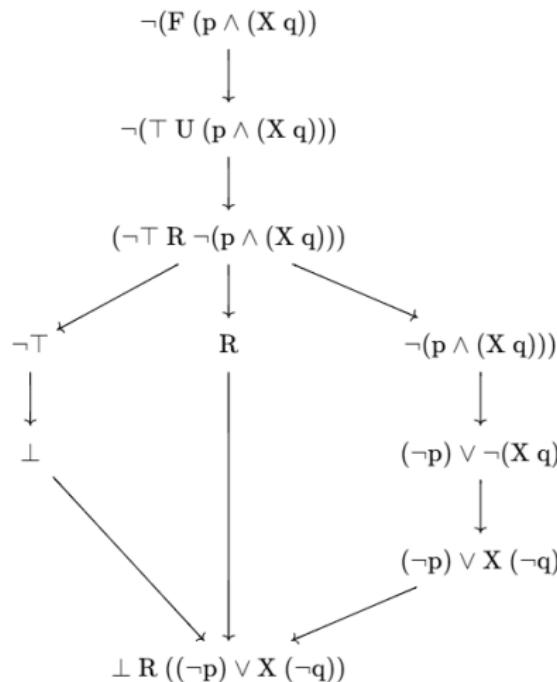
Exemple d'une transformation vers fnn

$$f := F(p \wedge (X q))$$



Exemple d'une transformation vers fnn

$$f := F(p \wedge (X q))$$

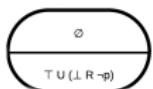


Exemple d'une transformation vers fnn

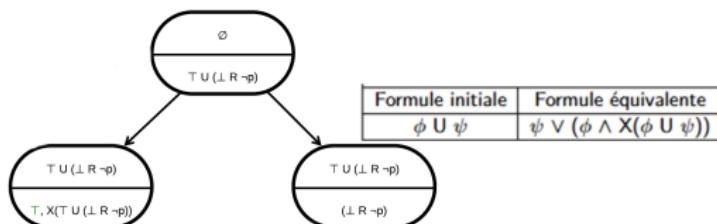
$$f := F(p \wedge (X q))$$

$$\boxed{\text{fnn } (\neg f) = \perp R (\neg p \vee (X \neg q))}$$

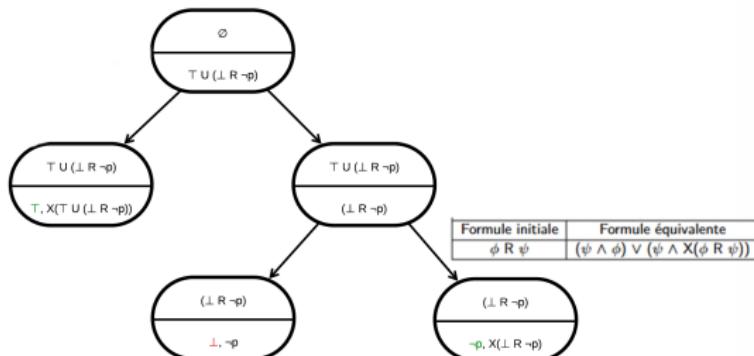
Construction du LGBA de $\neg G(F p)$ ($= \top \cup (\perp R \neg p)$)

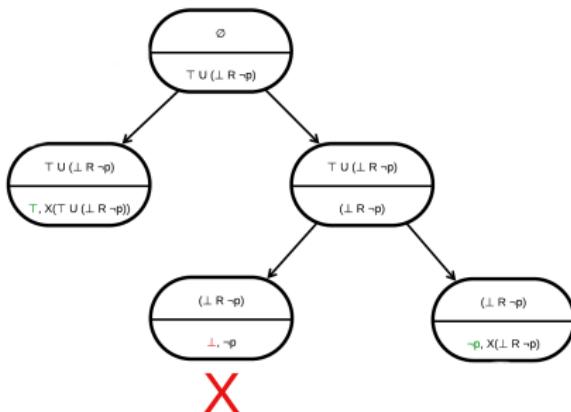


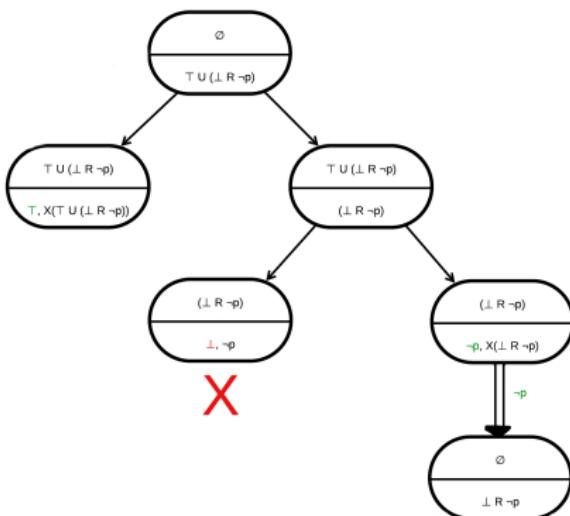
Construction du LGBA de $\neg G(F p)$ ($= \top \cup (\perp R \neg p)$)



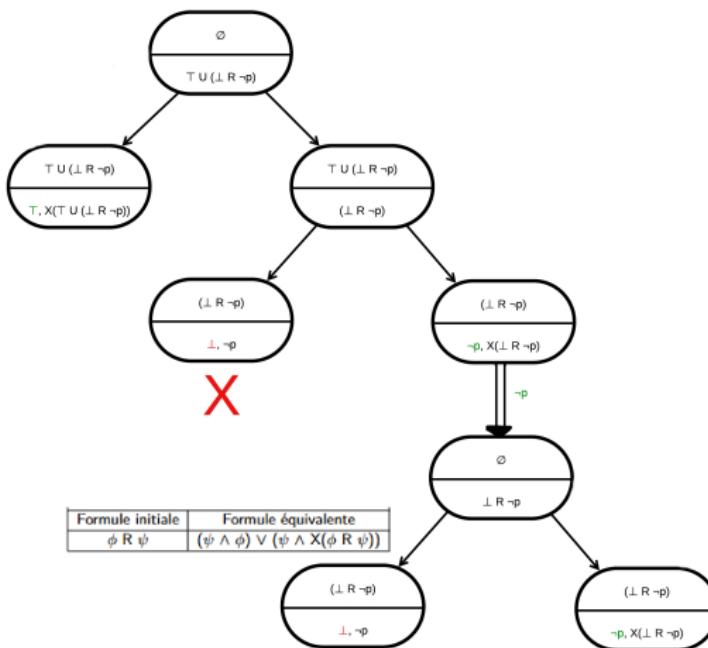
Construction du LGBA de $\neg G(F p)$ ($= \top \cup (\perp R \neg p)$)



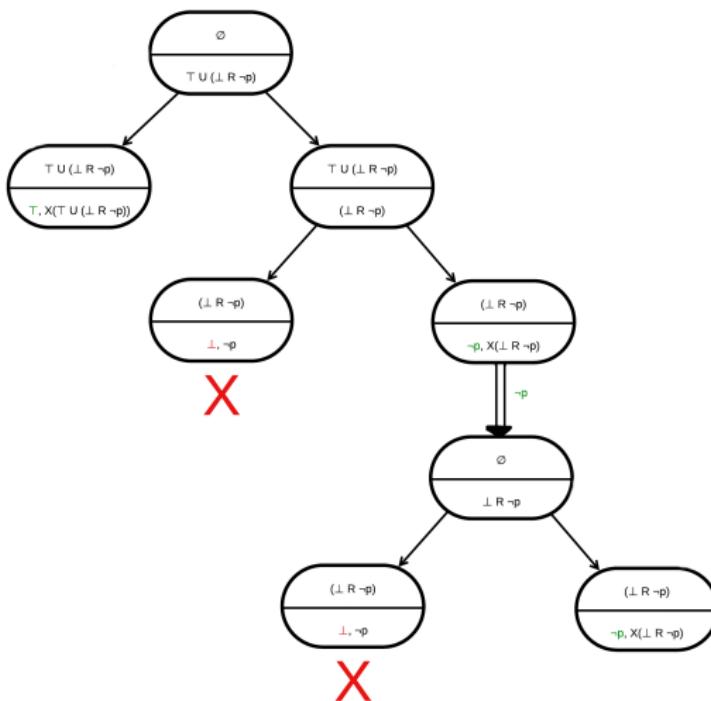
Construction du LGBA de $\neg G(F p) (= \top U (\perp R \neg p))$ 

Construction du LGBA de $\neg G(F p) (= \top U (\perp R \neg p))$ 

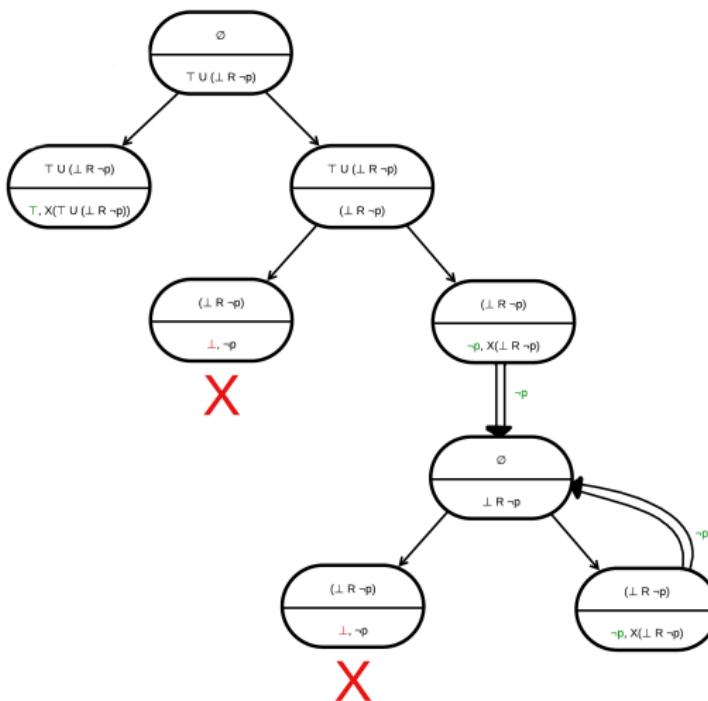
Construction du LGBA de $\neg G(F p) (= \top \cup (\perp R \neg p))$



