UNIVERSITÉ LIBRE DE BRUXELLES

Informatique Fondamentale

Projet – Apprentissage automatique de programmes de validation de chaînes de caractères

Issa Kevin, 514550

Charels Hugo , 544051

Nieto Navarrete Matias , 502920

B-INFO

15 decembre 2023

Table des matières

1. Introduction

Dans ce rapport [1], nous explorons diverses problématiques en rapport avec la conception d'automates et la logique propositionnelle. Nous cherchons à créer des automates sous contraintes spécifiques, telles que le nombre d'états limité, la consistance avec des ensembles d'exemples positifs P et négatifs N. Pour chaque problème posé, nous employons des variables booléennes afin de modéliser les transitions et les états des automates, ainsi que les exécutions des mots. Nous formulons des expressions en logique propositionnelle et les soumettons à un solveur SAT pour déterminer si un automate répondant aux critères existe et le crée.

1.1 Variables

Construction de l'Automate

- p_x : représente l'appartenance de l'état x à l'automate. Un état x est considéré comme présent dans l'automate si et seulement si p_x est vrai.
- a_x : détermine si l'état x est un état acceptant. Un mot est reconnu par l'automate si et seulement si le dernier état visité est un état acceptant, soit a_x est vrai pour cet état.
- $t_{x,y,l}$: caractérise les transitions de l'automate. La variable est vraie lorsque l'automate effectue une transition de l'état x à l'état y en lisant la lettre l.

Exécution sur l'Automate

- $v_{x,i,w}$: représente la visite de l'état x à la position i du mot w lors de son traitement par l'automate. Cette variable est essentielle lors de l'exécution d'un mot.

1.1.1 Transformation de Tseitin

Transformation de Tseitin

- $x_{x,s,w}$: Utilisée dans la contrainte 2 de la consistance.
- $y_{x,y,i,w}$: Utilisée dans la contrainte 5 de la consistance.

Nous utiliserons ces variables de façon systématique pour établir la formule φ_{AUT} en forme normale conjonctive (FNC), qui servira à prouver l'existence d'un automate conforme aux critères établis.

1. Nous avons fait appel à ChatGPT pour améliorer la syntaxe, corriger les fautes d'orthographe et améliorer le LaTeX.

2. Analyse Détaillée des Questions

Question 1

Dans cette question, nous sommes chargés de fournir une formule de logique propositionnelle satisfaisable si et seulement si il existe un automate fini A avec au plus k états, tel que $P \subseteq L(A)$ et $L(A) \cap N = \emptyset$ où P représente l'ensemble des mots acceptant de A et N représente l'ensemble des mots rejetant de A.

Contraintes de Cohérence

Dans la définition de l'automate fini A, nous établissons les contraintes de cohérence suivantes :

Contrainte 1

La source est dans l'automate (par défaut la source est l'état 0) : p_0

Contrainte 2

Tous les états acceptants font partie de l'automate :

— Description : Pour tous états x, si x est un état acceptant alors x est dans l'automate.

— Formule initiale :

$$\bigwedge_{x=0}^{k-1} a_x \rightarrow p_x$$

— Mise en forme normale conjonctive (FNC) :

$$\bigwedge_{x=0}^{k-1} \neg a_x \vee p_x$$

Contrainte 3

Toutes les transitions sont valides :

— Description : S'il existe une transition x, y, l qui va de l'état x à l'état y avec la lettre l, alors les états x et y sont dans l'automate.

— Formule initiale :

$$\bigwedge_{x=0}^{k-1} \bigwedge_{y=0}^{k-1} \bigwedge_{l \in \Sigma} t_{x,y,l} \rightarrow p_x \wedge p_y$$

— Transformation en FNC : On retire l'implication

$$\bigwedge_{x=0}^{k-1} \bigwedge_{y=0}^{k-1} \bigwedge_{l \in \Sigma} \neg t_{x,y,l} \vee (p_x \wedge p_y)$$

On distribue le terme de gauche de la disjonction sur la conjonction et nous obtenons cette contrainte en FNC :

$$\bigwedge_{x=0}^{k-1} \bigwedge_{y=0}^{k-1} \bigwedge_{l \in \Sigma} (\neg t_{x,y,l} \vee p_x) \wedge (\neg t_{x,y,l} \vee p_y)$$

Contraintes de Consistance

Les contraintes de consistance garantissent que l'automate A est consistant avec les ensembles P et N, respectant ainsi les conditions $P \subseteq L(A)$ et $L(A) \cap N = \emptyset$.

