

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

LGBA

Logique LTL

Objectifs

2. Implémentation LTL

3. Faire l'intersection entre deux automates de Büchi

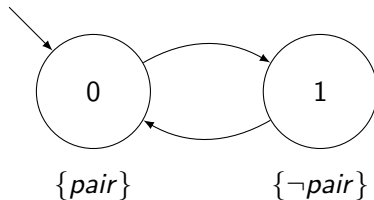
4. Annexe

Objectifs du Model-Checking

3 étapes clés :

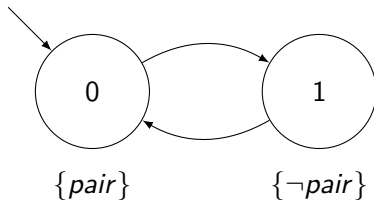
- **Modéliser** : trouver une **bonne** modélisation pour le système pour lequel on souhaite vérifier une propriété
- **Définir** : la **propriété** que doit vérifier le modèle
- **Tester** : si le modèle vérifie la propriété

Structure de Kripke - Exemple



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

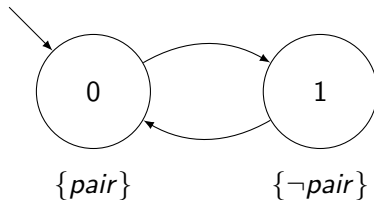
Structure de Kripke - Exemple



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

$$\mathcal{Q} = \{0, 1\}$$

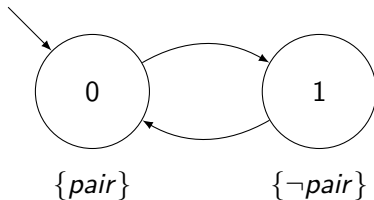
Structure de Kripke - Exemple



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

$$\begin{array}{lcl} \mathcal{Q} & = & \{0, 1\} \\ I (\subseteq \mathcal{Q}) & = & \{0\} \end{array}$$

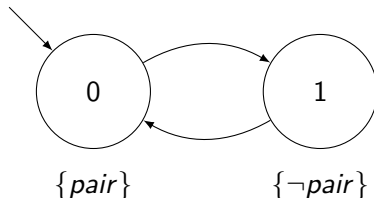
Structure de Kripke - Exemple



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

$$\begin{aligned}\mathcal{Q} &= \{0, 1\} \\ I (\subseteq \mathcal{Q}) &= \{0\} \\ \rightarrow (\subseteq \mathcal{Q} \times \mathcal{Q}) &= \{(0, 1), (1, 0)\}\end{aligned}$$

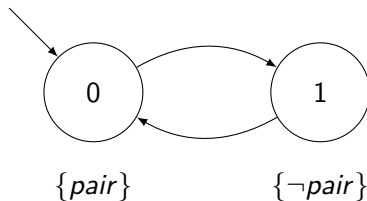
Structure de Kripke - Exemple



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

$$\begin{aligned}
 \mathcal{Q} &= \{0, 1\} \\
 I (\subseteq \mathcal{Q}) &= \{0\} \\
 \rightarrow (\subseteq \mathcal{Q} \times \mathcal{Q}) &= \{(0, 1), (1, 0)\} \\
 \mathcal{L} (\subseteq \mathcal{Q} \times \mathcal{P}(AP \cup \neg AP)) &= \{(0, \{pair\}), (1, \{\neg pair\})\}
 \end{aligned}$$

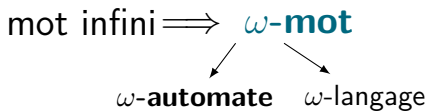
Structure de Kripke - Exécution



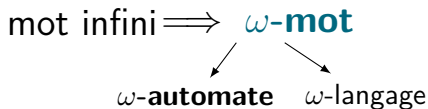
Structure de Kripke modélisant un compteur

Exécution sur \mathcal{M} : $S = (s_n)_{n \in \mathbf{N}}$ tq $\forall i \in \mathbf{N}, s_{2i} = 0$ et $s_{2i+1} = 1$
mot infini

LGBA



LGBA



Automate de Büchi Généralisé Étiqueté (LGBA)



$$F = \{F_1, \dots, F_n\} \subseteq (\mathcal{P}(\mathcal{Q}))^n$$

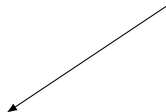
$\forall i, \exists q \in F_i$ tq on
passe par q infiniment souvent

LTL

Linear Temporal Logic (LTL)

LTL

Linear Temporal Logic (LTL)



Logique propositionnelle

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

$p \in AP$ $\varphi_1 \vee \varphi_2$

LTL

Linear Temporal Logic (LTL)

Logique propositionnelle

+

Opérateurs temporels

\perp \top $\neg\varphi$ $\varphi_1 \wedge \varphi_2$
 $p \in AP$ $\varphi_1 \vee \varphi_2$

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

LTL

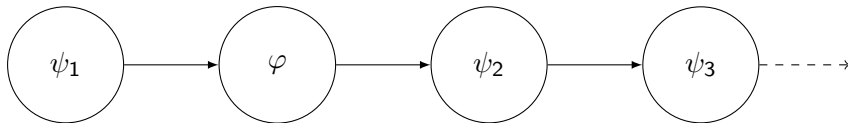
Linear Temporal Logic (LTL)

Logique propositionnelle

+

Opérateurs temporels

\perp \top $\neg\varphi$ $\varphi_1 \wedge \varphi_2$ $X\varphi$ $F\varphi$ $G\varphi$ $\varphi_1 U \varphi_2$
 $p \in AP$ $\varphi_1 \vee \varphi_2$ $\varphi_1 R \varphi_2$



LTL

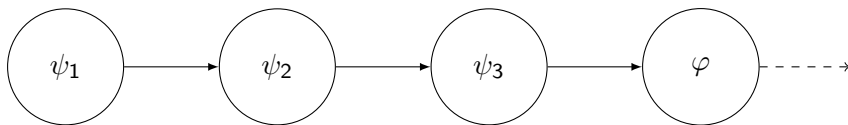
Linear Temporal Logic (LTL)

Logique propositionnelle

+

Opérateurs temporels

\perp \top $\neg\varphi$ $\varphi_1 \wedge \varphi_2$ $X\varphi$ $F\varphi$ $G\varphi$ $\varphi_1 U \varphi_2$
 $p \in AP$ $\varphi_1 \vee \varphi_2$ $\varphi_1 R \varphi_2$



LTL

Linear Temporal Logic (LTL)

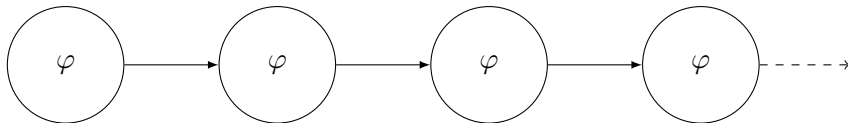
Logique propositionnelle

+

Opérateurs temporels

\perp \top $\neg\varphi$ $\varphi_1 \wedge \varphi_2$
 $p \in AP$ $\varphi_1 \vee \varphi_2$

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



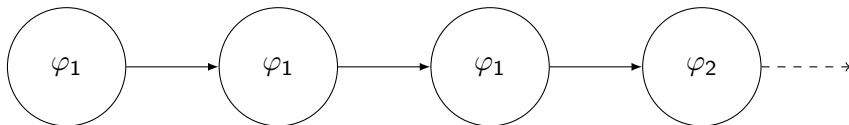
LTL

Linear Temporal Logic (LTL)

Logique propositionnelle

+

Opérateurs temporels

 \perp \top $\neg\varphi$ $\varphi_1 \wedge \varphi_2$
 $p \in AP$ $\varphi_1 \vee \varphi_2$
 $X\varphi$ $F\varphi$ $G\varphi$ $\varphi_1 U \varphi_2$
 $\varphi_1 R \varphi_2$


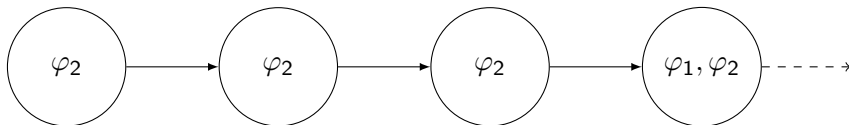
LTL

Linear Temporal Logic (LTL)

Logique propositionnelle

+

Opérateurs temporels

 \perp \top $\neg\varphi$ $\varphi_1 \wedge \varphi_2$
 $p \in AP$ $\varphi_1 \vee \varphi_2$
 $X\varphi$ $F\varphi$ $G\varphi$ $\varphi_1 U \varphi_2$
 $\varphi_1 R \varphi_2$