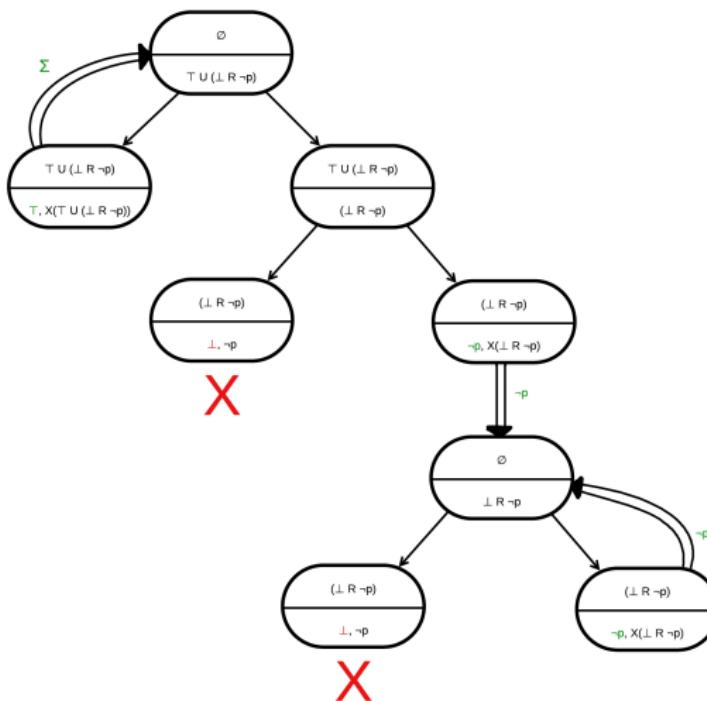
Construction du LGBA de $\neg G(F p) (= \top U (\perp R \neg p))$



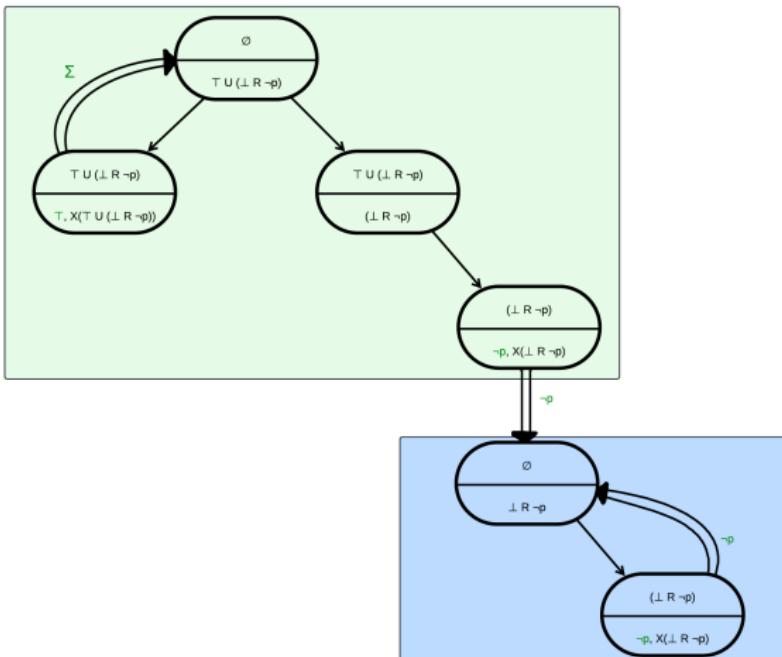
Construction du LGBA de $\neg G(F p) (= \top U (\perp R \neg p))$



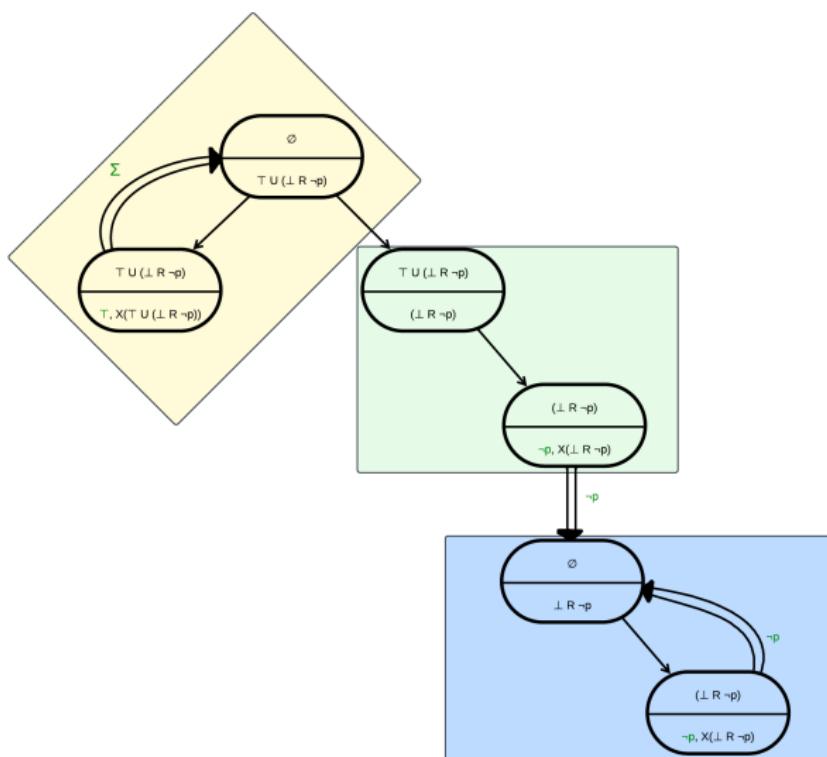
Construction du LGBA de $\neg G(F p) (= \top \cup (\perp R \neg p))$



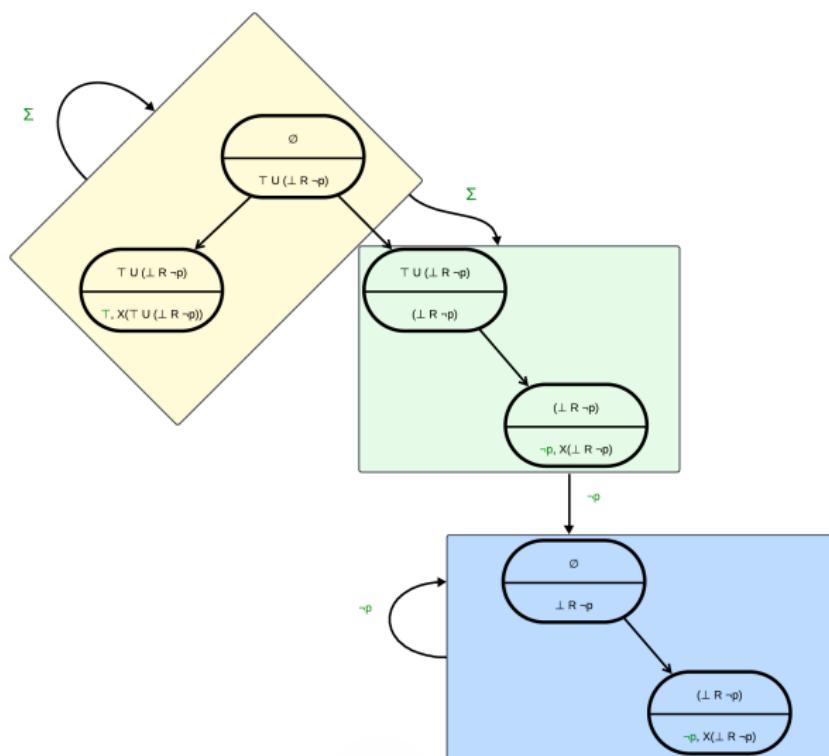
Construction du LGBA de $\neg G(F p) (= \top \cup (\perp R \neg p))$



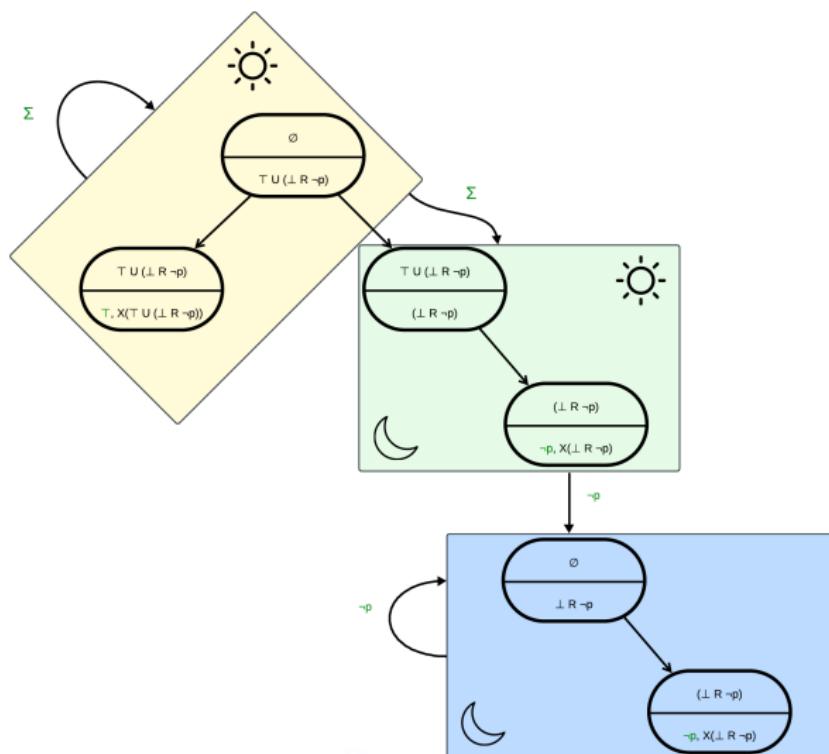
Construction du LGBA de $\neg G(F p) (= \top \cup (\perp R \neg p))$

