GRAMMAIRE FORMELLE APPLIQUÉE À L'OPTIMISATION

Revue des différentes techniques pour utiliser des automates et des grammaires non contextuelles dans des problèmes d'optimisation.

Rédigé par

ALEXANDRE DAIGLE

alexandre@adaigle.com

Université Laval
Québec

Résumé

Ce rapport vise à faire une revue de littérature sur le problème de la reconnaissance d'une chaîne de caractères par une grammaire formelle et à expliciter un modèle qui effectue cette reconnaissance en $O(n^{\log_2 7}) \approx O(n^{2.807})$. Il sera question de deux types de langages, d'abord, les langages réguliers, définis par un automate fini déterministe, et, par la suite, de grammaires non contextuelles, qui permettent de représenter des langages plus riches que le premier. Cela permettra d'introduire les contraintes REGULAR et GRAMMAR qui sont utilisées en optimisation combinatoire. Il est décrit les techniques utilisées pour définir ces contraintes et pour en faire le filtrage afin de les utiliser dans des solveurs. Par la suite, une réduction du problème de reconnaissance d'une chaîne de n caractères par une grammaire non contextuelle est effectuée vers le problème de multiplication de matrices $n \times n$. La réduction inverse permettra de montrer que les deux problèmes sont de complexité équivalente. Finalement, on énonce un modèle mathématique qui est une amélioration des algorithmes traditionnels de reconnaissance par une grammaire.

Table des matières

1 Préliminaires

Avant de s'attaquer au vif du sujet, il s'impose de définir les notions d'optimisation combinatoire, d'automates et de grammaires non contextuelles dont nous aurons besoin. Cette première section permet de poser ces bases.

1.1 Optimisation combinatoire

Les éléments introduits dans ce rapport sont en général utilisés dans des problèmes d'optimisation, il est donc nécessaire de bien poser cette catégorie de problème.

Définition Un *problème d'optimisation* consiste à définir des variables $X_1, \ldots, X_n$ de domaines donnés $dom(X_1), \ldots, dom(X_n)$ et une série de contraintes. On souhaite trouver une affectation de valeurs aux variables de façon à respecter les domaines et les contraintes. On peut définir une *fonction objectif* que l'on souhaite maximiser ou minimiser. En l'absence de fonction objectif, on parle plutôt de *problème de satisfaction* et l'on cherche n'importe quelle solution qui respecte les contraintes.

La définition précédente donne la structure d'un problème d'optimisation. En pratique, on ajoute ces variables et ces contraintes dans un solveur qui explore l'arbre de recherche automatiquement en effectuant tous les cas possibles. Dans le cas général, l'arbre de recherche est de taille exponentielle et le problème est $\mathcal{NP}$*-complet*. Toutefois, des heuristiques et du filtrage efficace permettent de résoudre une grande quantité de ces problèmes.

Définition Le *filtrage* est une technique utilisée par les solveurs qui permet de couper des branches entières de l'arbre de recherche lorsque les contraintes permettent d'éliminer la possibilité qu'une solution se retrouve dans un sous-arbre à partir d'un nœud donné.

Définition On dit qu'une valeur dans le domaine d'une variable à un *support* si celle-ci est réalisable par au moins une sélection de valeurs dans chacun des autres domaines en respectant une contrainte donnée. Sous forme mathématique, le support est un tuple $t[1..n]$ qui satisfait la contrainte et dont chaque composantes $t[i] \in dom(X_i)$.

Des deux définitions précédentes, on conclut que si l'on peut détecter qu'une valeur, dans un domaine, n'a pas de support, alors on peut la filtrer.

Définition Une contrainte est dite *cohérente de domaine* si toutes les valeurs de toutes les variables possèdent un support.

On s'intéressera aussi au problème de satisfaction particulier qui suit.

Définition Un *problème SAT* est un problème de satisfaction de variables booléennes soumis à des *clauses SAT*. On définit ces clauses comme des conjonctions de disjonctions de littéraux (x ou $\neg x$) de la forme :

$$(x_1 \vee x_2 \vee \cdots \vee x_k) \wedge (y_1 \vee y_2 \vee \cdots \vee y_k) \wedge \cdots \wedge (z_1 \vee z_2 \vee \cdots \vee z_k).$$

1.2 Automates

Avant de définir ce qu'est un automate, il s'impose de donner une définition de langage, puisque la principale utilité des automates est de définir ceux-ci.

Définition Un *langage* est un ensemble de *mots* sur un *alphabet* fini. L'alphabet est composé d'un ensemble de symboles et les mots sont une énumération de ces symboles.