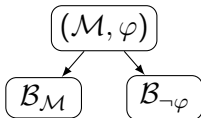
Tester

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

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

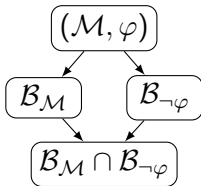
Tester

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



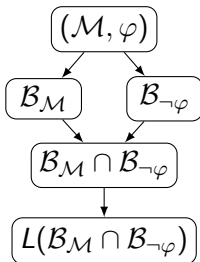
Tester

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



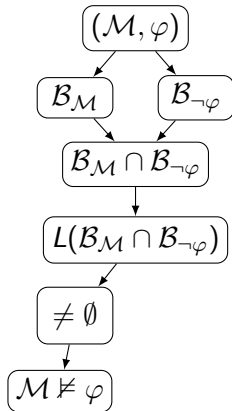
Tester

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



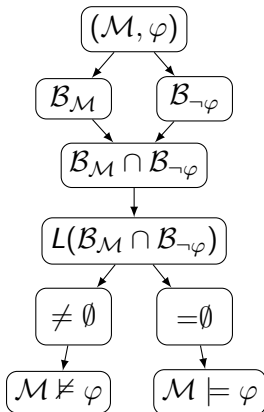
Tester

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

Transformation $LTL \rightarrow LGBA$

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
- ▶ $p, p \in AP$
- ▶ $\neg \varphi, \varphi \in LTL$
- ▶ $\varphi \wedge \psi, (\varphi, \psi) \in LTL \times LTL$
- ▶ $\varphi \vee \psi, (\varphi, \psi) \in LTL \times LTL$
- ▶ $X \varphi, \varphi \in LTL$ (Next)
- ▶ $G \varphi, \varphi \in LTL$ (Globally)
- ▶ $F \varphi, \varphi \in LTL$ (Finally)
- ▶ $\varphi U \psi, (\varphi, \psi) \in LTL \times LTL$ (Until)
- ▶ $\varphi R \psi, (\varphi, \psi) \in LTL \times LTL$ (Release)

```

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)

Une formule LTL est dite sous forme normale négative lorsque :

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

Définition d'une FNN (forme normale négative)

Une formule LTL est dite sous forme normale négative lorsque :

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

Exemple

$$(pUq) \wedge (X(\neg q))$$

$$\neg(pUq)$$

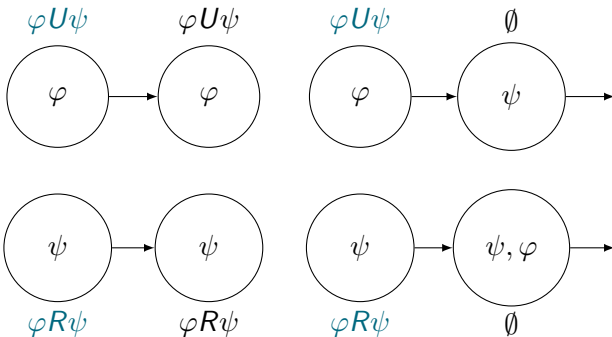
$$(Xp)U(F(\neg q))$$

Les règles utilisées

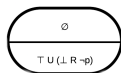
| Loi de De Morgan | |
|------------------------------|-------------------------------------|
| Formule initiale | Formule équivalente |
| $\neg (\varphi \wedge \psi)$ | $(\neg \varphi) \vee (\neg \psi)$ |
| $\neg (\varphi \vee \psi)$ | $(\neg \varphi) \wedge (\neg \psi)$ |

Règles utilisées

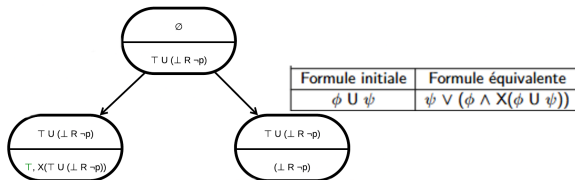
| "Formules d'expansion" | |
|---------------------------|--|
| Formule initiale | Formule équivalente |
| $\varphi \text{ U } \psi$ | $\psi \vee (\varphi \wedge \text{X}(\varphi \text{ U } \psi))$ |
| $\varphi \text{ R } \psi$ | $(\psi \wedge \varphi) \vee (\psi \wedge \text{X}(\varphi \text{ R } \psi))$ |



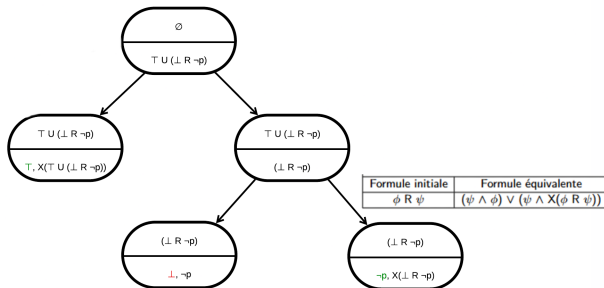
Construction du LGBA de $\neg G(F p) (= \top \cup (\perp R \neg p))$



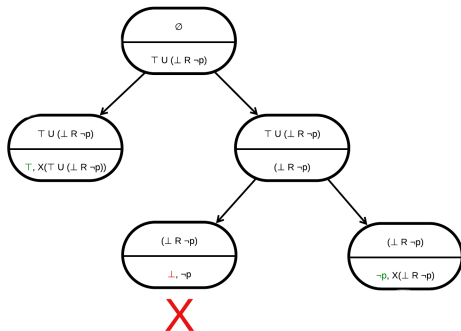
Construction du LGBA de $\neg G(F p) (= \top U (\perp R \neg p))$



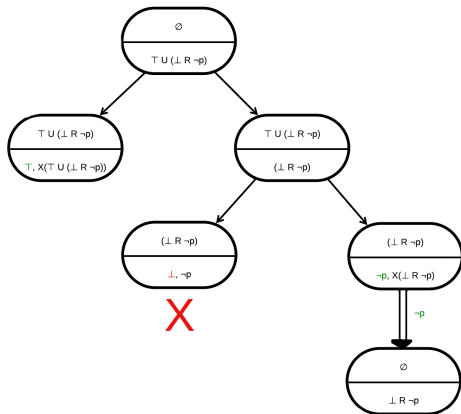
Construction du LGBA de $\neg G(F p) (= \top U (\perp R \neg p))$