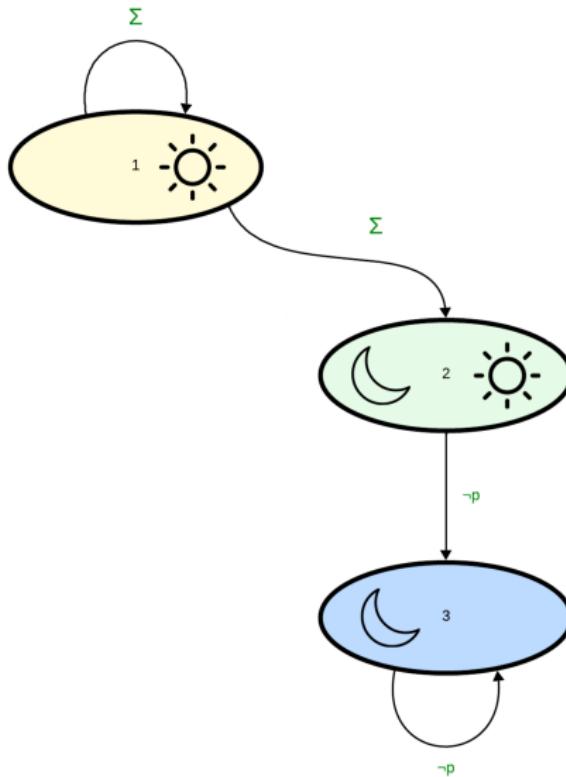


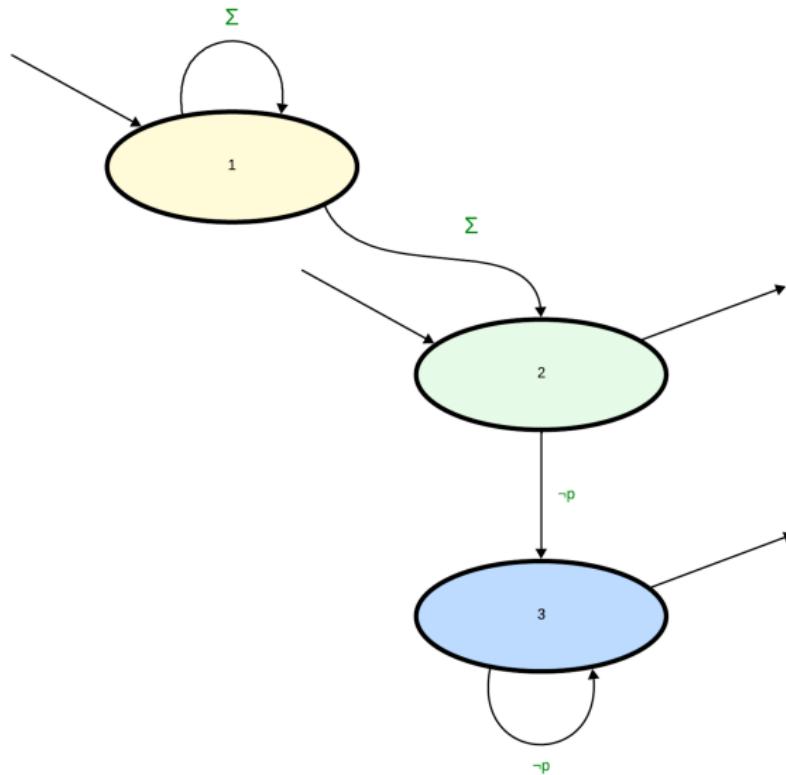
Construction du LGBA de $\neg G(F p) (= \top \cup (\perp R \neg p))$



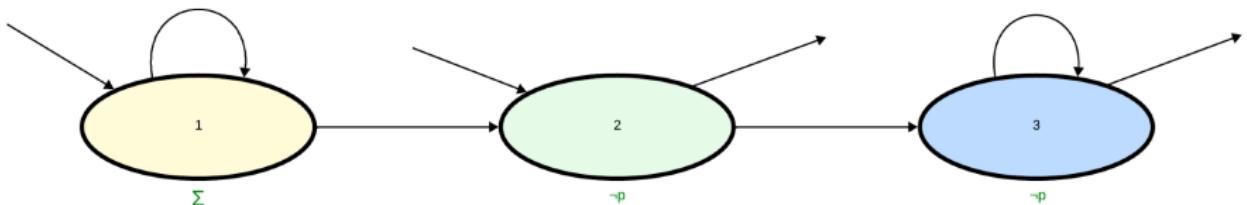
Construction du LGBA de $\neg G(F p) (= \top \cup (\perp R \neg p))$



Construction du LGBA de $\neg G(F p)$ ($= \top \cup (\perp R \neg p)$)

Construction du LGBA de $\neg G(F p)$ ($= \top \cup (\perp R \neg p)$)

Construction du LGBA de $\neg G(F p) (= \top \cup (\perp R \neg p))$



1. Définitions

2. Implémentation LTL

3. Faire l'intersection entre deux automates de Büchi

Passer de structure de Kripke à LGBA

Intersection de deux LGBA

Test de vacuité

4. Annexe

Transformation d'une structure de Kripke vers un LGBA

Soit Σ un alphabet.

Soit $K = (Q, I, \rightarrow, L)$ une structure de Kripke.

On pose $A = (\Sigma, Q', I', F, \delta, L')$ un labelled generalized Büchi automaton (LGBA) où

$$\rightarrow Q' = Q$$

$$\rightarrow I' = I$$

$$\rightarrow F = Q$$

$$\rightarrow \delta = \rightarrow$$

$$\rightarrow L' = L$$

Intersection de deux LGBA

Definition

Soit $A_1 = (\Sigma, Q_1, I_1, F_1, \delta_1, L_1)$ et
 $A_2 = (\Sigma, Q_2, I_2, F_2, \delta_2, L_2)$ deux LGBA.

On pose A_p l'automate produit $(\Sigma, Q, I, F, \delta, L)$ où

- ▶ $Q = Q_1 \times Q_2$
- ▶ $I = I_1 \times I_2$
- ▶ $F = F_1 \times F_2$
- ▶ $\delta = \{((q, q'), (s, s')) \in Q \times Q / (q, s) \in \delta_1 \text{ et } (q', s') \in \delta_2 \text{ et } (L_1(q) \cap L_2(s)) \neq \emptyset\}$
- ▶ $L = \{((q, s), (L_1(q) \cap L_2(s)))\}$

