

ltl2ba

Un compilateur de formules LTL en automate de Büchi généralisés

Emile ROLLEY Thomas MORIN

Université de Bordeaux

12 mai 2022

Automates de Büchi sur *les transitions*

Automates de Büchi sur *les transitions*

Même définition que pour un automate de Büchi généralisé :

$$\mathcal{A} = (S, \rightarrow, S_0, F_1, \dots, F_l) \text{ avec } \forall i \in \{1, \dots, l\}, F_i \subseteq \rightarrow$$

Automates de Büchi sur *les transitions*

Même définition que pour un automate de Büchi généralisé :

$$\mathcal{A} = (S, \rightarrow, S_0, F_1, \dots, F_l) \text{ avec } \forall i \in \{1, \dots, l\}, F_i \subseteq \rightarrow$$

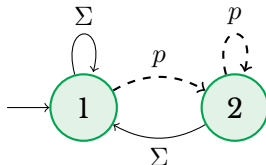


Figure 1: Exemple d'automate reconnaissant la formule LTL GFp , avec en pointillé, les transitions appartenant à l'unique condition d'acceptation.

L'algorithme de traduction

Intuition

Diviser la formule de départ φ en sous-formules plus simple (dites *réduites*) et ajouter une condition d'acceptation pour chaque sous-formule de la forme $\alpha \cup \beta$.

L'algorithme de traduction

Intuition

Diviser la formule de départ φ en sous-formules plus simple (dites *réduites*) et ajouter une condition d'acceptation pour chaque sous-formule de la forme $\alpha \cup \beta$.

Étapes

1. Mise en forme normale négative de φ .
2. $S_0 = \{\varphi\}$.
3. Pour chaque état Y dans S :
 - ▶ Calculer un graphe orienté temporaire \mathcal{G}_Y .
 - ▶ Ajouter dans \mathcal{A} les transitions et les nouveaux états correspondants grâce à \mathcal{G}_Y .

L'algorithme de traduction

L'algorithme de traduction

Définition (*NNF*)

Une formule est en **forme normale négative** (*NNF*) si elle est constituée uniquement des sous-formules suivantes :

- ▶ \perp, p et $\neg p$ avec $p \in AP$
- ▶ $\exists x \alpha$ et $\alpha * \beta$ avec $*$ $\in \{U, R, V, \wedge\}$

L'algorithme de traduction

Définition (*NNF*)

Une formule est en **forme normale négative** (*NNF*) si elle est constituée uniquement des sous-formules suivantes :

- ▶ \perp, p et $\neg p$ avec $p \in AP$
- ▶ $X\alpha$ et $\alpha * \beta$ avec $*$ $\in \{U, R, V, \wedge\}$

Définition (*ensemble réduit*)

Un ensemble de formules Z est **réduit** si :

- ▶ toutes les formules de Z sont **réduites**, c'est-à-dire, de la forme $p, \neg p$ ou $X\alpha$ avec $p \in AP$
- ▶ $\perp \notin Z$, et $\{p, \neg p\} \not\subseteq Z$ pour tout $p \in AP$.

L'algorithme de traduction

L'algorithme de traduction

Calcul de \mathcal{G}_Y

Soit $Y = Z \cup \{\alpha\}$ où α n'est pas réduite et si possible maximale (càd. n'est sous-formule d'aucune autre formule non réduite de Y). Les arêtes à partir de Y sont :

- ▶ Si $\alpha = \alpha_1 \vee \alpha_2$, $Y \rightarrow Z \cup \{\alpha_1\}$ et $Y \rightarrow Z \cup \{\alpha_2\}$.
- ▶ Si $\alpha = \alpha_1 \wedge \alpha_2$, $Y \rightarrow Z \cup \{\alpha_1, \alpha_2\}$
- ▶ Si $\alpha = \alpha_1 \text{ R } \alpha_2$, $Y \rightarrow Z \cup \{\alpha_1, \alpha_2\}$ et $Y \rightarrow Z \cup \{X\alpha, \alpha_2\}$.
- ▶ Si $\alpha = \alpha_1 \cup \alpha_2$, $Y \rightarrow Z \cup \{\alpha_2\}$ et $Y \rightarrow^\alpha Z \cup \{X\alpha, \alpha_1\}$.