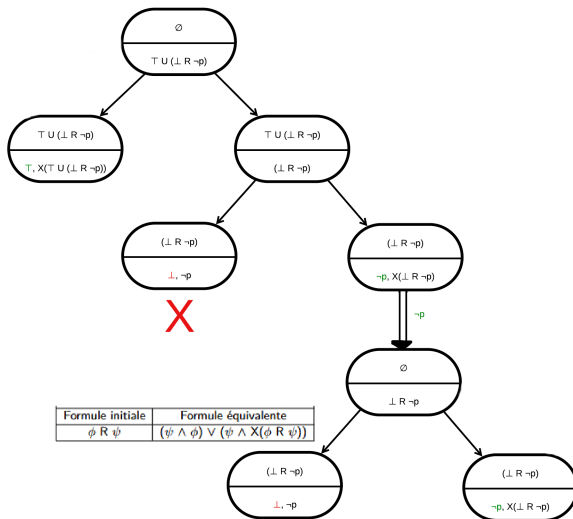
Construction du LGBA de $\neg G(F p) (= \top \cup (\perp R \neg p))$



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

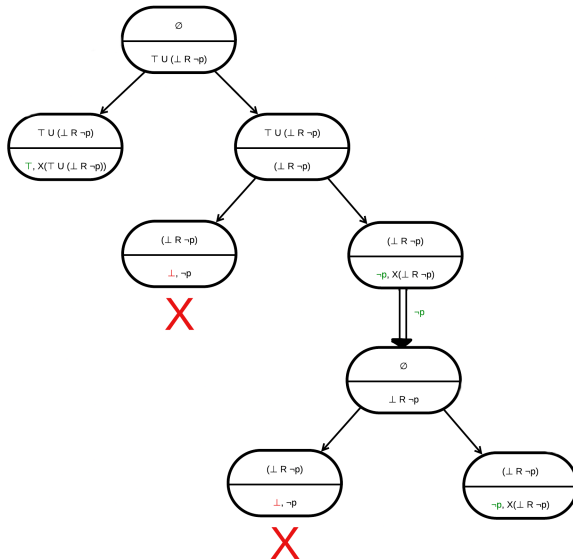


Construction du LGBA de $\neg G(F p) (= \top U (\perp R \neg p))$

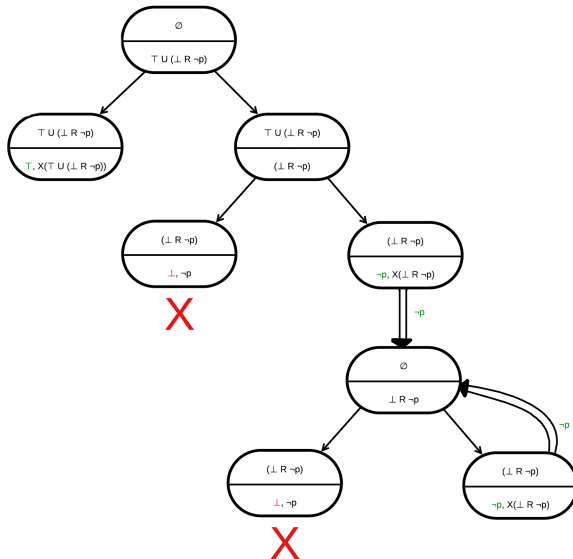


| Formule initiale | Formule équivalente |
|------------------|--|
| $\phi R \psi$ | $(\psi \wedge \phi) \vee (\psi \wedge X(\phi R \psi))$ |

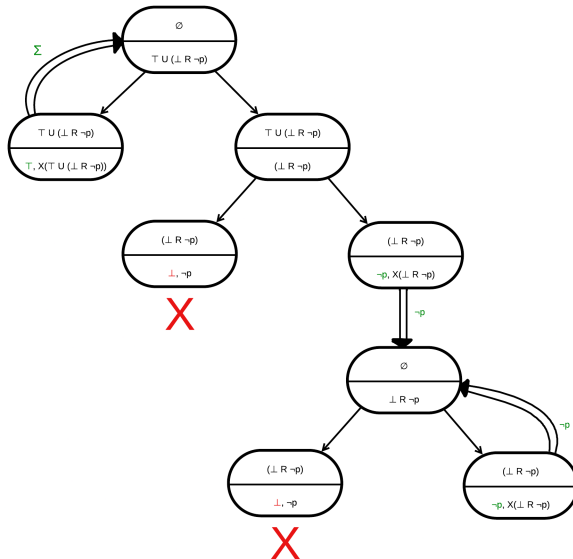
Construction du LGBA de $\neg G(F p) (= \top U (\perp R \neg p))$



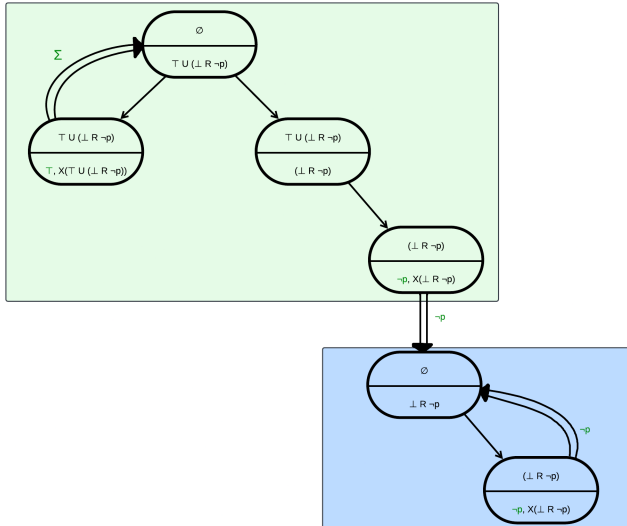
Construction du LGBA de $\neg G(F p) (= \top U (\perp R \neg p))$



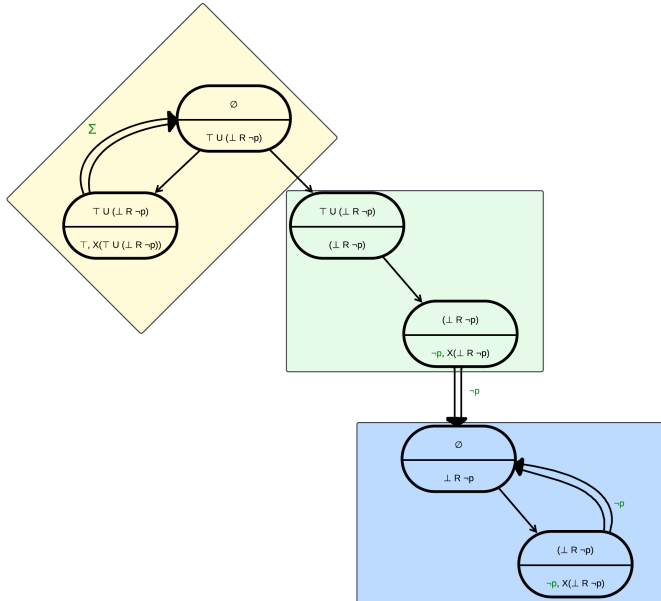
Construction du LGBA de $\neg G(F p) (= \top U (\perp R \neg p))$



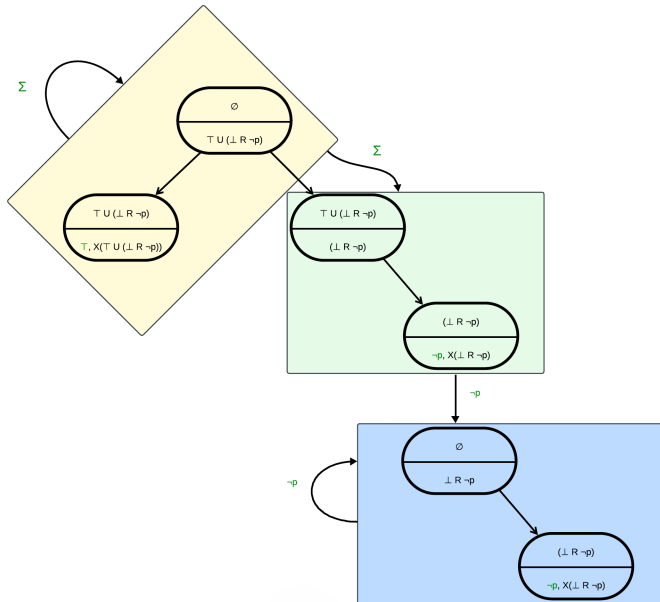
Construction du LGBA de $\neg G(F p) (= \top U (\perp R \neg p))$



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

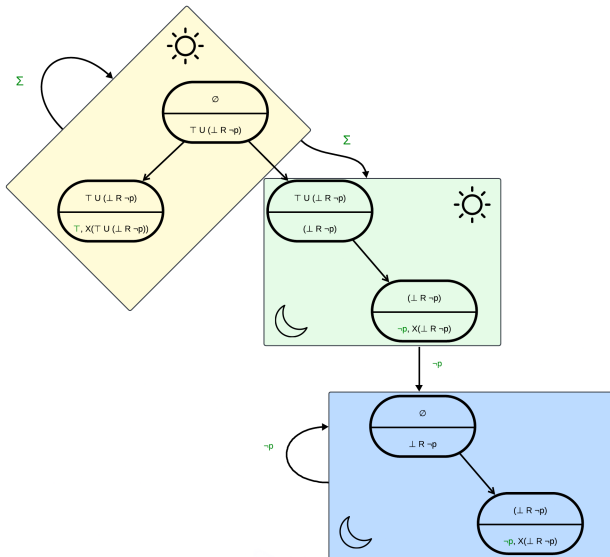


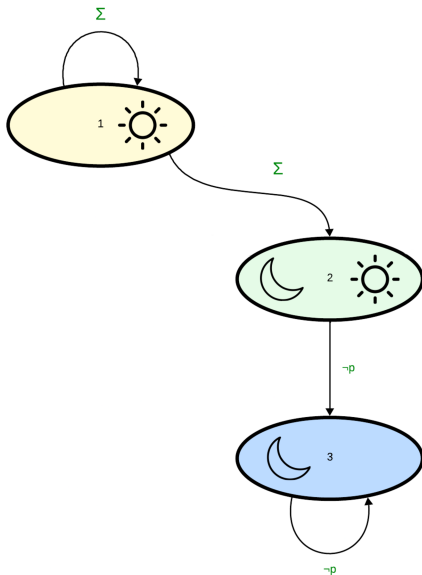
Construction du LGBA de $\neg G(F p) (= \top \cup (\perp R \neg p))$

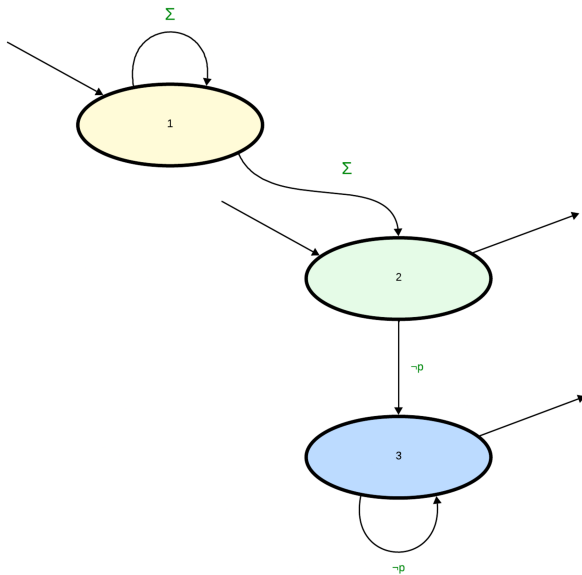




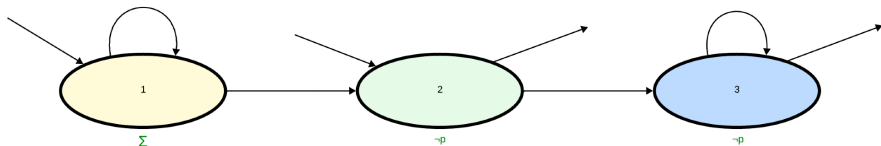
Construction du LGBA de $\neg G(F p) (= \top U (\perp R \neg p))$



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

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

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



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

On pose $A = (\Sigma, Q', I', F, \delta, \mathcal{L}')$ un LGBA où

$$\rightarrow Q' = Q$$

$$\rightarrow I' = I$$

$$\rightarrow F = Q$$

$$\rightarrow \delta = \rightarrow$$

$$\rightarrow \mathcal{L}' = \mathcal{L}$$

Intersection de deux LGBA

Soient $A_1 = (\Sigma, Q_1, I_1, F_1, \delta_1, \mathcal{L}_1)$ et $A_2 = (\Sigma, Q_2, I_2, F_2, \delta_2, \mathcal{L}_2)$ deux LGBA.

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

- ▶ $Q = Q_1 \times Q_2$
- ▶ $I = I_1 \times I_2$
- ▶ $F = F_1 \times F_2$
- ▶ $\delta = \{((q,s),(q',s')) \in Q \times Q \mid (q,q') \in \delta_1 \text{ et } (s,s') \in \delta_2 \text{ et } (\mathcal{L}_1(q) \cap \mathcal{L}_2(q')) \neq \emptyset\}$
- ▶ $\mathcal{L} = \{((q,q'), (\mathcal{L}_1(q) \cap \mathcal{L}_2(q')))) \mid (q,q') \in Q\}$

Exemple

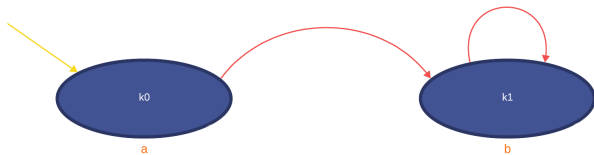


Figure – Structure de Kripke vérifiant $G(F b)$

Exemple

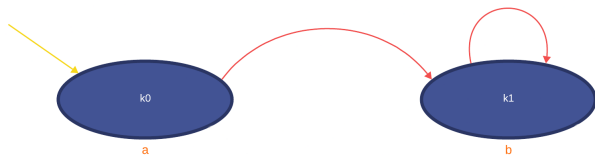


Figure – Structure de Kripke vérifiant $G(F b)$

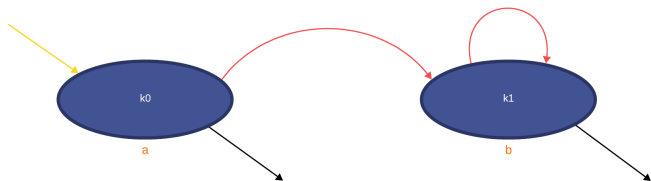


Figure – LGBA associé à la structure de Kripke