Exemple

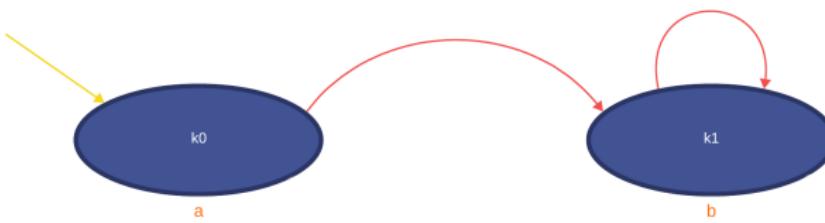


Figure – Structure de Kripke vérifiant $G(F b)$

Exemple

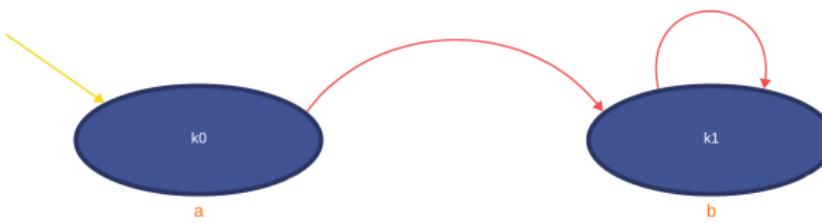


Figure – Structure de Kripke vérifiant $G(F b)$

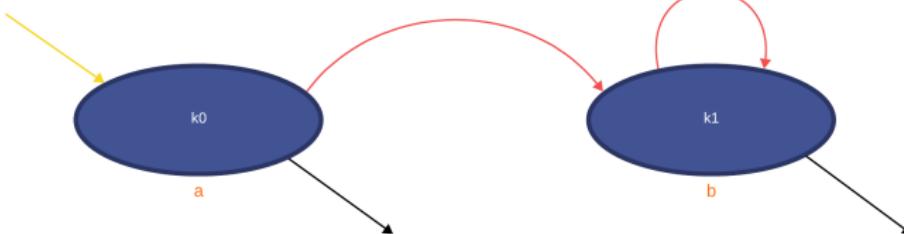


Figure – LGBA associé à la structure de Kripke

Exemple

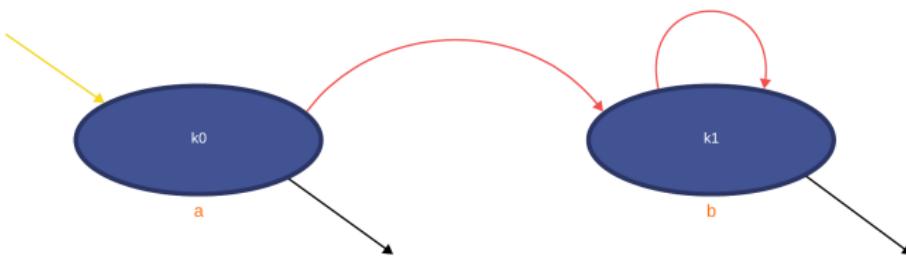


Figure – LGBA associé à la structure de Kripke

Exemple

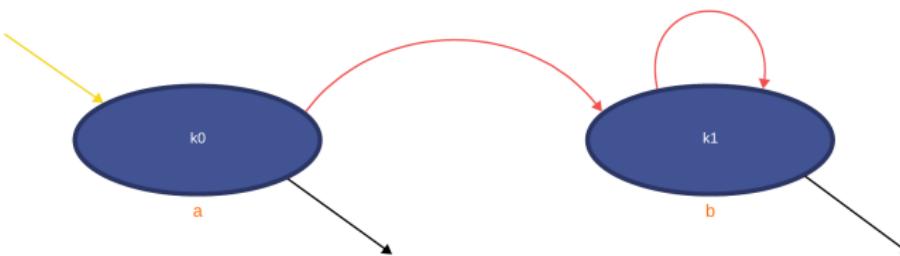


Figure – LGBA associé à la structure de Kripke

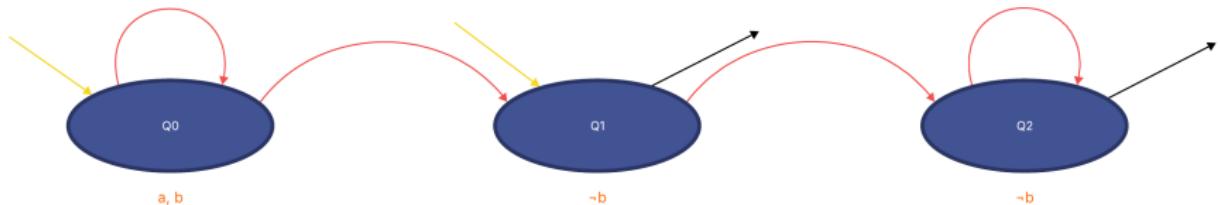


Figure – LGBA associé à la formule $\neg G(F b)$

Exemple

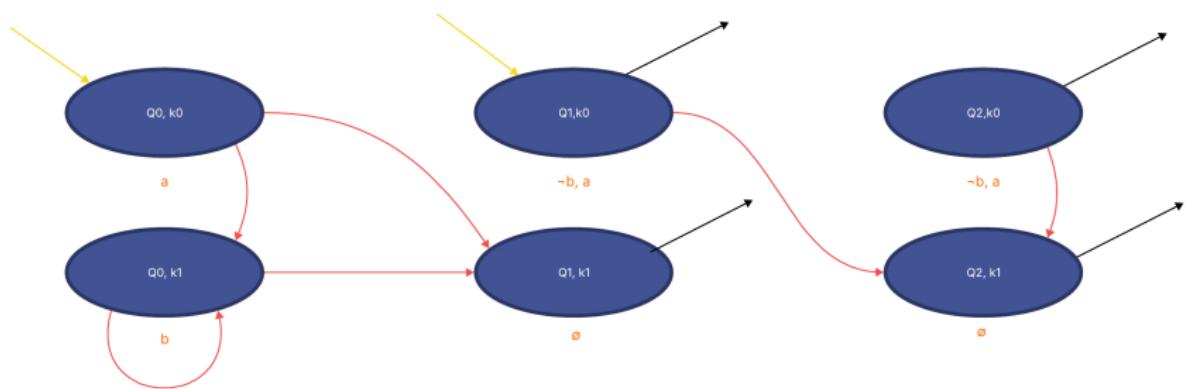


Figure – Intersection des deux LGBA

Test de vacuité du langage d'un LGBA (Algo 1)

Algorithme 1 : outerdfs (Recherche externe de cycle)

Entrée : état q , liste *visités*, automate $A = (\Sigma, Q, I, F, \delta, L)$

Sortie : Recherche de cycle acceptant depuis q

```
1 visités' ← ajouter  $q$  à visités
2 pour tout  $q' \in \delta(q)$  faire
3   si  $q' \notin$  visités' alors
4     // Appel récursif
5     outerdfs( $q'$ , visités', A)
6   si  $q \in F$  alors
7     innerdfs( $q$ , visités', [ ], A)
```

Test de vacuité du langage d'un LGBA (Algo 2)

Algorithme 2 : innerdfs (Recherche interne de cycle)

Entrée : état q , liste visitesExt , liste visitesIn , automate

$$A = (\Sigma, Q, I, F, \delta, L)$$

Sortie : Recherche d'un cycle vers un état visité par outerdfs

- ```

1 visitesIn' ← ajouter q à visitesIn
2 pour tout $q' \in \delta(q)$ faire
3 si $q' \in$ visitesExt alors
4 lancer exception Found // Cycle détecté
5 sinon si $q' \notin$ visitesIn' alors
6 appeler innerdfs(q' , visitesExt, visitesIn', A)

```

# Test de vacuité du langage d'un LGBA (Algo 3)

---

**Algorithm 3** : estVide (Test de vacuité)

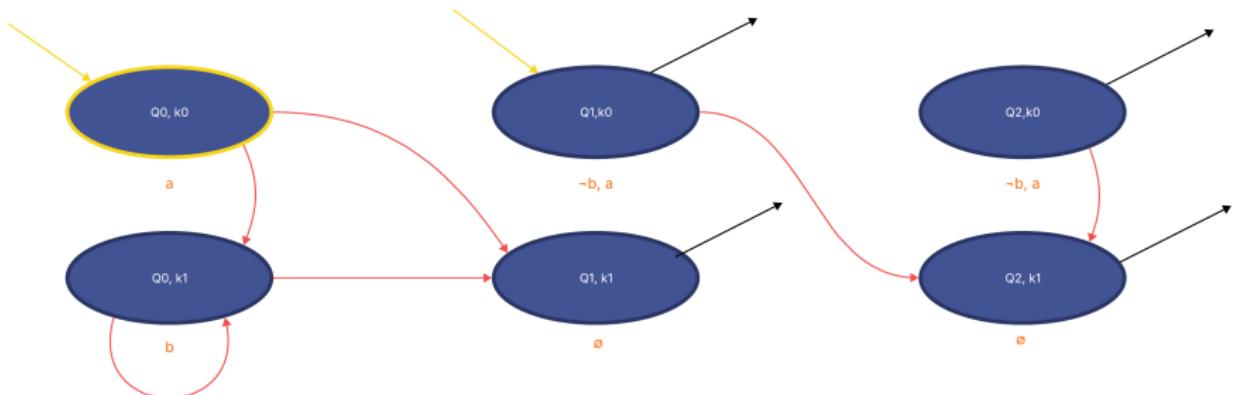
---

**Entrée** : Automate  $A = (\Sigma, Q, I, F, \delta, L)$

**Sortie** : Vrai si un cycle acceptant est trouvé, Faux sinon

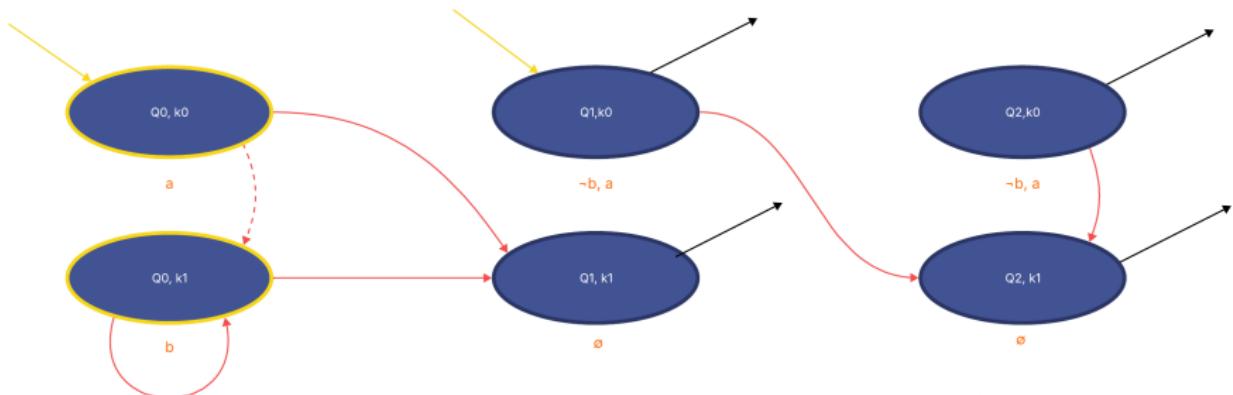
- 1 **pour tout**  $q_0 \in I$  **faire**
  - 2   | **essayer** outerdfs( $q_0, [ ]$ ,  $A$ ) **avec**
  - 3   |   | Found  $\rightarrow$  **retourner** Faux
  - 4 **retourner** Vrai
-

# Exemple d'exécution



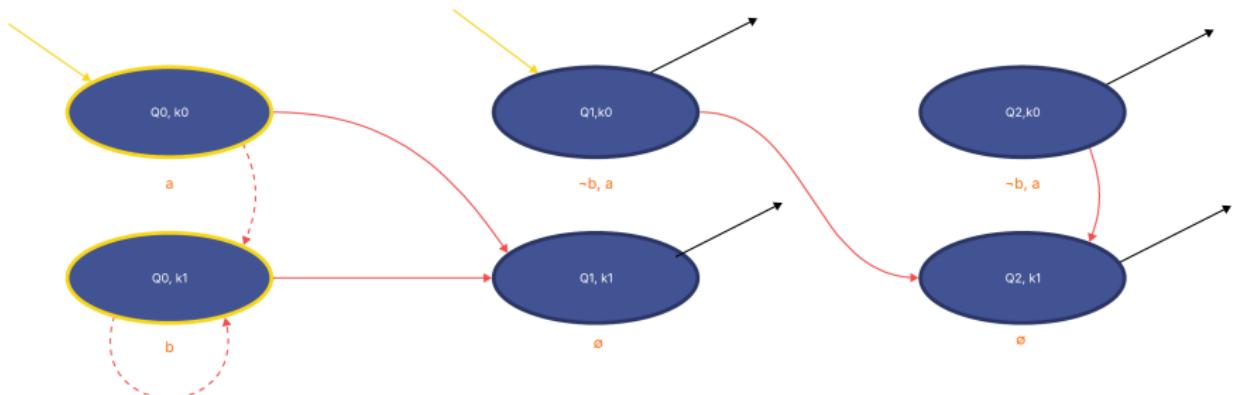
outerdfs Visités : [(Q<sub>0</sub>, k<sub>0</sub>)]

# Exemple d'exécution



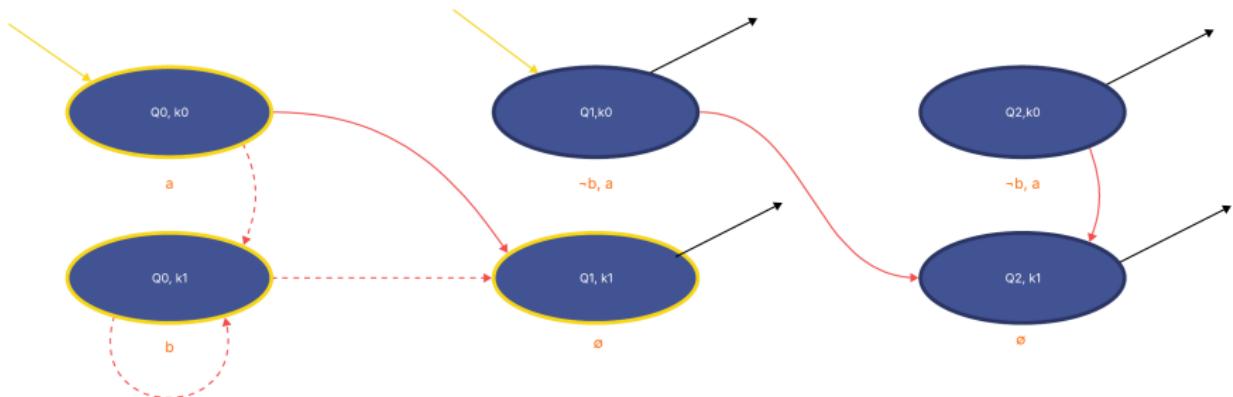
outerdfs Visités :  $[(Q0, k0), (Q0, k1)]$