L'algorithme de traduction

Calcul de \mathcal{G}_Y

Soit $Y = Z \cup \{\alpha\}$ où α n'est pas réduite et si possible maximale (càd. n'est sous-formule d'aucune autre formule non réduite de Y). Les arêtes à partir de Y sont :

- ▶ Si $\alpha = \alpha_1 \vee \alpha_2$, $Y \rightarrow Z \cup \{\alpha_1\}$ et $Y \rightarrow Z \cup \{\alpha_2\}$.
- ▶ Si $\alpha = \alpha_1 \wedge \alpha_2$, $Y \rightarrow Z \cup \{\alpha_1, \alpha_2\}$
- ▶ Si $\alpha = \alpha_1 \text{ R } \alpha_2$, $Y \rightarrow Z \cup \{\alpha_1, \alpha_2\}$ et $Y \rightarrow Z \cup \{X\alpha, \alpha_2\}$.
- ▶ Si $\alpha = \alpha_1 \cup \alpha_2$, $Y \rightarrow Z \cup \{\alpha_2\}$ et $Y \rightarrow^\alpha Z \cup \{X\alpha, \alpha_1\}$.

Cette construction est appliquée récursivement jusqu'à ce que toutes les feuilles du graphe soient réduites.

L'algorithme de traduction

L'algorithme de traduction

Calcul des transitions à partir de Y

Finalement, une fois \mathcal{G}_Y calculé, sont ajoutées dans \mathcal{A} :

- ▶ les transitions suivantes $\{Y \rightarrow^{\Sigma_Z} \text{next}(Z) \mid Z \in \text{Red}(Y)\}$
- ▶ pour chaque sous-formule $\alpha = \alpha_1 \cup \alpha_2$, les conditions d'acceptations $F_\alpha = \{Y \rightarrow^{\Sigma_Z} \text{next}(Z) \mid Y \in S, Z \in \text{Red}_\alpha(Y)\}$

L'algorithme de traduction

Calcul des transitions à partir de Y

Finalement, une fois \mathcal{G}_Y calculé, sont ajoutées dans \mathcal{A} :

- ▶ les transitions suivantes $\{Y \rightarrow^{\Sigma_Z} \text{next}(Z) \mid Z \in \text{Red}(Y)\}$
- ▶ pour chaque sous-formule $\alpha = \alpha_1 \cup \alpha_2$, les conditions d'acceptations $F_\alpha = \{Y \rightarrow^{\Sigma_Z} \text{next}(Z) \mid Y \in S, Z \in \text{Red}_\alpha(Y)\}$

Avec,

$$\text{Red}(Y) = \{Z \text{ réduit} \mid Y \rightarrow^* Z\}$$

$$\text{Red}_\alpha(Y) = \{Z \text{ réduit} \mid Y \rightarrow^{*\setminus\alpha} Z\}$$

$$\text{next}(Z) = \{\alpha \mid X\alpha \in Z\}$$

$$\Sigma_Z = \bigcap_{p \in Z} \Sigma_p \cap \bigcap_{\neg p \in Z} \Sigma_{\neg p}$$

Un exemple *comparé* pour $\varphi = p \cup q$

Algorithme classique

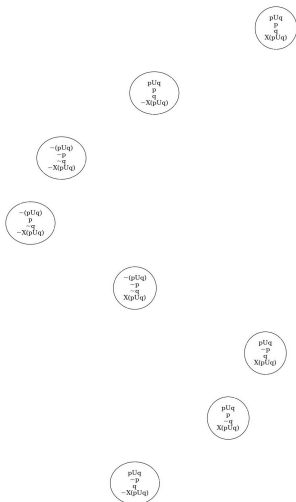
On commence par calculer la clôture de la formule :

$cl(\varphi) = \{p \cup q ; \neg(p \cup q) ; X(p \cup q) ; \neg(X(p \cup q)) ; p ; \neg p ; q ; \neg q\}$;

$cl(\varphi)$ est constitué de 8 formules (4 formules et leurs négations).