Exemple

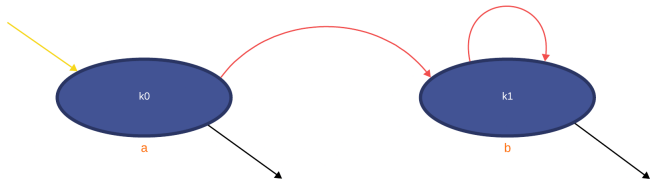


Figure – LGBA associé à la structure de Kripke

Exemple

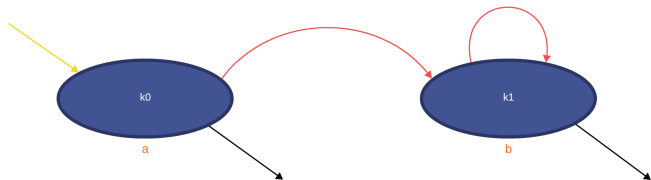


Figure – LGBA associé à la structure de Kripke

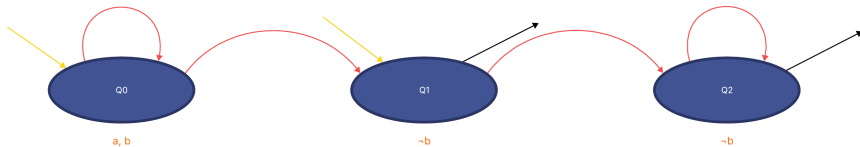


Figure – LGBA associé à la formule $\neg G(F b)$

Exemple

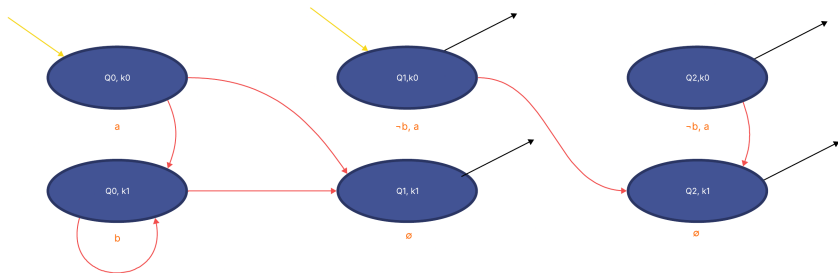
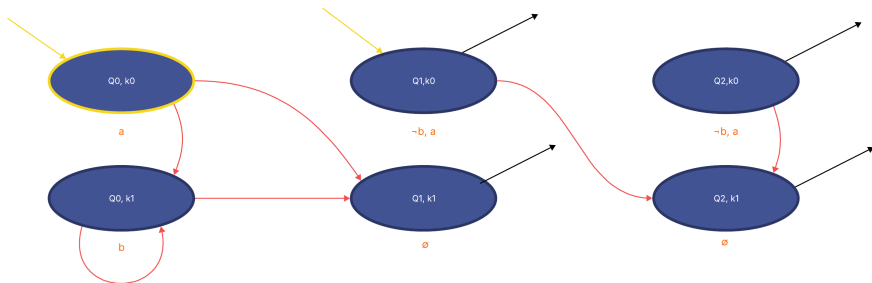


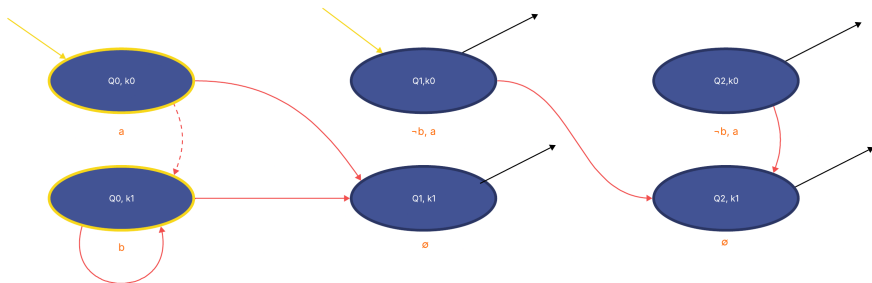
Figure – Intersection des deux LGBA

Exemple d'exécution



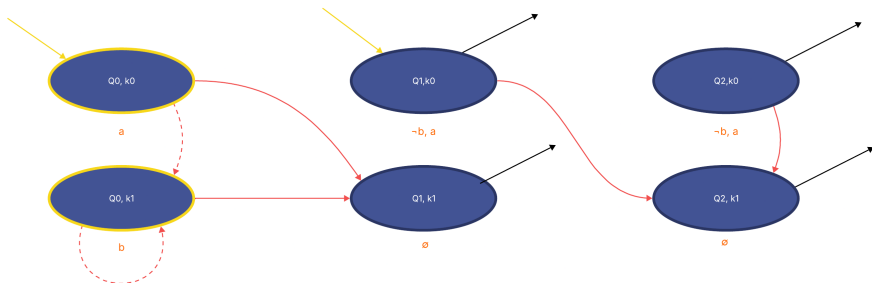
outerdfs Visités : $[(Q0, k0)]$

Exemple d'exécution



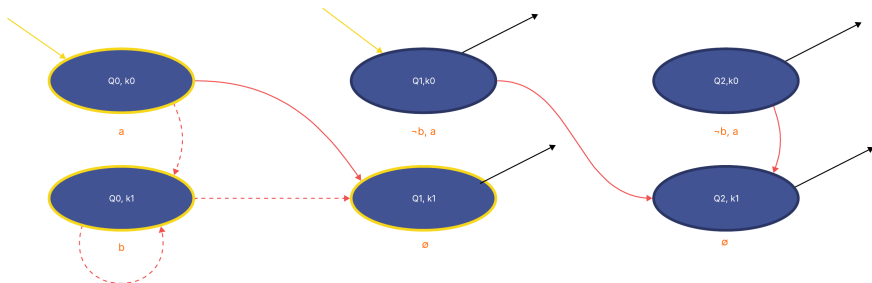
outerdfs Visités : $[(Q0, k0), (Q0, k1)]$

Exemple d'exécution



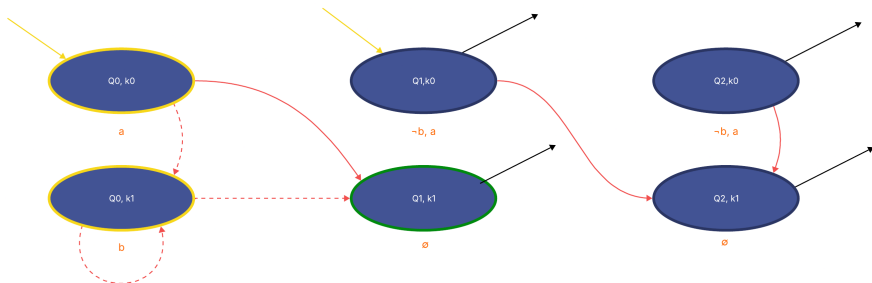
outerdfs Visités : [(Q0,k0), (Q0,k1)]

Exemple d'exécution



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

Exemple d'exécution

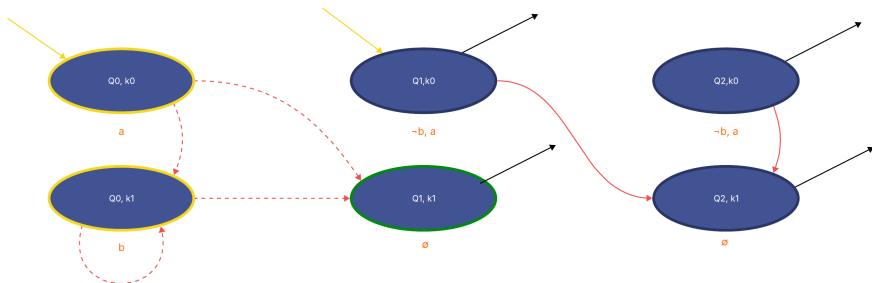


outerdfs Visités : $[(Q0, k0), (Q0, k1), (Q1, k1)]$

innerdfs1 VisitésIn1 : $[(Q1, k1)]$

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

Exemple d'exécution

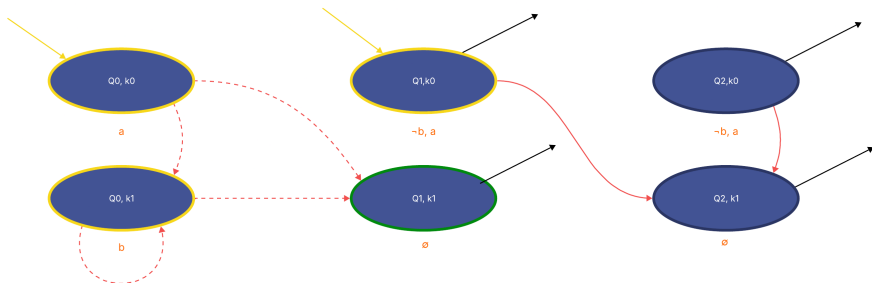


outerdfs Visités : $[(Q0, k0), (Q0, k1), (Q1, k1)]$

innerdfs1 VisitésIn1 : $[(Q1, k1)]$

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

Exemple d'exécution

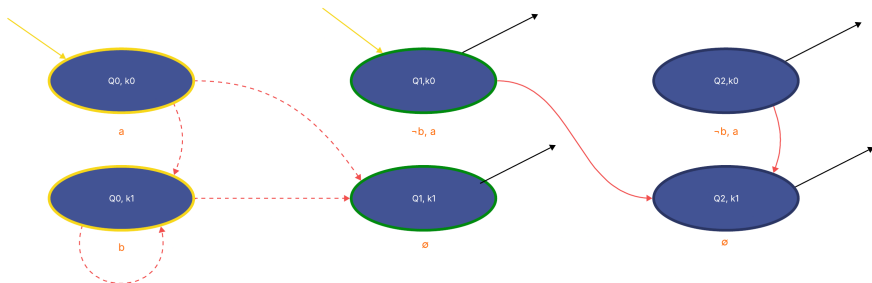


outerdfs Visités : $[(Q1, k0)]$

innerdfs1 VisitésIn1 : $[(Q1, k1)]$

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

Exemple d'exécution



outerdfs Visités : $[(Q1, k0)]$

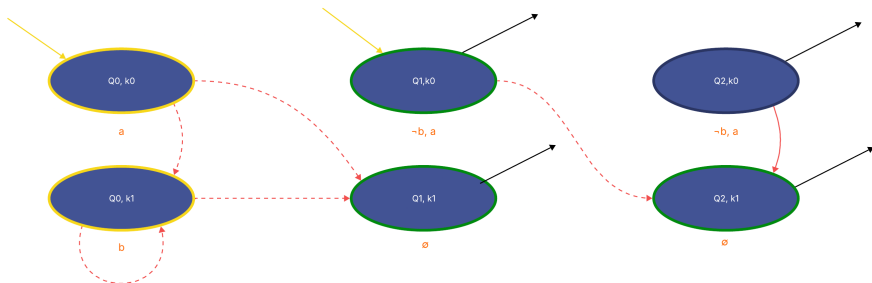
innerdfs1 VisitésIn1 : $[(Q1, k1)]$

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

innerdfs2 VisitésIn2 : $[(Q1, k0)]$

innerdfs2 VisitésExt2 : $[(Q1, k0)]$

Exemple d'exécution



outerdfs Visités : $[(Q1, k0)]$

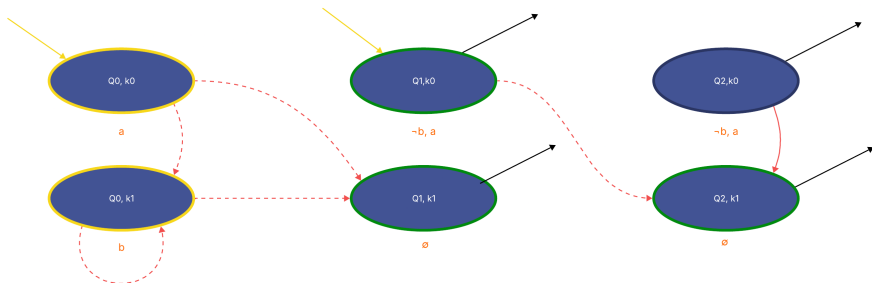
innerdfs1 VisitésIn1 : $[(Q1, k1)]$

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

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

innerdfs2 VisitésExt2 : $[(Q1, k0)]$

Exemple d'exécution



outerdfs Visités : [(Q1,k0)]

innerdfs1 VisitésIn1 : [(Q1,k1)]

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

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

innerdfs2 VisitésExt2 : [(Q1,k0)]

Estvide \rightarrow Vrai

Conclusion

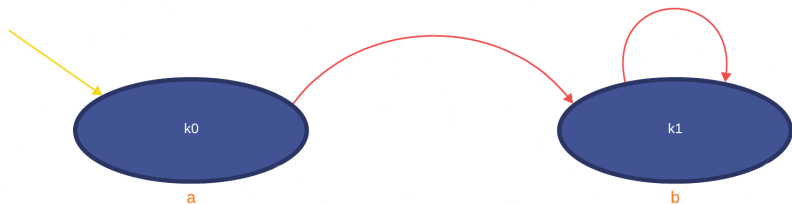


Figure – Structure de Kripke vérifiant $G(F b)$

1. Définitions

2. Implémentation LTL

3. Faire l'intersection entre deux automates de Büchi

4. Annexe

Code

Langage

Algorithmes

Règles pour transformer en FNN

FNN

Notions d'ensembles en Ocaml

Code du TIPE