Contrainte 1

Toutes les exécutions commencent uniquement à la source (p_0) :

— Description : Chaque mot $w \in P \cup N$ visite la source et aucun autre état en premier

— Formule initiale en FNC :

$$\bigwedge_{w \in P \cup N} v_{0,0,w} \wedge \bigwedge_{x=1}^{k-1} \neg v_{x,0,w}$$

Contrainte 2

Pour tous les mots de P, il existe une exécution acceptante :

— Description : Il existe un état x qui est visité à la dernière position dans le mot $w \in P$ et que cet état x soit acceptant.

— Formule initiale :

$$\bigwedge_{w \in P} \bigvee_{x=0}^{k-1} v_{x,|w|,w} \wedge a_x$$

— Mise en forme normale conjonctive (FNC) : Afin de transformer cette formule en FNC, il est préférable d'utiliser la transformation de Tseitin. Cette transformation implique, dans ce cas, de séparer la double implication en deux implications simples. Ensuite, il faut les éliminer et distribuer l'élément de gauche de la disjonction sur la conjonction de la sous-formule de gauche où les implications ont été retirées.

$$\bigwedge_{w \in P} \left(\bigvee_{x=0}^{k-1} x_{x,|w|,w} \right) \wedge \left(\bigwedge_{x=0}^{k-1} x_{x,|w|,w} \leftrightarrow (v_{x,|w|,w} \wedge a_x) \right)$$

$$\bigwedge_{w \in P} \left(\bigvee_{x=0}^{k-1} x_{x,|w|,w} \right) \wedge \left(\bigwedge_{x=0}^{k-1} \left(x_{x,|w|,w} \rightarrow (v_{x,|w|,w} \wedge a_x) \right) \wedge \left((v_{x,|w|,w} \wedge a_x) \rightarrow x_{x,|w|,w} \right) \right)$$

$$\bigwedge_{w \in P} \left(\bigvee_{x=0}^{k-1} x_{x,|w|,w} \right) \wedge \left(\bigwedge_{x=0}^{k-1} \left(\neg x_{x,|w|,w} \vee (v_{x,|w|,w} \wedge a_x) \right) \wedge \left(\neg v_{x,|w|,w} \vee \neg a_x \vee x_{x,|w|,w} \right) \right)$$

$$\bigwedge_{w \in P} \left(\bigvee_{x=0}^{k-1} x_{x,|w|,w} \right) \wedge \left(\bigwedge_{x=0}^{k-1} (\neg x_{x,|w|,w} \vee v_{x,|w|,w}) \wedge (\neg x_{x,|w|,w} \vee a_x) \wedge \left(\neg v_{x,|w|,w} \vee \neg a_x \vee x_{x,|w|,w} \right) \right)$$

Contrainte 3

Pour tous les mots de N, il n'existe pas d'exécution acceptante :

— Description : Il n'existe pas d'état x qui est visité à la dernière position dans le mot $w \in N$ et que cet état x soit acceptant.

— Formule initiale :

$$\bigwedge_{w \in P} \neg \bigvee_{n=0}^{k-1} v_{x,|w|,w} \wedge a_x$$

— Mise en forme normale conjonctive (FNC) : Il faut appliquer la loi de Morgan d'abord à la grande disjonction, puis à la petite conjonction. Cela nous donne la formule suivante :

$$\bigwedge_{w \in N} \bigwedge_{n=0}^{k-1} \neg v_{x,|w|,w} \vee \neg a_x$$

Contrainte 4

Une exécution est une suite d'état visité par les transitions :

— Description : Si l'état x est visité à la $i$éme lettre du mot w alors s'il existe une transition qui va de l'état x vers un état y avec la $i$éme lettre du mot w alors l'état y est visité à la $i+1$ème lettre du mot w

— Formule initiale :

$$\bigwedge_{w \in P \cup N} \bigwedge_{i=0}^{|w|-1} \bigwedge_{x=0}^{k-1} v_{x,i,w} \rightarrow \bigwedge_{y=0}^{k-1} t_{x,y,w[i]} \rightarrow v_{y,i+1,w}$$