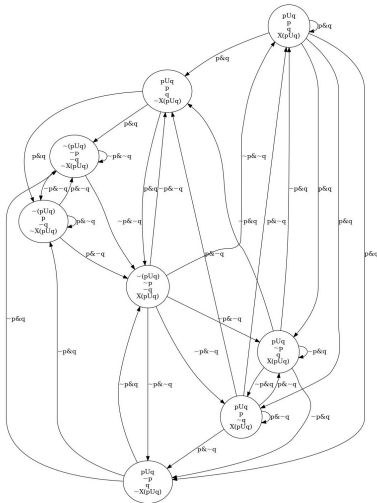
Un exemple *comparé* pour $\varphi = p \cup q$

On calcule ainsi les états consistants suivants :



Un exemple *comparé* pour $\varphi = p \cup q$

On a ainsi l'automate suivant :

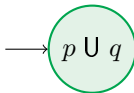


A peine sale ...

Un exemple *comparé* pour $\varphi = p \cup q$

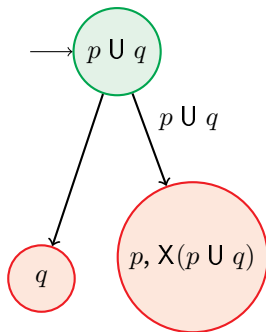
Algorithme Malin

On commence par mettre l'état initial : c'est la formule actuelle :



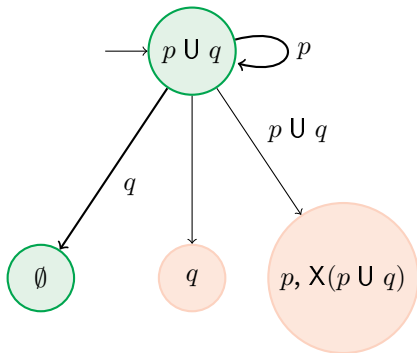
Un exemple *comparé* pour $\varphi = p \cup q$

On construit ensuite le graphe temporeire pour l'état à considérer :



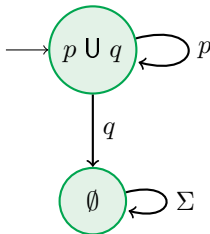
Un exemple *comparé* pour $\varphi = p \cup q$

On ajoute ainsi les vrais états du graphe, en vert, à partir des états réduits accessibles :



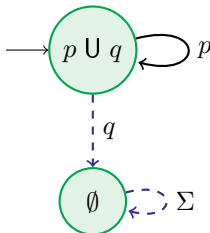
Un exemple *comparé* pour $\varphi = p \cup q$

On a un autre état à considérer, l'état vide. Comme il est réduit et qu'il ne contient pas d'état, il boucle sur lui même :



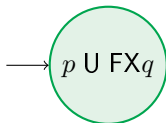
Un exemple *comparé* pour $\varphi = p \cup q$

On retire les états temporaires. Comme il y a un Until, il faut ajouter un ensemble de transitions d'acceptations, en bleu et pointillé :



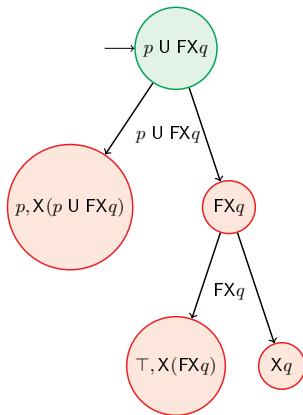
(Un autre exemple pour $\varphi = p \cup FXq$)

Comme d'habitude, on commence par placer l'état initial, qui est φ :



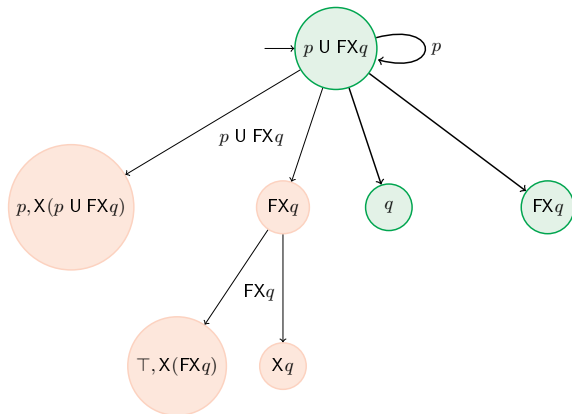
(Un autre exemple pour $\varphi = p \cup \text{FX}q$)

La deuxième étape est toujours la même, faire le graphe temporaire pour cet état accessible :