```
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 noeud = {name : string}
15
16   type state = {etat : string}
17
18   type kripke_struct = {
19     q : state list;
20     init: state list;
21     transition : (state * state) list;
22     valuation : (state * (ltl list)) list ;
23     (* une fonction qui à un état associe un ensemble de formules ltl*)
24     (* Cf définition dans le diapo *)
25   }
26
```

Code du TIPE

```
27 type buchi_automate = {
28   q : state list;
29   init : state list;
30   final : state list;
31   transition : (state * state) list;
32   valuation : (state * (ltl list)) list
33   (* La valuation correspond au "L" dans la définition du LGBA (pour labelled generalised
34   ↪ buchi automaton) *)
35 }
36 (* Fin des declarations de types *)
37
38 (* On va maintenant coder un système d'ensemble avec des listes *)
39 (* Exactionent les mêmes que pour le model checker ctl *)
40 let rec prive (a: 'a list) (b: 'a list) : 'a list =
41   (* Réalise  $A \setminus B$  *)
42   match a with
43   | [] -> []
44   | x::q -> if (List.mem x b) then prive q b
45   else x::(prive q b)
46
```

Code du TIPE

```

47 let rec union (a: 'a list) (b: 'a list) : 'a list =
48   (* Réalise A ∪ B (sans doublons si a et b sont sans doublons)*)
49   match a with
50   | [] -> b
51   | x::q -> if (List.mem x b) then union q b
52   else x::(union q b)
53
54 let rec inter (a: 'a list) (b: 'a list) : 'a list =
55   (* réalise a ∩ b *)
56   match a with
57   | [] -> []
58   | x::q -> if (List.mem x b) then x::(inter q b)
59   else inter q b
60
61 let appartient (x: 'a) (a: 'a list) : bool =
62   List.mem x a
63
64 let inclus (a: 'a list) (b: 'a list) : bool =
65   (* Test si A inclus dans B *)
66   match (prive a b) with (* A \ B est vide ssi A inclus dans B *)
67   | [] -> true
68   | _ -> false
69
70 exception Elem_pas_present
71 (* Supprime x de a, et lève une exception si x n'est pas dans a *)
72 let rec suppr (x: 'a) (a: 'a list) : 'a list =
73   match a with
74   | [] -> raise Elem_pas_present
75   | y::q -> if (x=y) then q
76   else y::(suppr x q)

```

Code du TIPE

```

78  (* Fonction auxiliaire qui décide une égalité ensembliste entre a et b *)
79  let rec egal_ens_aux (a: 'a list) (b: 'a list) : bool =
80      match a with
81      | [] -> (b = [])
82      | x::q -> egal_ens_aux q (suppr x b)
83
84  let egal_ens (a: 'a list) (b:'a list) : bool =
85      try
86          egal_ens_aux a b
87      with
88      | Elem_pas_present -> false
89
90  (* Fin de l'implémentation des ensembles *)
91
92  (* Renvoie une formule équivalente à f, ne contenant plus aucun F, G
93  cf wikipedia pour les règles utilisées*)
94  let suppr_fg (phi : ltl) : ltl =
95      match phi with
96      | F psi -> U (Top, psi)
97      | G psi -> R (Bot, psi)
98      | _ -> phi
99
100  (* Cette fonction nous servira a transformé notre formule en
101  automate de büchi généralisé, en effet, on doit d'abord transformer
102  notre formule f en forme normal négative *)

```

Code du TIPE

```

104  (* Transforme une formule f en une formule f' équivalente
105  mise sous forme normale négative *)
106  let rec fnn (psi:ltl) : ltl =
107      match psi with
108      | Top | Bot | Atom _ -> psi
109      | Or (psi1, psi2) -> Or (fnn psi1, fnn psi2)
110      | And (psi1, psi2) -> And (fnn psi1, fnn psi2)
111      | U (psi1, psi2) -> U (fnn psi1, fnn psi2)
112      | R (psi1, psi2) -> R (fnn psi1, fnn psi2)
113      | X phi -> X (fnn phi)
114      | F _ | G _ -> fnn (suppr_fg psi)
115      | Not phi ->
116          (* On va ici appliqué les lois de De Morgan *)
117          (* Et les lois de dualités spécifiques à la logique LTL *)
118          begin
119              match phi with
120              | Top -> Bot
121              | Bot -> Top
122              | Atom _ -> Not phi
123              | Not psi2 -> (fnn psi2) (* Elimination de la double négation*)
124              | Or (psi1,psi2) -> And (fnn (Not psi1), fnn (Not psi2)) (* Loi de De Morgan *)
125              | And (psi1,psi2) -> Or (fnn (Not psi1), fnn (Not psi2))
126              | X psi2 -> X (fnn (Not psi2))
127              | F _ | G _ -> fnn (Not (suppr_fg phi))
128              | U (psi1, psi2) -> R (fnn (Not psi1), fnn (Not psi2))
129              | R (psi1, psi2) -> U (fnn (Not psi1), fnn (Not psi2))
130          end

```


Code du TIPE