— Mise en forme normale conjonctive (FNC) : Il faut d'abord éliminer les deux implications, puis rentrée la variable dans la grande conjonction.

$$\bigwedge_{w \in P \cup N} \bigwedge_{i=0}^{|w|-1} \bigwedge_{x=0}^{k-1} \neg v_{x,i,w} \vee \bigwedge_{y=0}^{k-1} \neg t_{x,y,w[i]} \vee v_{y,i+1,w}$$

$$\bigwedge_{w \in P \cup N} \bigwedge_{i=0}^{|w|-1} \bigwedge_{x=0}^{k-1} \bigwedge_{y=0}^{k-1} \neg v_{x,i,w} \vee \neg t_{x,y,w[i]} \vee v_{y,i+1,w}$$

Contrainte 5

Un état ne peut être visité que s'il existe un état précédent qui est également visité et qu'il y a une transition qui lie les deux états.

— Description : Si l'état x est visité à la $i+1$éme lettre du mot w alors il existe une transition qui va d'un état y vers l'état x avec la $i$éme lettre du mot w et l'état y est visité à la $i$éme lettre du mot w

— Formule initiale :

$$\bigwedge_{w\in P\cup N}\bigwedge_{i=0}^{|w|-1}\bigwedge_{x=0}^{k-1} v_{x,i+1,w} \rightarrow \bigvee_{y=0}^{k-1}(t_{y,x,w[i]} \wedge v_{y,i,w})$$

— Mise en forme normale conjonctive (FNC) : On commence par éliminer l'implication. Ensuite, on déplace la variable v dans la grande disjonction. Et nous obtenons la formule suivante :

$$\bigwedge_{w\in P\cup N}\bigwedge_{i=0}^{|w|-1}\bigwedge_{x=0}^{k-1}\bigvee_{y=0}^{k-1} \neg v_{x,i+1,w} \vee (t_{y,x,w[i]} \wedge v_{y,i,w})$$

Par la suite, on applique la transformation de Tseitin. Dans ce cas, cela implique de décomposer la double implication en deux implications simples, lesquelles sont ensuite éliminées. Finalement, on effectue la distribution pour obtenir la formule sous FNC.

$$\bigwedge_{w\in P\cup N}\bigwedge_{i=0}^{|w|-1}\bigwedge_{x=0}^{k-1}\left(\bigvee_{y=0}^{k-1} \neg v_{x,i+1,w} \vee y_{x,y,i,w}\right) \wedge \left(\bigwedge_{y=0}^{k-1} y_{x,y,i,w} \leftrightarrow (t_{y,x,w[i]} \wedge v_{y,i,w})\right)$$

$$\bigwedge_{w\in P\cup N}\bigwedge_{i=0}^{|w|-1}\bigwedge_{x=0}^{k-1}\left(\bigvee_{y=0}^{k-1} \neg v_{x,i+1,w} \vee y_{x,y,i,w}\right) \wedge \left(\bigwedge_{y=0}^{k-1}(y_{x,y,i,w} \rightarrow (t_{y,x,w[i]} \wedge v_{y,i,w})) \wedge ((t_{y,x,w[i]} \wedge v_{y,i,w}) \rightarrow y_{x,y,i,w})\right)$$

$$\bigwedge_{w\in P\cup N}\bigwedge_{i=0}^{|w|-1}\bigwedge_{x=0}^{k-1}\left(\bigvee_{y=0}^{k-1} \neg v_{x,i+1,w} \vee y_{x,y,i,w}\right) \wedge \left(\bigwedge_{y=0}^{k-1}(\neg y_{x,y,i,w} \vee (t_{y,x,w[i]} \wedge v_{y,i,w})) \wedge (\neg t_{y,x,w[i]} \vee \neg v_{y,i,w} \vee y_{x,y,i,w})\right)$$

$$\bigwedge_{w\in P\cup N}\bigwedge_{i=0}^{|w|-1}\bigwedge_{x=0}^{k-1}\left(\bigvee_{y=0}^{k-1} \neg v_{x,i+1,w} \vee y_{x,y,i,w}\right) \wedge \left(\bigwedge_{y=0}^{k-1}(\neg y_{x,y,i,w} \vee t_{y,x,w[i]}) \wedge (\neg y_{x,y,i,w} \vee v_{y,i,w}) \wedge (\neg t_{y,x,w[i]} \vee \neg v_{y,i,w} \vee y_{x,y,i,w})\right)$$

Contraintes de Déterminisme

Contrainte 1

Les transitions sont unique :

— Description : S'il y a une transition qui va de l'état x à l'état y avec la lettre l alors il n'existe pas de transition qui va de ce même état x avec la même lettre l vers un état z différent de y.