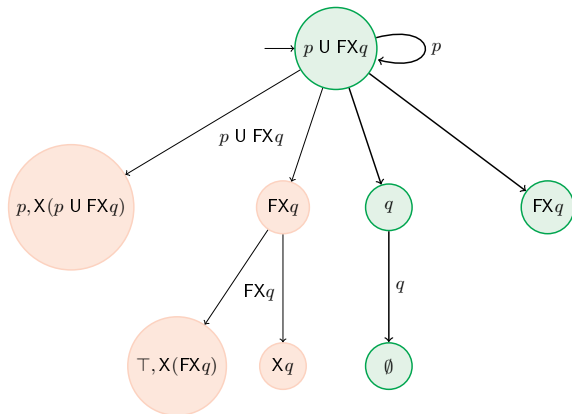
(Un autre exemple pour $\varphi = p \cup FXq$)

La troisième étape consiste en créer les nouveaux états accessibles de l'automates à partir des états réduits du graphe temporaire :



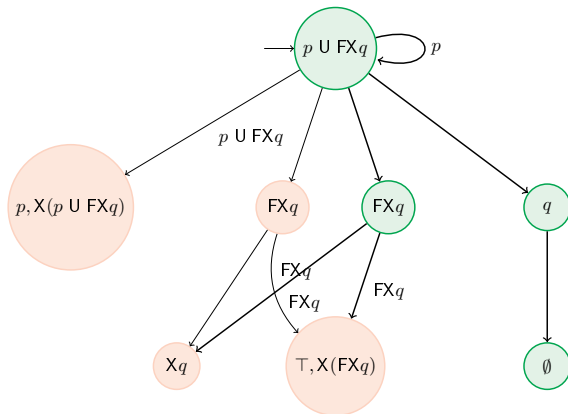
(Un autre exemple pour $\varphi = p \cup \text{FX}q$)

On a de nouveaux états accessibles, il faut les considérer un par un. Comme l'état $\{q\}$ est déjà réduit, il ne crée aucun état intermédiaire, et permet d'accéder à un nouvel état :



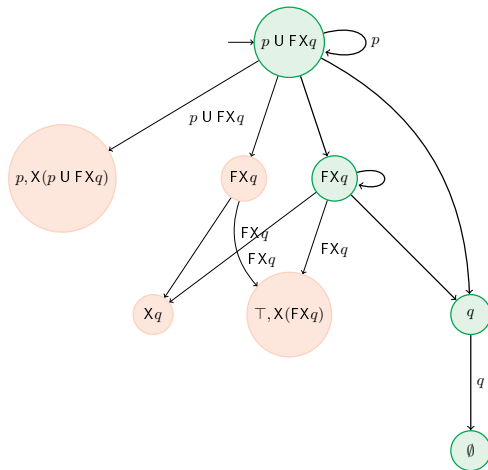
(Un autre exemple pour $\varphi = p \cup \text{FX}q$)

On considère maintenant l'état $\{ \text{FX}q \}$, qui crée un nouveau graphe intermédiaire :



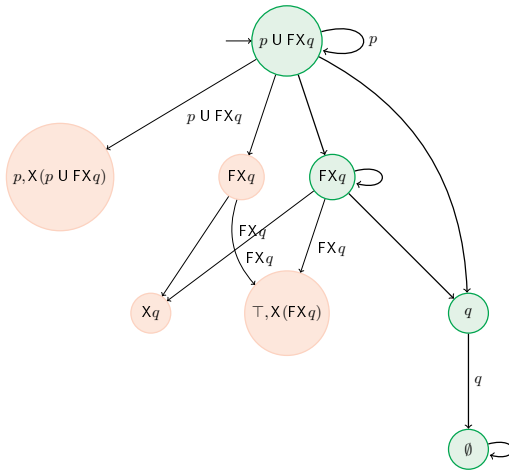
(Un autre exemple pour $\varphi = p \cup \text{FX}q$)

... et ainsi les nouvelles transitions suivantes :



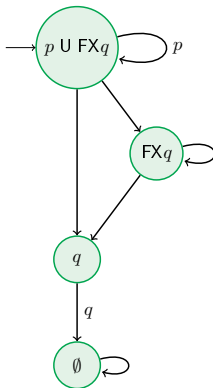
(Un autre exemple pour $\varphi = p \cup \text{FX}q$)

Il reste enfin le dernier état, vide. Il est réduit, on boucle :