```

132  (* On a ici des formules ne contenant pas de F, de G, et dont toutes les négations précèdent
    ↪  les littéraux
133  (mis sous forme normale négative) *)
134  let curr1 (phi: ltl) : ltl list =
135    match phi with
136    | U (psi1, psi2) -> [psi1]
137    | R (psi1, psi2) -> [psi2]
138    | Or (psi1, psi2) -> [psi2]
139    | _ -> []
140
141  let next1 (phi: ltl) : ltl list =
142    match phi with
143    | U (psi1, psi2) -> [phi]
144    | R (psi1, psi2) -> [phi]
145    | _ -> []
146
147  let curr2 (phi : ltl): ltl list =
148    match phi with
149    | U (psi1, psi2) -> [psi2]
150    | R (psi1, psi2) -> [psi1; psi2]
151    | Or (psi1, psi2) -> [psi1]
152    | _ -> []
153
154  let neg (phi : ltl) : ltl =
155    match phi with
156    | Top -> Bot
157    | Bot -> Top
158    | Atom p -> Not phi
159    | Not (Atom p) -> Atom p
160    | _ -> phi

```

Code du TIPE

```

162  (* Récupère l'ensemble d'éléments associé à un noeud dans set *)
163  let rec get_set (set : (noeud * ('a list)) list) (q:noeud) : 'a list =
164      match set with
165      | [] -> failwith "From get_set : Valeur non présente dans set"
166      | (x,l)::p -> if (x.name = q.name) then l else get_set p q
167
168  (* Renvoie incoming_u en version modifiée de tels manière que Incoming_u(q) := Incoming_u(q)
  ↪ union incoming *)
169  let rec edit_incoming (incoming_u : ((noeud * (noeud list)) list)) (incoming : noeud list)
  ↪ (q:noeud): (noeud * (noeud list)) list =
170      match incoming_u with
171      | [] -> [q,incoming]
172      | (x,l)::p -> if (x.name = q.name) then (q, union l incoming)::p
173      else (x,l)::(edit_incoming p incoming q)
174

```

Code du TIPE

```

175  (* Renvoie true si Il existe q appartient à nodeset / q.next=node.next et q.old=node.old et
    ↪ dans ce cas la : modifie
176  q.incoming par q.incoming union node.incoming
177  Sinon renvoie false*)
178  let rec exists (next_u : (noeud * (ltl list)) list) (next : ltl list) (now_u : (noeud * (ltl
    ↪ list)) list) (old: ltl list) (incoming: noeud list) (incoming_u : ((noeud * (noeud list))
    ↪ list) ref) (nodeset: noeud list) : bool=
179      match nodeset with
180      | [] -> false
181      | q::p -> let lnext = get_set next_u q in
182          let lnow = get_set now_u q in
183          if ((egal_ens lnext next) && (egal_ens lnow old)) then (
184              let incoming_bis = edit_incoming (!incoming_u) incoming q in
185              incoming_u := incoming_bis;
186              let _ = exists next_u next now_u old incoming incoming_u p in true
187          )
188      else exists next_u next now_u old incoming incoming_u p
189
190  (* Renvoie le plus grand nombre présent dans nodeset *)
191  let rec max_name (l: noeud list) =
192      match l with
193      | [] -> 0
194      | x::q -> max (int_of_string (x.name)) (max_name q)

```

Code du TIPE

```

196 let expand (curr: ltl list) (old : ltl list) (next : ltl list) (incoming : noeud list) =
197   let nodes_u = ref [] in
198   let incoming_u = ref [] in
199   let now_u = ref [] in
200   let next_u = ref [] in
201   let rec expand_aux (curr: ltl list) (old : ltl list) (next : ltl list) (incoming : noeud
    ↪ list) : unit =
202     if curr = [] then (
203       if (exists (!next_u) next (!now_u) old incoming incoming_u (!nodes_u)) then ()
204       else (
205         let q = {
206           name = string_of_int ((max_name (!nodes_u))+1)
207         } in
208         nodes_u := union (!nodes_u) [q];
209         incoming_u := union (!incoming_u) [(q, incoming)];
210         now_u := union (!now_u) [(q,old)];
211         next_u := union (!next_u) [(q,next)];
212         (expand_aux next [] [] [q])
213       )
214     )

```

Code du TIPE

```

215   else (
216     let f = List.hd curr in
217     let curr3 = List.tl curr in
218     let old3 = union old [f] in
219     match f with
220     | Top | Bot | Atom _ | Not (Atom _) ->
221       if ((f= Bot) || (appartient (neg f) old3)) then ()
222       else (expand_aux curr3 old3 next incoming)
223     | And (f1,f2) ->
224       (expand_aux (union curr3 (prive [f1;f2] old3)) old3 next incoming)
225     | X g ->
226       (expand_aux curr3 old3 (union next [g]) incoming)
227     | Or (f1,f2) | U (f1,f2) | R (f1,f2) ->
228       (expand_aux (union curr3 (prive (curr1 f) old3)) old3 (union next (next1 f))
229         ↪ incoming);
229       (expand_aux (union curr3 (prive (curr2 f) old3)) old3 next incoming)
230     | Not _ | G _ | F _ -> failwith "Formule non mise sous forme normale négative"
231   )
232   in (expand_aux curr old next incoming); (!nodes_u, !now_u, !incoming_u)

```

Code du TIPE

```

234 let create_graph (f : ltl) =
235   let init = {name = "init"} in
236   expand [f] [] [] [init]
237
238   (* Transforme une structure de kripke en un automate de buchi, (tout les états sont finaux)
239   On renomme ici les états par k_ (nom de l'état) afin de pouvoir différencier les états
   ↪   provenant d'une structure
   de kripke de ceux provenant d'un automate de buchi*)
240 let kripke_to_buchi (k: kripke_struct) : buchi_automate =
241   {q = List.map (fun x -> {etat = "k_" ^ x.etat}) (k.q);
242     init = List.map (fun x -> {etat = "k_" ^ x.etat}) (k.init);
243     transition = List.map (fun (x,y) -> ({etat = "k_" ^ x.etat}, {etat = "k_" ^ y.etat}))
   ↪   (k.transition);
244     final = List.map (fun x -> ({etat = "k_" ^ x.etat})) (k.q);
245     valuation = List.map (fun (x,f) -> ({etat = "k_" ^ x.etat}, f)) (k.valuation)}

```

Code du TIPE

```

248  (* Construit Trans = {(q,q') | q,q' appartient à Nodes and q appartient à Incoming(q')} *)
249  let rec build_transition (nodes: noeud list) (incoming: (noeud * (noeud list)) list) :
    ↪ (state*state) list =
250  (* Renvoi {(node,q') | node,q' appartient à Nodes and node appartient à l} *)
251  let rec build_transition_with_list (nodes : noeud list) (node : noeud) (l: noeud list) :
    ↪ (state*state) list =
252      match l with
253      | [] -> []
254      | q::p -> if (appartient q nodes) then ({etat = q.name}, {etat =
    ↪ node.name})::(build_transition_with_list nodes node p)
255      else (build_transition_with_list nodes node p)
256  in
257  match incoming with
258  | [] -> []
259  | (q',l)::p -> if ((appartient q' nodes)) then (build_transition_with_list nodes q'
    ↪ l)@(build_transition nodes p)
260  else (build_transition nodes p)
261
262  (* Construit Q0 = { q appartient à Nodes | init appartient à Incoming(q) } *)
263  let rec build_init (nodes : noeud list) (incoming: (noeud*(noeud list)) list) (init: noeud) :
    ↪ state list =
264      match incoming with
265      | [] -> []
266      | (q,l)::p -> if ((appartient init l) && (q.name <> "init")) then ({etat =
    ↪ q.name})::(build_init nodes p init)
267  else (build_init nodes p init) (* on exclus ici init avec la condition q.name <> "init" car
    ↪ on veut les q appartient à Nodes et init n'est pas
268      dans Nodes *)

```

Code du TIPE

```

270  (* Extrait l'ensemble des formules de now (qui est de type (noeud*ltl))* )
271  let rec extract_ltl (now: (noeud*ltl) list) : ltl list =
272      match now with
273      | [] -> []
274      | (q,f)::p -> let res = extract_ltl p in
275          if (appartient f res) then res
276          else f::res
277
278  (* On va maintenant construire les états finaux *)
279
280  (* Construit Fg = { q appartient à Nodes / g2 appartient à Now(q) or g n'appartient pas à
  ↪ Now(q) } où g = g1 U g2*)
281  let rec build_fg (nodes: noeud list) (now: (noeud* (ltl list)) list) (g1:ltl) (g2: ltl) :
  ↪ state list =
282      match now with
283      | [] -> []
284      | (q,l)::p -> let res = (build_fg nodes p g1 g2) in
285          if ((appartient g2 l || not (appartient (U (g1,g2)) l)) && (not (appartient {etat =
  ↪ q.name} res)))
286              then ({etat = q.name}::res)
287          else res
288
289  (* Détermine si g est une sous formule de f *)
290  let rec est_sous_formule (g: ltl) (f: ltl) : bool =
291      if (f=g) then true else
292      match f with
293      | Top | Bot | Atom _ -> (f=g) (*false donc*)
294      | Not f' | X f' | F f' | G f' -> est_sous_formule g f'
295      | Or (f1,f2) | And (f1,f2) | U (f1,f2) | R (f1,f2) -> (est_sous_formule g f1) ||
  ↪ (est_sous_formule g f2)

```


Code du TIPE

