Université Libre de Bruxelles

INFO-F302
Informatique Fondamentale

Rapport Informatique Fondamentale

Étudiants :
Hugo Callens
Rayan Contuliano Bravo
Ethan Rogge

Enseignants :
E. Filliot
R. Petit

15 décembre 2023

1 Modélisation d'un automate déterministe en FNC

Voici quelques notations utilisées dans la suite de ce rapport :

- P représente l'ensemble des mots acceptés par l'automate
- N représente l'ensemble des mots non-acceptés par l'automate
- Σ représente l'alphabet de l'automate
- k représente le nombre au plus d'états de l'automate

1.1 Choix des variables

1.1.1 Etats

L'ensemble Q est défini comme suit : $Q = \{q_{1a}, q_{1na}, \ldots, q_{ka}, q_{kna} | k\}$ contenant ainsi $2k$ états dans Q pour un automate à k états. Les variables q_i peuvent être définies comme suit :

- $q_{ia/na} = 1$ si l'état i acceptant/non-acceptant est présent dans l'automate.
- $q_{ia/na} = 0$ si l'état i acceptant/non-acceptant n'est pas présent dans l'automate.

Cette implémentation permet de représenter un automate avec au plus k états. En effet, si l'état i n'est pas présent dans l'automate, alors, les variables q_{ia} et q_{ina} sont toutes les deux égales à 0.

1.1.2 Transitions

L'ensemble Δ est défini comme suit : $\Delta = \{d_{i,l,j} | i, j \in Q, l \in \Sigma\}$, représentant l'ensemble des transitions de l'automate. Les variables $d_{i,l,j}$ peuvent être définies comme suit :

- $d_{i,l,j} = 1$ si la transition avec la lettre l existe entre l'état i et l'état j.
- $d_{i,l,j} = 0$ si la transition avec la lettre l n'existe pas entre l'état i et l'état j.

1.1.3 Exécutions

L'ensemble E est défini comme suit : $E = \{e_{m,i,t} | i \in Q, t \in \{0, \ldots, \text{len}(m) - 1\}, m \in P \cup N\}$, représentant l'ensemble des exécutions de l'automate. Les variables $e_{m,i,t}$ peuvent être définies comme suit :

- $e_{m,i,t} = 1$ si l'automate est dans l'état i après avoir lu les t premières lettres du mot m.
- $e_{m,i,t} = 0$ si l'automate n'est pas dans l'état i après avoir lu les t premières lettres du mot m.

1.2 Contraintes

1. Il y a un unique état initial :
$$\Phi_1 = q_{0a} \vee q_{0na}$$
2. Un état est exclusivement acceptant ou non-acceptant :
$$\Phi_2 = \bigwedge_{\substack{q \in Q \\ i \in \{1,\ldots,k\}}} q_{ia} \rightarrow \neg q_{ina} \equiv \bigwedge_{\substack{q \in Q \\ i \in \{1,\ldots,k\}}} \neg q_{ia} \vee \neg q_{ina}$$
3. Chaque état a au plus une transition par lettre de l'alphabet :
$$\Phi_3 = \bigwedge_{\substack{l \in \Sigma \\ i,j,q \in Q}} d_{i,l,j} \rightarrow \neg d_{i,l,q} \equiv \bigwedge_{\substack{l \in \Sigma \\ i,j,q \in Q}} \neg d_{i,l,j} \vee \neg d_{i,l,q}$$
4. Un état i est acceptant s'il existe une exécution d'un mot m de P qui se termine sur l'état

i à l'étape $t = len(m)$:

$$\Phi_4 = \bigwedge_{\substack{t=len(m)\\q\in Q\\m\in P\\i\in\{1,\dots,k\}}} e_{m,i,t} \rightarrow q_{ia} \equiv \bigwedge_{\substack{t=len(m)\\q\in Q\\m\in P\\i\in\{1,\dots,k\}}} \neg e_{m,i,t} \vee q_{ia}$$

5. Si on a une exécution qui est sur l'état i pour l'étape t du mot m et une transition de i vers j pour la lettre l, alors, on a une exécution pour le mot m à l'étape $t+1$ sur l'état j :

$$\Phi_5 = \bigwedge_{\substack{i,j\in Q\\l\in\Sigma\\m\in P\cup N\\t\in\{0,\dots,len(m)\}\\l=m[t]}} (e_{m,i,t} \wedge d_{i,l,j} \rightarrow e_{m,j,t+1}) \equiv \bigwedge_{\substack{i,j\in Q\\l\in\Sigma\\m\in P\cup N\\t\in\{0,\dots,len(m)\}\\l=m[t]}} (\neg e_{m,i,t} \vee \neg d_{i,l,j} \vee e_{m,j,t+1})$$

6. Si on a une exécution qui est sur l'état i pour l'étape t du mot m et une exécution à l'étape $t+1$ pour le même mot m qui se trouve sur l'état j, alors, il existe une transition de i vers j avec la lettre l :

$$\Phi_6 = \bigwedge_{\substack{i,j\in Q\\l\in\Sigma\\m\in P\cup N\\t\in\{0,\dots,len(m)\}\\l=m[t]}} (e_{m,i,t} \wedge e_{m,j,t+1} \rightarrow d_{i,l,j}) \equiv \bigwedge_{\substack{i,j\in Q\\l\in\Sigma\\m\in P\cup N\\t\in\{0,\dots,len(m)\}\\l=m[t]}} (\neg e_{m,i,t} \vee \neg e_{m,j,t+1} \vee d_{i,l,j})$$