— Formule initiale :

$$\bigwedge_{l\in\Sigma}\bigwedge_{x=0}^{k-1}\bigwedge_{y=0}^{k-1} t_{x,y,l} \rightarrow \neg \bigvee_{z=0,y\neq z}^{k-1} t_{x,z,l}$$

— Transformation en FNC : La première étape du processus consiste à éliminer les implications. Ensuite, on applique la loi de Morgan. Pour finir, il suffit de déplacer la variable dans la grande conjonction.

$$\bigwedge_{l\in\Sigma}\bigwedge_{x=0}^{k-1}\bigwedge_{y=0}^{k-1} \neg t_{x,y,l} \vee \neg \bigvee_{z=0,y\neq z}^{k-1} t_{x,z,l}$$

$$\bigwedge_{l\in\Sigma}\bigwedge_{x=0}^{k-1}\bigwedge_{y=0}^{k-1} \neg t_{x,y,l} \vee \bigwedge_{z=0,y\neq z}^{k-1} \neg t_{x,z,l}$$

$$\bigwedge_{l\in\Sigma}\bigwedge_{x=0}^{k-1}\bigwedge_{y=0}^{k-1}\bigwedge_{z=0,y\neq z}^{k-1} \neg t_{x,y,l} \vee \neg t_{x,z,l}$$

Note : Les contraintes de cohérence, de consistance et de déterminisme détaillées ci-dessus sont applicables à presque toutes les questions suivantes et seront référencées en conséquence.

Question 2

Suite à notre analyse précédente, nous avons développé une fonction, nommée `gen_aut`, pour générer un automate fini déterministe (DFA) sur un alphabet donné Σ et qui est consistant avec des ensembles de mots positifs P et négatifs N, tout en respectant une contrainte sur le nombre maximal d'états k. Cette fonction utilise la librairie PySAT pour résoudre un problème SAT correspondant à la construction de l'automate.

Description de l'Algorithme `gen_aut`

L'algorithme implémenté dans `gen_aut` suit les étapes suivantes :

1. **Construction d'un ensemble de clauses :** Utilisant la classe `DetAutGenerator`, l'algorithme génère un ensemble de clauses SAT représentant les contraintes de cohérence, de consistance et de déterminisme de l'automate. Ces contraintes assurent que l'automate est correctement construit et qu'il accepte tous les mots dans P et rejette ceux dans N, tout en ne dépassant pas k états.

2. **Appel d'un solveur SAT :** Le programme utilise ensuite un solveur SAT, tel que `Minisat`, pour déterminer si les clauses peuvent être satisfaites, c'est-à-dire s'il est possible de construire un automate répondant aux critères donnés.

3. **Retour de l'automate ou de None :** Si le solveur trouve une solution, l'algorithme utilise la classe `AutBuilder` pour construire l'automate fini correspondant à partir de la valuation trouvée par le solveur. Si aucune solution n'est trouvée, la fonction retourne `None`, indiquant qu'aucun automate répondant aux critères n'existe.

Question 3

Pour répondre à cette question, nous avons dû développer un algorithme capable de générer un automate fini déterministe (DFA) minimal en termes de nombre d'états pour un alphabet Σ. Cet automate doit être consistant avec les ensembles de mots P (positifs) et N (négatifs).

Implémentation de `gen_minaut`

— **Génération de l'automate :** La classe `MinAutGenerator` étend les fonctionnalités de `DetAutGenerator` en initialisant la valeur de k et en appliquant une recherche dichotomique pour trouver le nombre minimal d'états nécessaires à l'automate.

— **Construction de l'automate :** Après avoir trouvé un modèle satisfaisant, la fonction `build` de la classe `AutBuilder` est appelée pour construire le DFA à partir de ce modèle.