```

297  (* Extrait les sous formule de f de la forme phi U psi *)
298  let rec extrait_sous_formule (f: ltl) : ltl list =
299      match f with
300      | Atom p -> [U (Atom p, Atom p); U( Not (Atom p), Not (Atom p))]
301      | Top -> [U (Top, Top)]
302      | Bot -> []
303      | U (f1,f2) -> f::(union (extrait_sous_formule f1) (extrait_sous_formule f2))
304      | And (f1,f2) | Or (f1,f2) -> (union (extrait_sous_formule f1) (extrait_sous_formule f2))
305      | R (f1,f2) ->
306          (* R(f1,f2) équivaut à Not U (Not f1, Not f2) *)
307          f::(union (extrait_sous_formule (Not f1)) (extrait_sous_formule (Not f2)))
308      | X f1 | Not f1 -> extrait_sous_formule f1
309      | G _ | F _ -> failwith"formule non mise sous forme normale négative"
310
311  (* Transforme un ensemble de noeud en un ensemble d'états *)
312  let rec nodes_to_state (nodes : noeud list) : state list =
313      match nodes with
314      | [] -> []
315      | x::q -> {etat = x.name}::(nodes_to_state q)
316
317  let build_final (f: ltl) (nodes: noeud list) (now : (noeud* (ltl list)) list) : state list =
318      let rec aux (set : ltl list) : state list list =
319          (* Construit F = { Fg | g appartient à cl(f) } *)
320          match set with
321          | [] -> []
322          | U(g1,g2)::q -> (build_fg nodes now g1 g2)::(aux q)
323          | R(g1,g2)::q -> (build_fg nodes now (Not g1) (Not g2))::(aux q)
324          | phi::q -> failwith"ne devrait pas arriver"
325          in
326      List.fold_left (fun x y -> inter x y) (nodes_to_state nodes) (aux (extrait_sous_formule f))
327      (* On prend l'intersection de tout les ensembles obtenues avec aux *)

```

Code du TIPE

```

329  (* Extrait les littéraux d'un ensemble de couple (noeud, ensemble de formule) *)
330  let rec litteraux (q: noeud) (now: (noeud* (ltl list)) list) : ltl list =
331    let rec litteraux_aux (ltlset: ltl list) : ltl list =
332      (* Récupère tout les littéraux d'un ensemble de formule ltlset *)
333      match ltlset with
334      | [] -> []
335      | f::p -> let res = (litteraux_aux p) in
336        begin
337          match f with
338          | Atom p -> if (not (appartient f res)) then (Atom p)::res
339          | else res
340          | Not (Atom p) -> if (not (appartient f res)) then (Not (Atom p))::res
341          | else res
342          | _ -> res
343        end
344      in
345      match now with
346      | [] -> []
347      | (x,l)::p -> if (x.name = q.name) then (
348        (litteraux_aux l)@(litteraux q p)
349      )
350      | else (litteraux q p)

```

Code du TIPE

```

352  (* Construit la valuation pour l'automate de buchi *)
353  let rec build_valuation (nodes : noeud list) (now : (noeud* (ltl list)) list) : (state * (ltl
  ↪ list)) list =
354      match nodes with
355      | [] -> []
356      | q::p -> ({etat = q.name}, litteraux q now)::(build_valuation p now)
357
358  (* Construit l'ensemble d'états *)
359  let rec build_state (nodes : noeud list) : state list =
360      match nodes with
361      | [] -> []
362      | q::p -> {etat = q.name}::(build_state p)
363
364  (* Récupère le noeud initial dans nodes *)
365  let rec get_init (nodes : noeud list) : noeud =
366      match nodes with
367      | [] -> failwith "noeud initial non trouvé"
368      | x::q -> if (x.name = "init") then x
369      else get_init q

```

Code du TIPE

```

371 let build_lgba (f: ltl) : buchi_automate =
372   let init = {name = "init"} in
373   let (nodes,now,incoming) = create_graph f in
374   {
375     q = build_state nodes;
376     init = build_init nodes incoming init;
377     transition = build_transition nodes incoming ;
378     final = build_final f nodes now;
379     valuation = build_valuation nodes now
380   }
381
382   (* On va donc maintenant coder l'intersection entre deux automates de buchi, l'un d'eux sera
↪ une
structure de kripke qui aura été transformé en automate de buchi *)
383
384   (* Réalise l'opération  $q_1 \times q_2$  (produit cartésien des états) *)
385   let rec prod_states (q1: state list) (q2: state list) : (state*state) list =
386     let rec prod_cart (q: state) (l: state list) : (state*state) list =
387       match l with
388       | [] -> []
389       | y::p -> (q,y)::(prod_cart q p)
390     in
391     match q1 with
392     | [] -> []
393     | x::l -> let res = prod_states l q2 in
394               union (prod_cart x q2) res
395

```

Code du TIPE

```

397 |  (* Permet d'obtenir les états initiaux de l'automate produit,  $I = \{(q, q') \mid q \text{ appartient à } I1, q' \text{ appartient à } I2\}$  *)
    |  ↪ I1, q' appartient à I2} *)
398 |  (* On va donc naturellement utilisé la fonction précédente *)
399 |  let prod_init (b1 : buchi_automate) (b2 : buchi_automate) : (state*state) list =
400 |    prod_states (b1.init) (b2.init)
401 |
402 |  (* Un état (q,q') est final pour l'automate produit ssi q final pour b1 et q' final pour b2
    |  ↪ *)
403 |  let prod_final (b1: buchi_automate) (b2:buchi_automate) : (state*state) list =
404 |    prod_states (b1.final) (b2.final)
405 |
406 |  (* Récupère l'ensemble des transitions sortantes de q *)
407 |  let rec get_transition (q: state) (t: (state * state) list) : (state * state) list =
408 |    match t with
409 |    | [] -> []
410 |    | (x,y)::p -> if (x=q) then (x,y)::(get_transition q p)
411 |    else (get_transition q p)

```

Code du TIPE

```

413  (* Fusionne deux ensembles de transitions : exemple
414  fusion [(q1,q1); (q1;q2)] [(s0,s0);(s0;s1)] renverra
415  [((q1,s0),(q1,s0)); ((q1,s0),(q1,s1)) ; ((q1,s0),(q2,s0)) ; ((q1,s0), (q2,s1))]*
416  let rec fusion (t1: (state * state) list) (t2: (state * state) list) : ((state * state) *
↳ (state * state)) list =
417  (* idem mais avec un seul état *)
418  let rec fusion_bis (q: (state * state)) (t: (state * state) list) : ((state * state) *
↳ (state * state)) list =
419      let (q1,q2) = q in
420      match t with
421      | [] -> []
422      | (x,y)::p -> ((q1,x),(q2,y))::(fusion_bis q p)
423  in
424  match t1 with
425  | [] -> []
426  | q::p -> union (fusion_bis q t2) (fusion p t2)

```

Code du TIPE

```

428  (* Récupère l'ensemble des étiquettes de notre automate de buchi b *)
429  let get_alphabet (b:buchi_automate) : ltl list =
430    (* récupère les littéraux de la forme Atom "p" d'un ensemble de formule fset *)
431    let rec get_litteral (fset : ltl list) : ltl list =
432      match fset with
433      | [] -> []
434      | (Atom p)::l | (Not (Atom p))::l -> (Atom p)::(get_litteral l)
435      | _::l -> get_litteral l
436    in
437    let rec aux_alphabet (l: (state * (ltl list)) list) : ltl list =
438      match l with
439      | [] -> []
440      | (q,fset)::p -> (get_litteral fset)@(aux_alphabet p)
441    in aux_alphabet (b.valuation)
442
443  (* Récupère l'ensemble des formules atomiques de la forme Not (Atom "p") dans fset (On
444  ↪ renverra une version épurée sans le Not) *)
445  let rec get_neg (fset: ltl list) : ltl list =
446    match fset with
447    | [] -> []
448    | (Not (Atom p))::q -> (Atom p)::(get_neg q)
449    | _::q -> get_neg q

```

Code du TIPE

```

451  (* Renvoie une version modifié de fset, si fset contient une formule de la forme Not (Atom
↪ "p"), alors on renvoie
452  alphabet\{Atom "p"} *)
453  let rec modifie_ltlset (fset : ltl list) (alphabet : ltl list) : ltl list =
454    prive alphabet (get_neg fset)
455
456  (* Detecte si un ensemble de formule contient une absurdité du type [Not (Atom "p"); (Atom
↪ "p")] *)
457  let rec absurde (fset : ltl list) : bool =
458    match fset with
459    | [] -> false
460    | (Atom p)::q -> if (appartient (Not (Atom p)) fset) then true
461    else absurde q
462    | (Not (Atom p))::q -> if (appartient (Atom p) fset) then true
463    else absurde q
464    (* On a ici pas besoin de "regarder en arrière" dans la liste car on la parcourt de gauche a
↪ droite *)
465    | f::q -> absurde q

```


Code du TIPE

```

467  (* Réalise la fonction de transition pour l'automate produit *)
468  let rec prod_transition (b1: buchi_automate) (b2:buchi_automate) : ((state * state) * (state
↳ * state)) list =
469    let states = prod_states (b1.q) (b2.q) in
470    let res = ref [] in
471    let alphabet1 = get_alphabet b1 in let alphabet2 = get_alphabet b2 in
472    let alpha = union alphabet1 alphabet2 in
473    List.iter (fun (q,q') ->
474      (* Rappel : valuation correspond à l'étiquetage des transitions sortants d'un état, ainsi
↳ si
475      deux transitions (q1,q2) et (q1',q2') correspondent en valuation_dans_autom1(q1) =
↳ valuation_dans_autom2(q2)
476      alors on peut ajouter la transition ((q1,q1'),(q2,q2')) *)
477      let v1 = List.assoc q (b1.valuation) in let v2 = List.assoc q' (b2.valuation) in
478      (* Il y a ici un problème, lorsque la valuation étiquette (Not (Atom "p")), cela
479      signifie que n'importe quelle lettre différente de p permet de prendre la transition *)
480      let v1' = modifie_ltlset v1 alpha in let v2' = modifie_ltlset v2 alpha in
481      let union_v1v2 = union v1 v2 in
482      if ((inter v1' v2') <> [] && (not (absurde union_v1v2))) then (
483        let t1 = get_transition q (b1.transition) in
484        let t2 = get_transition q' (b2.transition) in
485        let fus = fusion t1 t2 in
486        res := union (!res) fus
487      )
488    else ()) states;
489  !res

```

Code du TIPE