# Exemple d'exécution



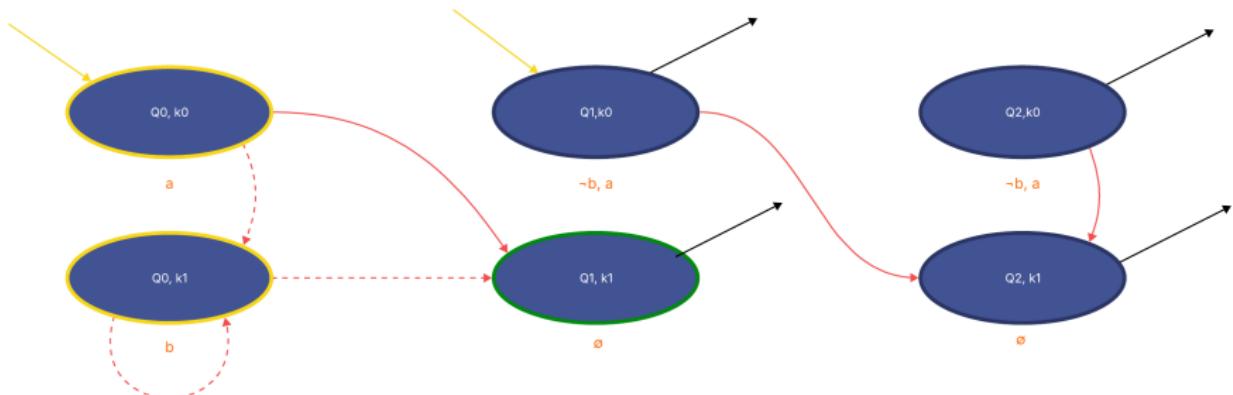
outerdfs Visités :  $[(Q0, k0), (Q0, k1)]$

# Exemple d'exécution



outerdfs Visités :  $[(Q0, k0), (Q0, k1), (Q1, k1)]$

# Exemple d'exécution

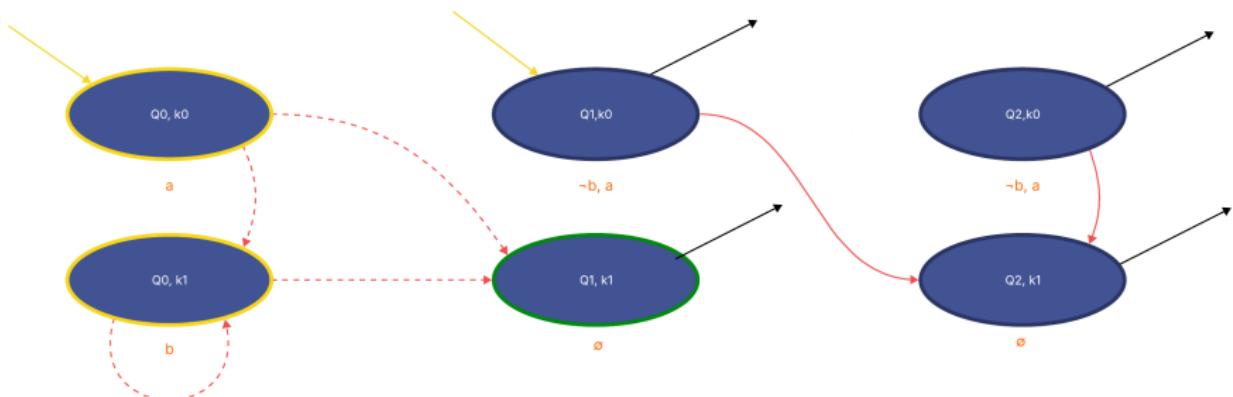


outerdfs Visités :  $[(Q0, k0), (Q0, k1), (Q1, k1)]$

innerdfs1 VisitésIn1 :  $[(Q1, k1)]$

innerdfs1 VisitésExt1 :  $[(Q0, k0), (Q0, k1), (Q1, k1)]$

# Exemple d'exécution

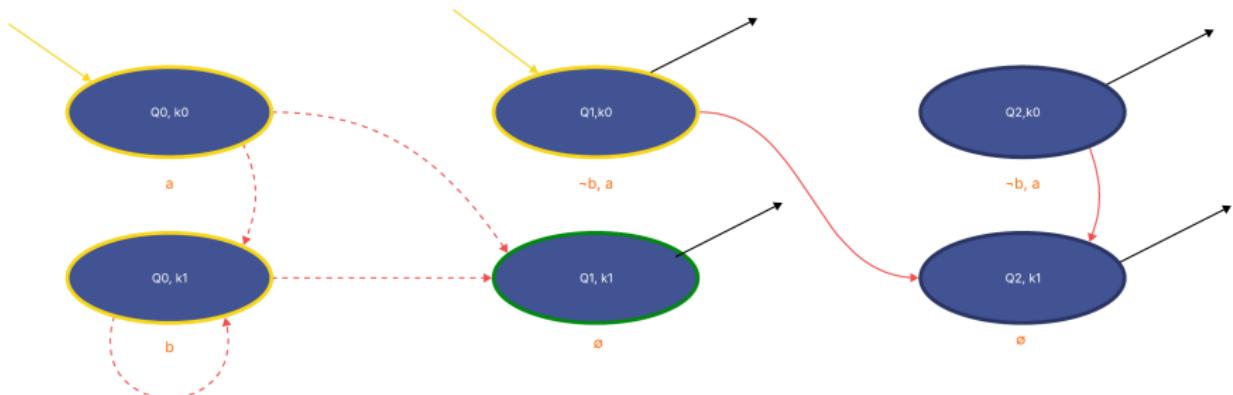


**outerdfs** Visités :  $[(Q0, k0), (Q0, k1), (Q1, k1)]$

**innerdfs1** VisitésIn1 :  $[(Q1, k1)]$

**innerdfs1** VisitésExt1 :  $[(Q0, k0), (Q0, k1), (Q1, k1)]$

# Exemple d'exécution

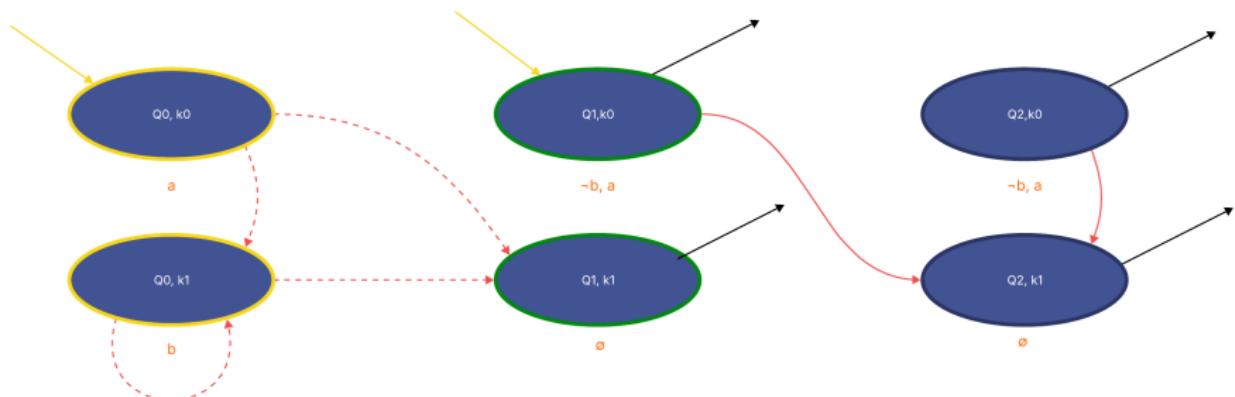


outerdfs Visités :  $[(Q_0, k_0), (Q_0, k_1), (Q_1, k_1), (Q_1, k_0)]$

innerdfs1 VisitésIn1 :  $[(Q_1, k_1)]$

innerdfs1 VisitésExt1 :  $[(Q_0, k_0), (Q_0, k_1), (Q_1, k_1)]$

# Exemple d'exécution



**outerdfs** Visités :  $[(Q0, k0), (Q0, k1), (Q1, k1), (Q1, k0)]$

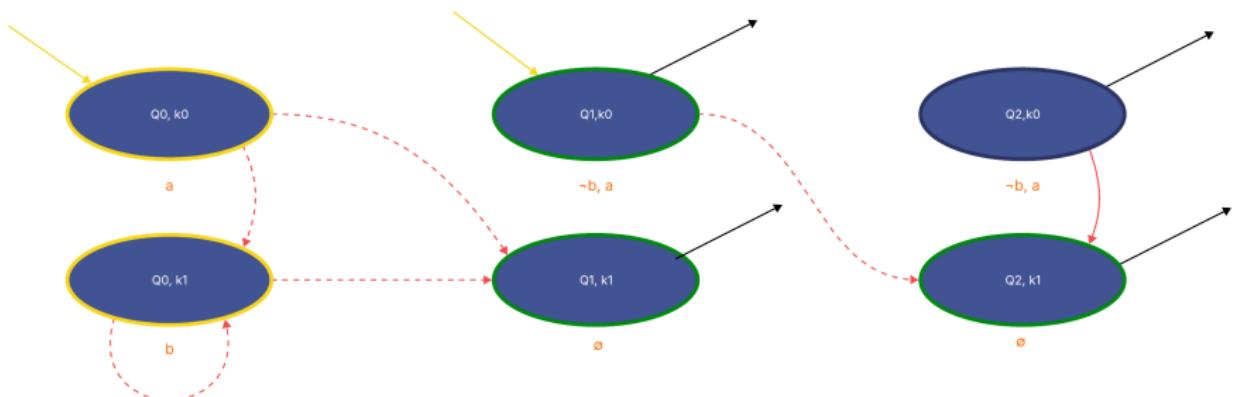
**innerdfs1** VisitésIn1 :  $[(Q1, k1)]$