Approche Algorithmique

L'approche algorithmique [2] mise en œuvre dans `MinAutGenerator` consiste à :

1. Utiliser la méthode `_check_bounds_when_small` pour estimer un nombre initial d'états k en augmentant exponentiellement sa valeur. Cette étape s'arrête lorsque la génération d'un automate est satisfaisante ou que la limite $k_{\max}$ est atteinte. Cette limites est calculée en faisant la somme de la taille de tout les mots de P et N.
2. Affiner cette estimation à l'aide de la méthode `_check_bounds`, qui implémente une recherche dichotomique pour déterminer le plus petit k qui satisfait les contraintes. Cette recherche dichotomique est effectuée entre la moitié de la limite supérieure trouvée et cette limite.
3. À chaque étape de k, formuler la génération de l'automate comme un problème de satisfaction de contraintes booléennes (SAT), en tenant compte des mots acceptés et rejetés, ainsi que des contraintes de cohérence, consistance et déterminisme.
4. Utiliser un solveur SAT pour tester si un modèle satisfaisant existe pour la valeur actuelle de k. Si un tel modèle est trouvé, l'automate correspondant est construit. Sinon, la recherche continue avec une nouvelle valeur de k déterminée par la recherche dichotomique.

Cette méthode assure que l'automate généré est le plus petit possible en termes de nombre d'états, tout en étant consistant avec les ensembles de mots P et N.

2. L'approche utilisée est inspirée de la correction de l'exercice 1.3 du cours d'Algorithmique 2.

Question 4

Dans le prolongement dans la question 1 et 2, nous voulons créer un automate complet, s'il existe. Pour cela, la fonction `gen_autc` a été conçue pour générer un automate fini déterministe (DFA) complet sur un alphabet Σ, consistant avec les ensembles de mots positifs P et négatifs N, tout en ne dépassant pas un nombre maximal d'états k. Cette fonction étend la logique précédente en ajoutant une contrainte de complétude à l'automate.

Description de l'Algorithme `gen_autc`

La fonction `gen_autc` utilise la classe `CompAutGenerator` pour générer un ensemble de clauses SAT qui respectent non seulement les contraintes de cohérence, de consistance et de déterminisme mais aussi la contrainte de complétude. Le solveur SAT est ensuite utilisé pour trouver un modèle qui satisfait ces clauses. Si une solution est trouvée, l'automate est construit à l'aide de la classe `AutBuilder` ; sinon, la fonction retourne `None`.

Contrainte de Complétude

La contrainte de complétude impose que pour chaque état de l'automate, il doit exister une transition sortante pour chaque lettre de l'alphabet Σ.

Cette contrainte est formulée comme suit :

$$\bigwedge_{l \in \Sigma} \bigwedge_{x=0}^{k-1} \bigvee_{y=0}^{k-1} t_{x,y,l}$$

Question 5

Dans la suite des questions 1 et 2, une nouvelle exigence est introduite pour la fonction `gen_autr` : l'automate retourné doit être réversible. Cela signifie que sa fonction de transition reste une fonction même lorsque les transitions sont inversées, permettant ainsi de "rembobiner" de manière déterministe ses exécutions.

Description de l'Algorithme `gen_autr`

L'algorithme `gen_autr` utilise la classe `RevAutGenerator` pour générer un ensemble de clauses SAT respectant les contraintes de cohérence, de consistance et de déterminisme et en particulier la contrainte de réversibilité. Si le solveur SAT trouve une solution satisfaisant ces clauses, l'automate est construit à l'aide de la classe `AutBuilder` ; sinon, `None` est retourné.

Contrainte de Réversibilité

Les transitions inversées sont unique :

— Description : S'il y a une transition qui va de l'état x à l'état y avec la lettre l alors il n'existe pas de transition qui va d'un état z différent de x avec la même lettre l vers l'état y.

— Formule initiale :

$$\bigwedge_{l\in\Sigma}\bigwedge_{x=0}^{k-1}\bigwedge_{y=0}^{k-1} t_{x,y,l} \rightarrow \neg \bigvee_{z=0,z\neq x}^{k-1} t_{z,y,l}$$

— Transformation en FNC : La première étape du processus consiste à éliminer l'implication. Ensuite, on applique la loi de Morgan. Pour finir, il suffit de déplacer la variable dans la grande conjonction.