7. Toutes les exécutions, qui existent, sur les mots de N doivent se terminer sur un état non-acceptant :

$$\Phi_7 = \bigwedge_{\substack{m\in N\\i\in\{1,\dots,k\}\\t=len(m)\\q\in Q}} e_{m,i,t} \rightarrow q_{ina} \equiv \bigwedge_{\substack{m\in N\\i\in\{1,\dots,k\}\\t=len(m)\\q\in Q}} \neg e_{m,i,t} \vee q_{ina}$$

8. Toutes les exécutions sur les mots de P doivent exister :

$$\Phi_8 = \bigwedge\nolimits_{m\in P} \bigvee_{\substack{t\in\{0,\dots,len(word)\}\\i\in\{1,\dots,k\}}} e_{m,i,t+1}$$

9. Toutes les exécutions doivent commencer à l'état initial :

$$\Phi_9 = \bigwedge_{m\in P\cup N} e_{m,1,0}$$

La formule en logique propositionelle finale est :

$$\Phi = \bigwedge_{i=1,\dots,9} \Phi_i$$

2 Algorithme de minimisation d'un automate déterministe

Afin de minimiser le nombre d'états d'un automate fini sur Σ, une approche dichotomique est utilisée en considérant :

1. Recherche de la borne supérieure, donc un k tel qu'avec 2^k états, la solution est trouvée.
2. Recherche dichotomique entre $2^{(k-1)}$ et 2^k pour trouver le k minimal pour lequel la solution est trouvée.

Il suffit d'utiliser les contraintes précédemment définies (cf. 1.2) afin de trouver l'automate minimal en nombre d'états.

La réduction d'un automate fini à k états se fait dans le pire des cas en un automate à k états. Cependant, il est question ici de générer un automate minimal et non pas de réduire un automate fini. C'est pourquoi une variable ϵ a été introduite, l'algorithme développé va tester pour un maximum de 2^{10} états s'il trouve un automate. En effet, il se peut que la méthode `get_model()` ne renvoie jamais un automate si les paramètres donnés n'aboutissent pas à la création d'un automate déterministe.

3 Modélisation d'un automate déterministe complet en FNC

Il suffit de reprendre les contraintes précédentes (cf. 1.2) et d'ajouter la contrainte suivante :

1. Tous les états doivent avoir au moins une transition par lettre de l'alphabet :

$$\bigwedge_{l\in\Sigma}\bigvee_{i,j\in Q} d_{i,l,j}$$

Du fait que nous avons au moins une transition par lettre de l'alphabet et au plus une transition par lettre de l'alphabet, nous avons donc exactement une transition par lettre de l'alphabet pour chaque état, ce qui implique que l'automate est complet.

4 Modélisation d'un automate réversible en FNC

Il suffit de rajouter aux contraintes précédentes (cf. 1.2) la contrainte suivante :

1. S'il existe une transition de l'état i vers l'état j pour la lettre l, alors il n'existe pas de transition de l'état k vers l'état j pour la lettre l :

$$\bigwedge_{\substack{i,j,k\in Q\\ l\in\Sigma}} d_{i,l,j} \rightarrow \neg d_{k,l,j} \equiv \bigwedge_{\substack{i,j,k\in Q\\ l\in\Sigma}} \neg d_{i,l,j} \vee \neg d_{k,l,j}$$

FIGURE 4.1 – A gauche nous voyons un automate non-réversible et à droite nous voyons un automate réversible

5 Modélisation d'un automate fini complet avec au plus l états en FNC

Pour que l'automate fini complet ait au plus l états, il suffit de rajouter aux contraintes précédentes (cf. 3) la contrainte suivante (suivant le formalisme de la documentation de PySAT) :

1. Il y a au plus l états acceptants :

$$\sum_{i=0}^{k} q_{ia} \leq l$$

Cette contrainte a été ajoutée au solveur `MiniCard`.

6 Modélisation d'un automate non déterministe en FNC

La principale différence entre un automate non-déterministe et un automate déterministe est qu'il n'y a plus un unique état j accessible depuis un état i après la lecture d'une lettre l mais bien un ensemble d'états J accessibles depuis un état i après la lecture d'une lettre l. Les variables de transition deviennent donc des relations.
De ce fait, les variables de transitions changent de définition :

$$\Delta = \{d_{i,l,J} | i \in Q, J \subseteq Q, l \in \Sigma\}$$

La contrainte (Φ_3) qui émpêche d'avoir plusieurs transitions pour une même lettre l depuis un état i est donc supprimée. (cf. 1.2)
Après plusieurs essais, nous ne voyons pas en quoi les contraintes précédemment définies ne permettent pas au solveur de trouver une solution en NFA étant donné qu'on supprime les contraintes sur les transitions.

7 Modélisation d'un automate avec au plus k états et k' transitions en FNC

En suivant le raisonnement de la question 6, il suffit de rajouter une contrainte au solver `MiniCard` sur le nombre de transitions et faire en sorte que celui-ci ne dépasse pas k'.