**innerdfs1** VisitésExt1 :  $[(Q0, k0), (Q0, k1), (Q1, k1)]$

**innerdfs2** VisitésIn2 :  $[(Q1, k0)]$

**innerdfs2** VisitésExt2 :  $[(Q0, k0), (Q0, k1), (Q1, k1), (Q1, k0)]$

# Exemple d'exécution



**outerdfs** Visités :  $[(Q0, k0), (Q0, k1), (Q1, k1), (Q1, k0)]$

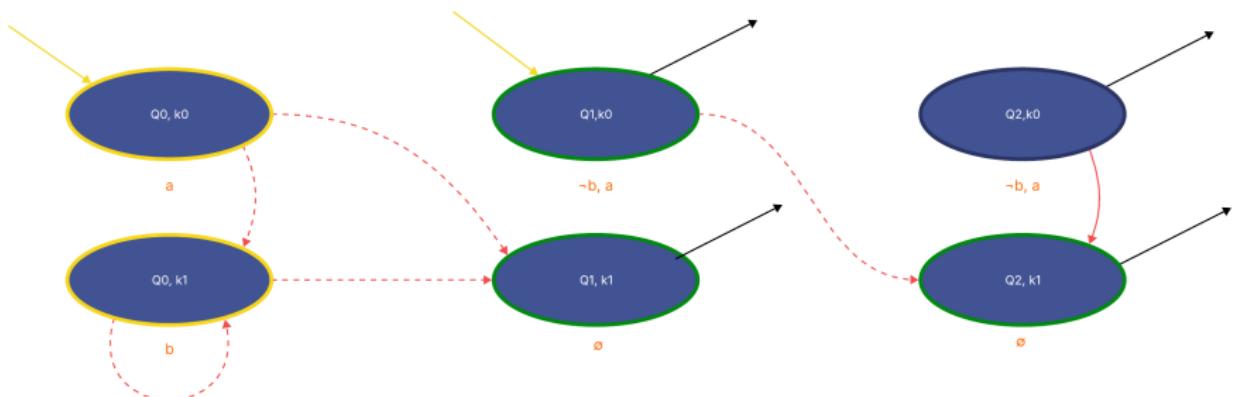
**innerdfs1** VisitésIn1 :  $[(Q1, k1)]$

**innerdfs1** VisitésExt1 :  $[(Q0, k0), (Q0, k1), (Q1, k1)]$

**innerdfs2** VisitésIn2 :  $[(Q1, k0), (Q2, k1)]$

**innerdfs2** VisitésExt2 :  $[(Q0, k0), (Q0, k1), (Q1, k1), (Q1, k0)]$

# Exemple d'exécution



**outerdfs** Visités :  $[(Q0, k0), (Q0, k1), (Q1, k1), (Q1, k0)]$

**innerdfs1** VisitésIn1 :  $[(Q1, k1)]$

**innerdfs1** VisitésExt1 :  $[(Q0, k0), (Q0, k1), (Q1, k1)]$

**innerdfs2** VisitésIn2 :  $[(Q1, k0), (Q2, k1)]$

**innerdfs2** VisitésExt2 :  $[(Q0, k0), (Q0, k1), (Q1, k1), (Q1, k0)]$

Estvide  $\rightarrow$  Vrai

# Conclusion

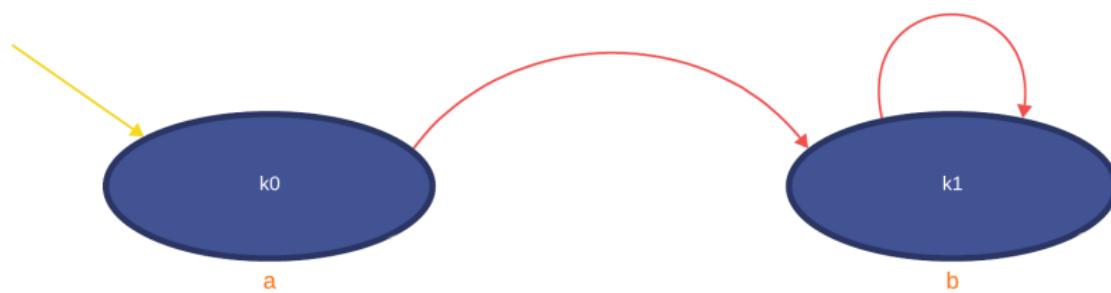


Figure – Structure de Kripke vérifiant  $G(F b)$

1. Définitions
2. Implémentation LTL
3. Faire l'intersection entre deux automates de Büchi
4. Annexe  
Code

## 4- Annexe • 4.1 Code

# Code du TIPE

VALENTIN TOMAS-TORTONI (12291)

Juin 2025

---

## Model Checker en LTL

### *Code du TIPE*

---

#### 1 Code de la première partie

##### 1.1 Code du fichier types.ml

```

1 type ltl =
2 | Top
3 | Bot
4 | Atom of string
5 | Not of ltl
6 | Or of ltl*ltl
7 | And of ltl*ltl
8 | X of ltl
9 | G of ltl
10 | F of ltl
11 | U of ltl*ltl
12 | R of ltl*ltl
13
14 type noword = {name : string}
15
16
17 type state = {stat : string}
18
19 type kripke_struct = {
20 state : state list;
21 init : state list;
22 transition : (state * state) list;
23 valuation : (state * (ltl list)) list;
24 (* une fonction qui à un état associe un ensemble de formules ltl*)
25 (* Cf définition dans le diapo *)
26 }
27
28 type buchi_automate = {
29 q : state list;
30 init : state list;
31 final : state list;
32 transition : (state * state) list;
33 valuation : (state * (ltl list)) list
34 (* La valuation correspond au "L" dans la définition du LGBA (pour labelled
35 -- generalized buchi automaton) *)
36 }
37

```

##### 1.2 Code du fichier ensembles.ml

```

1 (* On va maintenant coder un système d'ensemble avec des listes *)
2 let rec priv (a: 'a list) (b: 'a list) : 'a list =
3 (* Résulte d'UB (sans doublons si a et b sont sans doublons)*)
4 match a with
5 | [] -> []
6 | x::q -> if (List.mem x b) then priv q b
7 else x::(priv q b)
8
9 let rec union (a: 'a list) (b: 'a list) : 'a list =
10 (* Réalise A ∪ B (sans doublons si a et b sont sans doublons)*)
11 match a with
12 | [] -> b
13 | x::q -> if (List.mem x b) then union q b
14 else x::(union q b)
15
16 let rec inter (a: 'a list) (b: 'a list) : 'a list =
17 (* Réalise a ∩ b *)
18 match a with
19 | [] -> []
20 | x::q -> if (List.mem x b) then x::(inter q b)
21 else inter q b
22
23 let appartient (x: 'a) (a: 'a list) :bool =
24 List.mem x a
25
26 let inclus (a: 'a list) (b: 'a list) : bool =
27 (* Test si a ⊂ inclus dans B *)
28 match priv a b with (* A\B est videssi A inclus dans B *)
29 | [] -> true
30 | _ -> false
31
32 exception Elempas_present
33 (* S'exprime x ∈ a, et lève une exception si x n'est pas dans a*)
34 let rec suppr (x:'a) (a: 'a list) : 'a list =
35 match a with
36 | [] -> raise Elempas_present
37 | y::q -> if (x=y) then q
38 else y::(suppr x q)
39
40 (* Fonction auxiliaire qui décide une égalité ensembliste entre a et b *)
41 let rec egal_ens_aux (a: 'a list) (b : 'a list) : bool =
42 match a with
43 | [] -> (b = [])
44 | x::q -> egal_ens_aux q (suppr x b)
45
46 let egal_ens (a: 'a list) (b: 'a list) : bool =
47 try
48 egal_ens_aux a b
49 with
50 | Elempas_present -> false

```

## 4- Annexe • 4.1 Code

# Code du TIPE

## 2 Code de la deuxième partie

### 2.1 Code du fichier fnn.ml

```

1 (* Renvoie une formule équivalente à f, ne contenant plus aucun F, G *)
2 let supp_fg (phi : ltl) : ltl =
3 match phi with
4 | F phi -> U (Top, phi)
5 | G phi -> R (Bot, phi)
6 | _ -> phi
7
8 (* Transforme une formule f en une formule f' équivalente
9 mise sous forme normale négative *)
10 let rec fnn (phi ltl) : ltl =
11 match phi with
12 | Top | Bot | Atom _ -> phi
13 | Or (phi1, phi2) -> Or (fnn phi1, fnn phi2)
14 | And (phi1, phi2) -> And (fnn phi1, fnn phi2)
15 | U (phi1, phi2) -> U (fnn phi1, fnn phi2)
16 | R (phi1, phi2) -> R (fnn phi1, fnn phi2)
17 | X phi -> X (fnn phi)
18 | F _ | G _ -> fnn (supp_fg phi)
19 | Not phi ->
20 (* On va ici appliquer les lois de De Morgan *)
21 (* Et les lois de dualités spécifiques à la logique LTL *)
22 begin
23 match phi with
24 | Top -> Bot
25 | Bot -> Top
26 | Atom _ -> Not phi
27 | Not phi1 -> (fnn phi1) (* Elimination de la double négation *)
28 | Or (phi1, phi2) -> And (fnn (Not phi1), fnn (Not phi2)) (* Loi de De
29 Morgan *)
30 | And (phi1, phi2) -> Or (fnn (Not phi1), fnn (Not phi2))
31 | X phi1 -> X (fnn (Not phi1))
32 | F _ | G _ -> fnn (Not (supp_fg phi))
33 | U (phi1, phi2) -> R (fnn (Not phi1), fnn (Not phi2))
34 | R (phi1, phi2) -> U (fnn (Not phi1), fnn (Not phi2))
35 end
36
37
```