$$\bigwedge_{l\in\Sigma}\bigwedge_{x=0}^{k-1}\bigwedge_{y=0}^{k-1} \neg t_{x,y,l} \vee \neg \bigvee_{z=0,z\neq x}^{k-1} t_{z,y,l}$$

$$\bigwedge_{l\in\Sigma}\bigwedge_{x=0}^{k-1}\bigwedge_{y=0}^{k-1} \neg t_{x,y,l} \vee \bigwedge_{z=0,z\neq x}^{k-1} \neg t_{z,y,l}$$

$$\bigwedge_{l\in\Sigma}\bigwedge_{x=0}^{k-1}\bigwedge_{y=0}^{k-1}\bigwedge_{z=0,z\neq x}^{k-1} \neg t_{x,y,l} \vee \neg t_{z,y,l}$$

Elle assure qu'il existe au plus un état q' tel que $\delta(q', \sigma) = q$ pour tout état q et toute lettre σ, garantissant ainsi le déterminisme de l'automate avec ces transitions inversées.

Question 6

Pour répondre à cette question, nous utilisons la fonction `gen_autcard` afin de construire un automate fini complet avec une contrainte supplémentaire sur le nombre d'états acceptants. Cette contrainte est représentée par un entier l, limitant le nombre maximum d'états acceptants dans l'automate.

Implémentation de `gen_autcard`

L'implémentation de `gen_autcard` se fait en deux étapes principales :

— **Génération de l'automate :** Utilisant la classe `CardAutGenerator`, l'automate est généré en respectant les contraintes de cardinalité. Cette classe généralise `DetAutGenerator` en ajoutant la contrainte que le nombre d'états acceptants doit être au plus l. Cette contrainte est intégrée en utilisant la classe `CNFPlus`, qui permet l'ajout direct de contraintes de cardinalité.

— **Construction de l'automate :** Si la génération est réussie, la fonction `build` de la classe `AutBuilder` est appelée pour construire l'automate à partir du modèle trouvé. En cas d'échec, la fonction renvoie `None`.

Contrainte de Cardinalité

La contrainte est exprimée par la formule suivante, qui assure que le nombre d'états acceptants ne dépasse pas l :

$$\left|\{x|\forall x \in \{0,\ldots,k-1\} : a_x\}\right| \leq l$$

Cette formule garantit que, parmi les états générés, au plus l sont marqués comme états acceptants.

Les autres contraintes de cohérence, de consistance et de déterminisme définies à la question 1 sont également maintenues pour assurer la validité de l'automate généré.

Question 7

Cette question demande la création d'un automate non déterministe (NFA) en fonction d'un alphabet donné, de listes de mots acceptants et rejetant, et d'une contrainte sur le nombre d'états de l'automate. La fonction `gen_autn` est conçue pour répondre à cette exigence.

Implémentation de `gen_autn`

L'implémentation se décompose comme suit :

— **Génération de l'automate :** La classe `AutGenerator` est utilisée pour générer un ensemble de clauses SAT respectant les contraintes de cohérence, de consistance et la contrainte du nombre d'états `k`. Puis les clauses sont fournies au solver SAT pour atteindre l'objectif fixé.

— **Construction de l'automate :** Si la génération est fructueuse, la fonction `build` de la classe `AutBuilder` est utilisée pour créer le NFA à partir du modèle SAT retourné. En cas d'échec, la fonction renvoie `None`.

Contraintes pour la Génération de NFA

Les contraintes principales utilisées pour générer le NFA sont :

— **Contraintes de cohérence :** Ces contraintes s'assurent que l'automate est correctement construit, en vérifiant la présence d'un état initial et la validité des transitions et des états acceptants.

— **Contraintes de consistance :** Elles garantissent que l'automate accepte tous les mots dans `pos` et rejette ceux dans `neg`. Elles s'assurent également de la validité des chemins d'exécution au sein de l'automate.

Question 8 - BONUS

Afin de construire un automate avec un nombre de transition maximale, il nous suffit de nous baser sur la question 6 et d'appliquer le même procédé à la seule différence que cette fois-ci, nous établissons une contrainte sur le nombre de transition maximale au lieu du nombre d'états acceptants maximaux.

Contrainte de Cardinalité des Transitions

La contrainte est formulée comme suit :

$$\left|\{(x, y, l) \,|\, \forall x, y \in \{0, \ldots, k-1\}, \forall l \in \Sigma : t_{x,y,l}\}\right| \leq k'$$

Cette formule garantit que le nombre total de transition dans l'automate est au plus k'.