```

491  (* Réalise la fonction valuation pour l'automate produit *)
492  let rec prod_valuation (b1: buchi_automate) (b2:buchi_automate) : ((state * state) * (ltl
  → list)) list =
493    let states = prod_states (b1.q) (b2.q) in
494    let res = ref [] in let alphabet1 = get_alphabet b1 in let alphabet2 = get_alphabet b2 in
495    let alpha = union alphabet1 alphabet2 in
496    List.iter (
497      fun (q,q') ->
498        let v1 = List.assoc q (b1.valuation) in
499        let v2 = List.assoc q' (b2.valuation) in
500        let v1' = modifie_ltlset v1 alpha in let v2' = modifie_ltlset v2 alpha in
501        let union_v1v2 = union v1 v2 in
502        if ((inter v1' v2') <> [] && not (absurde union_v1v2)) then (
503          res := union (!res) [((q,q'),(union_v1v2))]
504        )
505        else (res := union (!res) [((q,q'),[])])
506    ) states;
507    !res
508
509  type buchi_automate_product = {
510    q : (state * state) list;
511    init : (state * state) list;
512    transition : ((state * state) * (state * state)) list;
513    final : (state * state) list;
514    valuation : ((state * state) * (ltl list)) list
515  }

```

Code du TIPE

```

517 let create_product (b1:buchi_automate) (b2:buchi_automate) : buchi_automate_product =
518 {
519   q = prod_states (b1.q) (b2.q);
520   init = prod_init b1 b2;
521   transition = prod_transition b1 b2;
522   final = prod_final b1 b2;
523   valuation = prod_valuation b1 b2;
524 }
525
526 (* Début de l'emptiness check *)
527
528 (* Fonction donné par le doc : https://www.cs.cmu.edu/~15414/f17/lectures/18-ltl.pdf *)
529 (* Récupère l'ensemble des états atteints en un coup depuis q avec t(transition) *)
530 let rec get_state_set (q: ((state*state))) (t: ((state * state)*(state * state)) list) :
    ↪ (state*state) list =
531   match t with
532   | [] -> []
533   | (x,y)::p -> if (x=q) then y::(get_state_set q p)
534   else (get_state_set q p)
535
536 exception Found

```

Code du TIPE

```

538 let rec innerdfs (q: (state*state)) (outervisited : (state*state) list) (innervisited :
↳ (state*state) list) (b: buchi_automate_product) : unit =
539   let innervisited' = q::innervisited in
540   List.iter (fun q' -> if (appartient q' outervisited) then (raise Found)
541     else if (not (appartient q' innervisited' )) then (innerdfs q' outervisited innervisited'
↳ b)) (get_state_set q (b.transition))
542
543 let rec outerdfs (q:(state*state)) (visited : (state*state) list) (b: buchi_automate_product)
↳ : unit =
544   let visited' = q::visited in
545   List.iter (fun q' -> if (not (appartient q' visited')) then (outerdfs q' visited' b))
↳ (get_state_set q (b.transition));
546   if (appartient q (b.final)) then (innerdfs q visited' [] b)
547   else ()
548
549 let is_empty (b: buchi_automate_product) : bool =
550   let res = ref true in
551   List.iter (fun q ->
552     try
553       outerdfs q [] b
554     with
555       | Found -> res := false
556     ) (b.init);
557   !res
558
559 let model_checker (k: kripke_struct) (f: ltl) : bool =
560   let b2 = kripke_to_buchi k in
561   let b1 = build_lgba (fnn (Not f)) in
562   let product = create_product b1 b2 in
563   ((is_empty product))

```

Langage jouet

```

<program> ::= "début" <listinstr> "fin"

<listinstr> ::=
    <instr> <listinstr>
  | <instr>

<instr> ::=
    "entier" <var> ";"
  | <var> " <- " <expr> ";"
  | "Tant que" "(" <condition> ")" "faire" <listinstr> "fin Tant
que" ";"
  | "Si" "(" <condition> ")" "faire" <listinstr> "fin Si" ";"
  | "Retourner" <expr>

<expr> ::=
    <expr> "+" <expr>
  | <expr> "-" <expr>
  | <expr> "*" <expr>
  | "(" <expr> ")"
  | <expr0>

<expr0> ::=
    <var>
  | <int>
  | "-" <int>

<int> ::=
    <chiffre> <int>
  | <chiffre>

<chiffre> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<condition> ::= <expr> <compsymbol> <expr>

<compsymbol> ::= ">" | "<" | ">=" | "<=" | "!=" | "="

```

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'  $\leftarrow$  ajouter  $q$  à visités
2 pour tout  $q' \in \delta(q)$  faire
3   si  $q' \notin \textit{visités'}$  alors
4     // Appel récursif
4     outerdfs( $q'$ , visités',  $A$ )
5 si  $q \in F$  alors
6   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  $visitesIn' \leftarrow$  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)

Algorithme 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
```

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

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

Exemple d'une transformation vers FNN

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

Exemple d'une transformation vers FNN

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

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

Exemple d'une transformation vers FNN

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

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



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

Exemple d'une transformation vers FNN

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

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

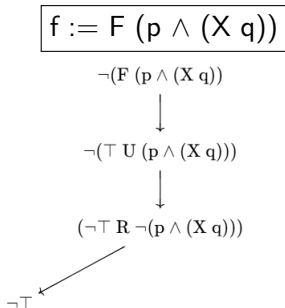


$$\neg(\top U (p \wedge (X q)))$$

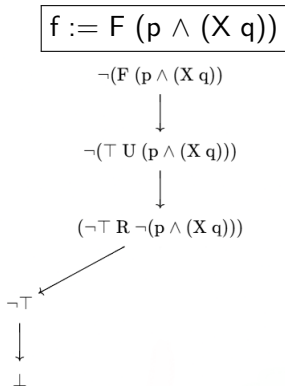


$$(\neg \top R \neg(p \wedge (X q)))$$

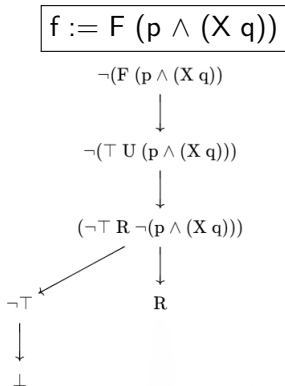
Exemple d'une transformation vers FNN



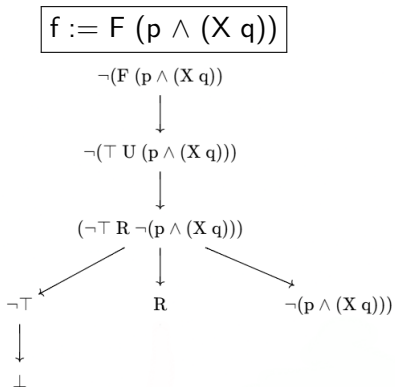
Exemple d'une transformation vers FNN



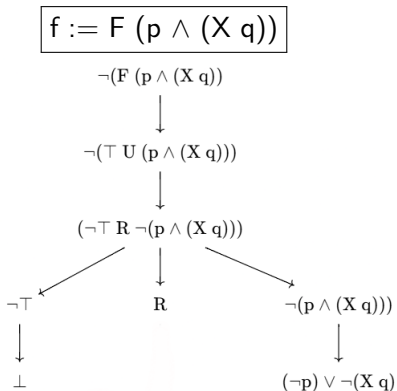
Exemple d'une transformation vers FNN



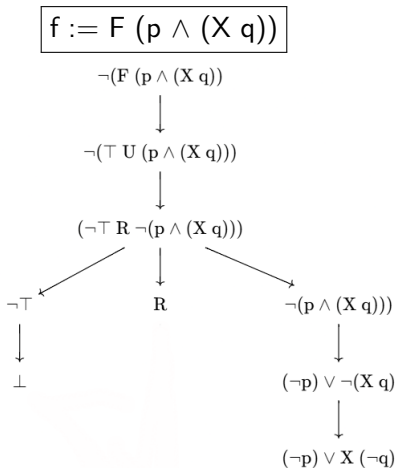
Exemple d'une transformation vers FNN



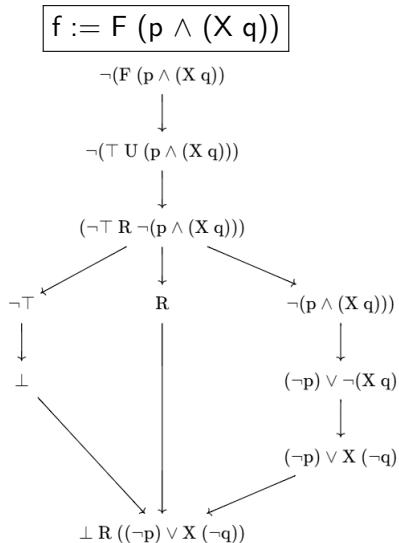
Exemple d'une transformation vers FNN



Exemple d'une transformation vers FNN



Exemple d'une transformation vers FNN



Exemple d'une transformation vers FNN

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

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

Ensembles en ocaml

 $A \setminus B$ $A \cup B$ $A \cap B$ $a \in A$ $A \subseteq B$ $A = B$

```
1 | prive : 'a list -> 'a list -> 'a list
2 | union : 'a list -> 'a list -> 'a list
3 | inter : 'a list -> 'a list -> 'a list
4 | appartient : 'a -> 'a list -> bool
5 | inclus : 'a list -> 'a list -> bool
6 | exception Elem_pas_present
7 | suppr : 'a -> 'a list -> 'a list
8 | egal_ens : 'a list -> 'a list -> bool
```