### 2.2 Code du fichier f\_to\_LGBA.ml

```

1 (* On a ici des formules ne contenant pas de F, de G, et donc toutes les
2 additions précédées les littères
3 (sauf sous forme normale négative) *)
4 let curr1 (phi: ltl) : ltl list =
5 match phi with
6 | U (phi1, phi2) -> [phi1]
7 | R (phi1, phi2) -> [phi2]
8 | Or (phi1, phi2) -> [phi1; phi2]
9 | _ -> []
10
11 let next1 (phi: ltl) : ltl list =
12 match phi with
13 | U (phi1, phi2) -> [phi]
14 | R (phi1, phi2) -> [phi]
15 | _ -> []
16
17 let curr2 (phi: ltl) : ltl list =
18 match phi with
19 | U (phi1, phi2) -> [phi2]
20 | R (phi1, phi2) -> [phi1; phi2]
21 | Or (phi1, phi2) -> [phi1]
22 | _ -> []
23
24 let neg (phi: ltl) : ltl =
25 match phi with
26 | Top -> Bot
27 | Bot -> Top
28 | Atom p -> Not phi
29 | Not (Atom p) -> Atom p
30 | _ -> phi
31
32 (* Réécritre l'ensemble d'éléments associé à un noeud dans set *)
33 let rec get_set (set : (noeud * ('a list)) list) (q: noeud) : 'a list =
34 match set with
35 | [] -> failwith "From get_set : Valeur non présente dans set"
36 | (x, l)::p -> if (x.name = q.name) then l else get_set p q
37
38 (* Renvoie incoming_u en version modifiée de tels manière que Incoming_u(q)
39 --> Incoming_u(q) U incoming *)
40 let rec edit_incoming (incoming : ((noeud * (noeud list)) list)) (incoming :
41 noeud list) (q: noeud) : (noeud * (noeud list)) list =
42 match incoming with
43 | [] -> [q, incoming]
44 | (x, l)::p -> if (x.name = q.name) then (q, union l incoming)::p
45 else (x, l)::(edit_incoming p incoming q)
46
47
48
49
50
```

## 4- Annexe • 4.1 Code

## Code du TIPE

```

47 (* Renvoie true si il existe q appartenant à nodeset / q.next=mode.next et
48 q.old=mode.old et dans ce cas la : modifie
49 q.incoming par q.incoming Union mode.incoming
50 Sinon renvoie false*)
51 let rec exists (next_u : (noud * (ltl list)) list) (next : ltl list) (now_u
52 : (noud * (ltl list)) list) (old: ltl list) (incoming : noud list)
53 (incoming_u : ((noud * (noud list)) list) ref) (nodeset: noud list) :
54 bool=
55 match nodeset with
56 | [] -> false
57 | (l,ltl)::rest Insert = get_set next_u q in
58 let lnow_u = get_set now_u q in
59 if ((egal_ens lnow_u) lk (egal_ens lnow old)) then (
60 let incoming_bis = edit_incoming ((incoming_u) incoming q) in
61 incoming_u := incoming_bis;
62 let _ = exists next_u next now_u old incoming incoming_u p in true
63)
64 else exists next_u next now_u old incoming incoming_u p
65
66 (* Renvoie le plus grand nombre présent dans nodeset *)
67 let rec max_name (l: noud list) =
68 match l with
69 | [] -> 0
70 | x::q -> max (int_of_string (x.name)) (max_name q)
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
847
848
849
849
850
851
852
853
854
855
856
856
857
858
859
859
860
861
862
863
864
865
865
866
867
867
868
869
869
870
871
872
873
873
874
875
875
876
877
877
878
879
879
880
881
882
882
883
884
884
885
886
886
887
887
888
889
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1640
1641
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1650
1651
1651
1652
1652
1653
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1660
1661
1661
1662
1662
1663
1663
1664
1664
1665
1665
1666
1666
1667
1667
1668
1668
1669
1669
1670
1670
1671
1671
1672
1672
1673
1673
1674
1674
1675
1675
1676
1676
1677
1677
1678
1678
1679
1679
1680
1680
1681
1681
1682
1682
1683
1683
1684
1684
1685
1685
1686
1686
1687
1687
1688
1688
1689
1689
1690
1690
1691
1691
1692
1692
1693
1693
1694
1694
1695
1695
1696
1696
1697
1697
1698
1698
1699
1699
1700
1700
1701
1701
1702
1702
1703
1703
1704
1704
1705
1705
1706
1706
1707
1707
1708
1708
1709
1709
1710
1710
1711
1711
1712
1712
1713
1713
1714
1714
1715
1715
1716
1716
1717
1717
1718
1718
1719
1719
1720
1720
1721
1721
1722
1722
1723
1723
1724
1724
1725
1725
1726
1726
1727
1727
1728
1728
1729
1729
1730
1730
1731
1731
1732
1732
1733
1733
1734
1734
1735
1735
1736
1736
1737
1737
1738
1738
1739
1739
1740
1740
1741
1741
1742
1742
1743
1743
1744
1744
1745
1745
1746
1746
1747
1747
1748
1748
1749
1749
1750
1750
1751
1751
1752
1752
1753
1753
1754
1754
1755
1755
1756
1756
1757
1757
1758
1758
1759
1759
1760
1760
1761
1761
1762
176
```

4- Annexe • 4.1 Code

## Code du TIPE

### 3 Code de la troisième partie

### 3.1 Code du fichier kripke\_to\_buchi.ml

```

(* Transformer une structure de kripke en un automate de buchi, (tout les
 - etats sont finaux)
D Renomme ici les états par k_i (nom de l'état) afin de pouvoir différencier
 - les états provenants d'une structure
de kripke de ceux provenant d'un automate de buchi)
let kripke_to_buchi (k : kripke_struct) : buchi_automate =
 let open List in
 let map (fun x -> (stat = "k_" ^ x stat)) (k,q) =
 init = List.map (fun x -> (stat = "k_" ^ x stat)) (k.init);
 transition = List.map (fun (x,y) -> ((stat = "k_" ^ x stat), (stat = "k_" ^ y stat)));
 final = List.map (fun x -> ((stat = "k_" ^ x stat))) (k.q);
 valuation = List.map (fun (x,f) -> ((stat = "k_" ^ x stat), f));
 -- (k.valuation))
 in
 buchi_automate {
 delta = Construir delta {q,g,q'} dans Nodes et q dans Incoming(g') } x
 let rec build_transition (nodes : noeud list) (incoming : (noeud * noeud list) list) : (state*state) list =
 (* Renomme l'(node,g') node,g' appartiennent Nodes et node appartiennent l *)
 let rec build_transition_with_list (nodes : noeud list) (node : noeud) (l : list) :
 noeud list : (state*state) list =
 match l with
 | [] -> []
 | q:p -> if (appartient q nodes) then ((stat = q.name),(stat =
 -- node.name));(build_transition_with_list nodes node p)
 else (build_transition_with_list nodes node p)
 in
 match incoming with
 | [] -> []
 | (q',l):p -> if ((appartient q' nodes)) then (build_transition_with_list
 -- nodes q'@(build_transition nodes p)
 else (build_transition nodes p)
 end

 (* Construit Q0 = { q appartient Nodes / init appartient Incoming(q) } *)
 let rec build_d_init (nodes : noeud list) (incoming : (noeud * noeud list) list) :
 init : noeud) : state list =
 match incoming with
 | [] -> []
 | (q,l):p -> if ((appartient init l) & (q.name <= "init")) then ((stat =
 -- q.name);(build_init nodes p init))
 else build_d_init nodes p init (* en exclut ici init avec la condition
 -- q.name <= "init" car en veux les q dans Nodes et init n'est pas
 dans Nodes *)
 end

```

```

(* Extract l'ensemble des formules de now (qui est de type (noeud*ltl)) *)
let rec extract_ltl (now: (noeud*ltl) list) : ltl list =
 match now with
 | [] => []
 | (q,f)::p => let res = extract_ltl p in
 if (appartient f res) then res
 else f::res
 in res

(* Du va maintenant construire les états finaux *)
let rec build_fg (nodes: noeud list) (now: (noeud*ltl) list) (gl: ltl list) =
 match now with
 | [] => []
 | (q,l)::p => let res = (build_fg nodes p gl) in
 if ((appartient q2 l) || (not (appartient (U(gl,g2)) l))) then
 (appartient (etat = q.name) res))
 then (etat = q.name)::res
 else res
 else res

(* Détermine si g est une sous formule de f *)
let rec est_sous_formule (g: ltl) (f: ltl) : bool =
 if (F=g) then true else
 match f with
 | Top | Bot | Atom _ => (f=g) (*false donc*)
 | Not f' | X f' | F f' | G f' => est_sous_formule g f'
 | Or (f1,f2) | And (f1,f2) | U (f1,f2) | R (f1,f2) => (est_sous_formule g
 -- f1) || (est_sous_formule g f2)
 in

(* Extrait les sous formules de f de la forme phi U psi *)
let rec extract_sous_formule (f: ltl) : ltl list =
 match f with
 | Atom p => [U (Atom p, Atom p); U (Not (Atom p), Not (Atom p))]
 | Top => [U (Top, Top)]
 | Bot => []
 | U (f1,f2) => f1::(union (extract_sous_formule f1) (extract_sous_formule
 -- f2))
 | And (f1,f2) | Or (f1,f2) => (union (extract_sous_formule f1)
 -- (extract_sous_formule f2))
 | R (f1,f2) =>
 if (R (f1,f2) équivalent à Not U (Not f1,Not f2) *)
 then (union (extract_sous_formule (Not f1)) (extract_sous_formule (Not
 -- f2)))
 else []
 | X f1 | Not f1 => extract_sous_formule f1
 | G _ | F _ => failwith "formule non mise sous forme normale négative"