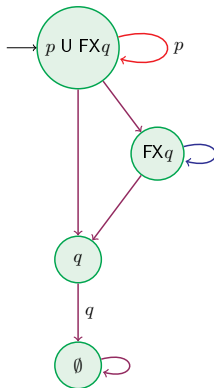
(Un autre exemple pour $\varphi = p \cup \text{FX}q$)

On a terminé. On retire les états temporaires :



(Un autre exemple pour $\varphi = p \cup \text{FX}q$)

On a deux formules Until. On ajoute donc deux ensembles d'états acceptants :



L'implémentation d'Emile

```
ltl2ba.docs-slides~?+7-0 ocaml(v4.12.0, ltl2ba*) 2s
» ltl2ba "p U FXq" -d test.dot && xdot test.dot
--- ltl2ba v0.2.0 ---
[OK]       $\varphi := (p \cup FXq)$ 
[OK] NNF( $\varphi$ ) :=  $(p \cup (\tau \cup Xq))$ 
[OK] Automata successfully printed in 'test.dot'
```

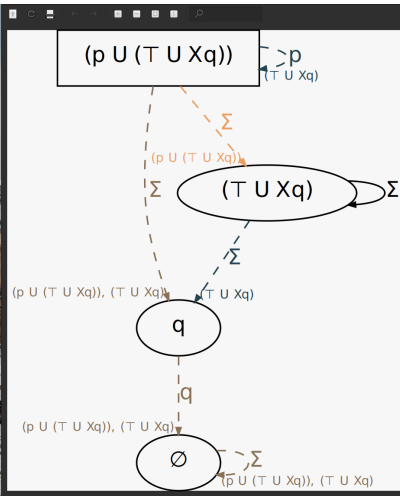


Figure 2: Exemple d'exécution pour la formule $\varphi = p \cup FXq$

L'implémentation d'Emile

Principales caractéristiques :

- ▶ Est implémenté en OCaml.
- ▶ Le parseur utilise `ocamllex` et `menhir`.
- ▶ Une soixantaine de test unitaires sont effectués avec `alcotest` et exécutés en intégration continue avec GitHub.
- ▶ Ne calcule pas le graphe intermédiaire mais directement l'ensemble des feuilles réduites.

L'implémentation d'Emile

En particulier, la fonction `val red : state -> red_states` retourne un record de type :

```
type red_states =  
  { all : StateSet.t  
    ; marked_by : StateSet.t FormulaMap.t  
  }
```

Où `all` représente l'ensemble des états réduits accessible en utilisant **aucune** arête marquée et `marked_by` est une map avec comme clé les formules de la forme $\alpha = \alpha_1 \cup \alpha_2$ et en valeur l'ensemble des états réduits accessible en utilisant **uniquement** des arêtes marquées par α .

L'implémentation d'Emile

En particulier, la fonction `val red : state -> red_states` retourne un record de type :

```
type red_states =  
  { all : StateSet.t  
    ; marked_by : StateSet.t FormulaMap.t  
  }
```

Où `all` représente l'ensemble des états réduits accessible en utilisant **aucune** arête marquée et `marked_by` est une map avec comme clé les formules de la forme $\alpha = \alpha_1 \cup \alpha_2$ et en valeur l'ensemble des états réduits accessible en utilisant **uniquement** des arêtes marquées par α .

Donc finalement :

$$\text{Red}(Y) = \text{all} \cup \bigcup_{\alpha \in \text{marked_by.keys}} \text{marked_by}[\alpha]$$
$$\text{Red}_\alpha(Y) = \text{Red}(Y) \setminus \text{marked_by}[\alpha]$$

L'implémentation d'Emile

Principales caractéristiques :

- ▶ Est implémenté en OCaml.
- ▶ Le parseur utilise `ocamllex` et `menhir`.
- ▶ Une soixantaine de test unitaires sont effectués avec `alcotest` et exécutés en intégration continue avec `GitHub`.
- ▶ Ne calcule pas le graphe intermédiaire mais directement l'ensemble des feuilles réduites.
- ▶ L'automate est défini comme un graphe orienté et utilise `ocamlgraph` pour l'implémentation.

L'implémentation d'Emile

```
module TransBuchi : sig
  include
    Graph.Sig.I
    with type V.t =
      [ `Init of state
      | `Normal of state
      ]
    and type E.label =
      [ `Normal of FormulaSet.t
      | `Acceptant of Ltl.formula list * FormulaSet.t
      ]
end
```