```

## 4- Annexe • 4.1 Code

## Code du TIPE

```

1 (* Transforme un ensemble de noeud en un ensemble d'états *)
2 let rec nodes_to_state (nodes : noeud list) : state list =
3 match nodes with
4 | [] => []
5 | x::q => (x.stat = x.name)::(nodes_to_state q)
6
7 let build_final (f: ltl) (nodes: noeud list) (now : (noeud* (ltl list)) list)
8 (* state list =
9 * let rec aux (set : ltl list) : state list list =
10 (* Construit F = { Fg | g appartient cl(f) } *)
11 match set with
12 | [] => []
13 | (Bg,g2)::q => (build_fg nodes now g2)::(aux q)
14 | (Bg,g2)::q => (build_fg nodes now (Not g1) (Not g2))::(aux q)
15 | phi::q => failwith "ne devrait pas arriver"
16 in
17 List.fold_left (fun x y => inter x y) (nodes_to_state nodes) (aux
18 (* extract_sous_formule f))
19 (* On prend l'intersection de tout les ensembles obtenues avec aux *)
20)
21
22 (* Extrait les littéraux d'un ensemble de couple (noeud, ensemble de
23 formule) *)
24 let rec littéraux (q: noeud) (now : (noeud* (ltl list)) list) : ltl list =
25 let rec littéraux_aux (ltsent: ltl list) : ltl list =
26 (* Récupère tous les littéraux d'un ensemble de formule ltiset *)
27 match ltsent with
28 | [] => []
29 | f::p => let res = (littéraux_aux p) in
30 begin
31 match f with
32 | Atom p => if (not (appartient f res)) then (Atom p)::res
33 else res
34 | Not (Atom p) => if (not (appartient f res)) then (Not (Atom p))::res
35 else res
36 | _ -> res
37 end
38 in
39 match now with
40 | [] => []
41 | (x1)::p => if (x1.name = q.name) then (
42 (littéraux_aux ltsent)@littéraux q p)
43)
44 else (littéraux q p)
45
46
47 (* Construit la valuation pour l'automate de buchi *)
48 let rec build_valuation (nodes : noeud list) (now : (noeud* (ltl list)) list)
49 (* state * (ltl list) list =
50 match nodes with
51 | [] => []
52 | q::p -> ((stat = q.name), littéraux q now)::(build_valuation p now)
53
54
55 (* Construit l'ensemble d'états *)
56 let rec build_state (nodes : noeud list) : state list =
57 match nodes with
58 | [] => []
59 | q::p -> (stat = q.name)::(build_state p)
60
61
62 (* Récupère le noeud initial dans nodes *)
63 let rec get_init (nodes : noeud list) : noeud =
64 match nodes with
65 | [] -> failwith "noeud initial non trouvé"
66 | x::q => if (x.name = "init") then x
67 else get_init q
68
69
70 let build_lgba (f: ltl) : buchi_automate =
71 let init = (name = "init") in
72 let (nodes,now,incoming) = create_graph f in
73
74 q = build_state nodes;
75 init = build_init nodes incoming init;
76 transition = build_transition nodes incoming;
77 final = build_final f nodes now;
78 valuation = build_valuation nodes now
79
80
81

```

## 4- Annexe • 4.1 Code

## Code du TIPE

## 3.2 Code du fichier intersect.ml

```

1 (* Réalise l'opération $q_1 \otimes q_2$ (produit cartésien des états) *)
2 let rec prod_states (q1: state list) (q2: state list) : (state * state) list =
3 let rec prod_cart (q: state) (l1: state list) : (state * state) list =
4 match l1 with
5 | [] -> []
6 | y::l1 -> (q,y)::(prod_cart q l1)
7 in
8 match q1 with
9 | [] -> []
10 | x::l -> let res = prod_states l q2 in
11 union (prod_cart x q2) res
12
13 (* Permet d'obtenir les états initiaux de l'automate produit, $I = \{(q,q') / q$
14 appartient I_1 , q' appartient $I_2\}$ *)
15 (* On va donc naturellement utiliser la fonction précédente *)
16 let prod_init (b1 : buchi_automate) (b2 : buchi_automate) : (state * state) list =
17 =
18 prod_states (b1.init) (b2.init)
19
20 (* Un état (q,q') est final pour l'automate produitssi q final pour b_1 et q'
21 final pour b_2 *)
22 let prod_final (b1: buchi_automate) (b2:buchi_automate) : (state * state) list =
23 prod_states (b1.final) (b2.final)
24
25 (* Récupère l'ensemble des transitions sortantes de q *)
26 let rec get_transition (q: state) (t: (state * state) list) : (state * state)
27 =
28 =
29 match t with
30 | [] -> []
31 | (x,y)::p -> if (x=q) then (x,y)::(get_transition q p)
32 else (get_transition q p)
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
737
738
739
739
740
741
742
743
744
745
745
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1640
1641
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1650
1651
1651
1652
1652
1653
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1660
1661
1661
1662
1662
1663
1663
1664
1664
1665
1665
1666
1666
1667
1667
1668
1668
1669
1669
1670
1670
1671
1671
1672
1672
1673
1673
1674
1674
1675
1675
1676
1676
1677
1677
1678
1678

```

4- Annexe • 4.1 Code

## Code du TIPE

```

(* Detecte si un ensemble de formule content une absurdité du type [Not
 ~Atom "p"] ou (~Atom "p'") *)
let rec absurdre (fset : ltl list) : bool =
 match fset with
 | [] => false
 | (Atom p)::q => if (appartient (Not (Atom p)) fset) then true
 else absurdre q
 | (Not (Atom p))::q => if (appartient (Atom p) fset) then true
 else absurdre q
 (* On a ici pas besoin de "regarder en arrière" dans la liste car on la
 parcourt de gauche à droite *)
 | f::q => absurdre q

(* Réalise la fonction de transition pour l'automate produit *)
let rec prod_transition (bl: buchi,automate) (b2:buchi,automate) : ((state *
 state) * (state * state)) list =
 let etat = ref bl.state in
 let etat2 = ref b2.state in
 let alphabet = get_alphabet bl.buchi in
 let alphabet2 = get_alphabet b2.buchi in
 let alpha = union_alphabet alphabet2 in
 List.assoc etat (List.assoc etat2 (fun q'=>
 (* Rappel : valuation correspond à l'étiquetage des transitions sortants
 d'un état, ainsi si
 deux transitions (q_1, q_2') et (q_1', q_2) correspondent en
 valuationdans auton1(q_1) = valuationdans auton2(q_2)
 alors on peut ajouter la transition ((q_1, q_1'), (q_2, q_2')) *)
 let v1 = List.assoc q (bl.valuation) in
 let v2 = List.assoc q' (b2.valuation) in
 (* Il y a ici un problème, lorsque la valuation étiquette (Not (Atom
 "p")), cela
 signifie que n'importe quelle lettre différente de p permet de prendre la
 transition *)
 let v1' = modifie_ltlset v1 alpha in
 let v2' = modifie_ltlset v2 alpha in
 let union_v1v2' = union v1 v2 in
 if ((inter v1' v2') <> [] || not (absurde union_v1v2')) then (
 let t1 = get_transition q (bl.transition) in
 let t2 = get_transition q' (b2.transition) in
 let fus = fusion t1 t2 in
 let res = union (res) fus
)
)
 else ()) states;
 res

```

```

4 * Réalise la fonction valuation pour l'automate produit *
let rec prod_valuation (b1: buchi_automate) (b2:buchi_automate) : ((state *
5 * state) * (Itl list)) list
6 * states = prod_states (b1,q) (b2,q) in
7 * let res = ref [] in let alphabet1 = get_alphabet b1 in let alphabet2 =
8 * get_alphabet b2 in
9 * let alpha = union alphabet1 alphabet2 in
List Liter
10 * fun (q,q') ->
11 * let v1 = List.assoc q (b1.valuation) in
12 * let v2 = List.assoc q' (b2.valuation) in
13 * let v1' = modifie_litlist v1 alpha in let v2' = modifie_litlist v2 alpha
14 * let v1v2 = union v1 v2 in
15 * if (inter v1' v2') <> [] then (absurde union.viv2)) then (
16 * res := union (res) (((q,q'),(union.viv2)))
17 *)
18 * else (res := union (res) [(q,q'),[]]))
) states;
!res

19 *
20 *

type buchi_automate_product = {
21 * q : ((state * state) list;
22 * init : ((state * state) list;
23 * transition : ((state * state) * (state * state)) list;
24 * final : ((state * state) list;
25 * valuation : ((state * state) * (Itl list)) list
}

26 *
27 let create_product (b1 buchi_automate) (b2:buchi_automate) :
28 buchi_automate_product =
29 {
30 q = prod_states (b1.q) (b2.q);
31 init = prod_init b1 b2;
32 transition = prod_transition b1 b2;
33 final = prod_final b1 b2;
34 valuation = prod_valuation b1 b2;
}

```

## 4- Annexe • 4.1 Code

# Code du TIPE

### 3.3 Code du fichier empty.ml

```

1 (* Récupère l'ensemble des états atteints en un coup depuis q avec
2 t (transition) *)
3 let rec get_state_set (q: ((state*state))) (t: ((state * state)*(state *
4 state) list) : (state*state) list =
5 match t with
6 | [] => []
7 | (x,y)::p -> if (x=q) then y::(get_state_set q p)
8 else (get_state_set q p)
9
10 exception Found
11
12 let rec innerdfa (q: (state*state)) (outervisited : (state*state) list) =
13 (innervisited : (state*state) list) (b: buchi_automate_product) : unit =
14 let innervisited' = q::innervisited in
15 List.iter (fun q' -> if (appartient q' outervisited) then (raise Found)
16 else if (not (appartient q' innervisited')) then (innerdfa q' outervisited
17 innervisited' b) (get_state_set q (b.transition)))
18
19
20 let rec outerdfa (q:(state*state)) (visited : (state*state) list) (b:
21 buchi_automate_product) : unit =
22 let visited' = q::visited in
23 List.iter (fun q' -> if (not (appartient q' visited')) then (outerdfa q'
24 visited' b) (get_state_set q (b.transition));
25 if (appartient q (b.final)) then (innerdfa q visited' [] b)
26 else ())
27
28 let is_empty (b: buchi_automate_product) : bool =
29 let res = ref true in
30 List.iter (fun q ->
31 try
32 outerdfa q [] b
33 with
34 | Found -> res := false
35 | _ -> (b.init);
36) res
37
38 let model_checker (k: kripke_struct) (f: ltl) : bool =
39 let b2 = kripke_to_buchi k in
40 let b1 = build_lgba (fm (Not f)) in
41 let product = create_product b1 b2 in
42 ((is_empty product))
43

```