Les langages sont donc définis de manière très générale, mais ce qui nous intéresse particulièrement ce sont les langages possédant une certaine structure de sorte qu'il est possible de détecter, par un ensemble de règles, si un mot appartient au langage. Les automates et les grammaires non contextuelles permettent de définir une telle structure, ce qui nous permet d'étudier plus en profondeur leurs propriétés.

Les automates permettent de définir les grammaires les plus simples dans la hiérarchie de Chomsky, elles sont toutefois bien utiles pour représenter des séquences de valeurs avec répétitions par exemple. Nous examinerons ici les automates à nombre fini d'états.

Définition Un *automate fini* est un quadruplet $\mathcal{A} = (\mathcal{Q}, \mathcal{F}, \mathcal{I}, \mathcal{T})$ qui utilise l'alphabet Σ. On définit ses constituants comme suit :

- $\mathcal{Q}$ est un ensemble fini d'*états* ;
- $\mathcal{F} \subseteq \mathcal{Q} \times \Sigma \times \mathcal{Q}$ est l'ensemble des *transitions* sur l'alphabet Σ ;
- $\mathcal{I} \subseteq \mathcal{Q}$ est l'ensemble des *états initiaux* ;
- $\mathcal{T} \subseteq \mathcal{Q}$ est l'ensemble des *états terminaux*.

Un *mot* est accepté par l'automate si, partant d'un état initial, il existe une séquence de transition qui utilise tous les symboles du mot, en ordre, et qui se termine dans un état final.

Exemple Soit l'alphabet $\Sigma = \{0, 1, 2\}$. On définit l'automate $\mathcal{A} = (\mathcal{Q}, \mathcal{F}, \mathcal{I}, \mathcal{T})$ avec les ensembles suivants :

$\mathcal{Q} = \{q_0, q_1, q_2, q_3\}$
$\mathcal{F} = \{(q_0, 0, q_0), (q_0, 2, q_1), (q_0, 1, q_2), (q_1, 1, q_2), (q_2, 0, q_3)\}$
$\mathcal{I} = \{q_0\}$
$\mathcal{T} = \{q_3\}$

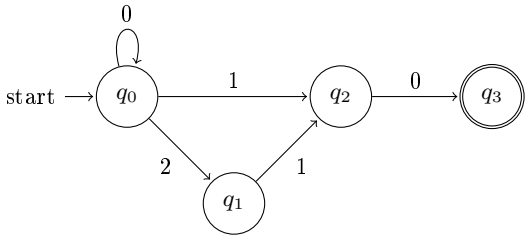


FIGURE 1 – Représentation visuelle de l'automate. Les états finaux sont doublement encerclés et l'on représente les transitions avec une flèche dont le libellé est le symbole de l'alphabet utilisé dans la transition.

Cet automate accepte les mots suivants :

– 10	– 010	– 0010	– 00010
– 210	– 0210	– 00210	– ...

On distingue alors deux types d'automates finis.

Définition Un automate fini est *déterministe* si l'ensemble des états initiaux $\mathcal{I}$ est un singleton et si $\forall q \in \mathcal{Q}$, $\forall \sigma \in \Sigma$, il existe au plus une transition $(q, \sigma, p_2) \in \mathcal{F}$. S'il ne respecte pas ces propriétés, on dit que l'automate est *non déterministe*.

Remarque Tout automate fini non déterministe peut être écrit sous la forme d'un automate fini déterministe. Toutefois, ce dernier peut avoir un nombre d'états exponentiellement plus grand.

Définition Les langages acceptés par l'ensemble des automates finis déterministes sont appelés les *langages réguliers*.

1.3 Grammaires non contextuelles

Les grammaires non contextuelles définissent le deuxième niveau dans la hiérarchie de Chomsky, c'est-à-dire qu'ils sont plus généraux que les langages réguliers. Par plus généraux, on entend que tout langage régulier peut être reconnu par une grammaire, mais que tout langage non contextuel ne peut pas nécessairement être reconnu par un automate.

Définition Une *grammaire non contextuelle* $\mathcal{G}$ est un quadruplet $(\mathcal{N}, \Sigma, S, \mathcal{P})$ défini comme suit :
- $\mathcal{N}$ est l'ensemble des *symboles non terminaux* ;
- Σ est l'ensemble des *symboles terminaux* ;
- $S \in \mathcal{N}$ est le *symbole de départ* ;
- $\mathcal{P} \subseteq \mathcal{N} \times (\mathcal{N} \cup \Sigma)^*$ est l'ensemble des *règles de dérivation* ou *règles de production*.
  - Une règle de dérivation est un couple $(X, \alpha) \in \mathcal{N} \times (\mathcal{N} \cup \Sigma)^*$. On dit alors que X *dérive* α, c'est-à-dire que la séquence α peut être engendré par le symbole non terminal X.

Un *mot* est accepté par la grammaire si, partant du symbole de départ S, il existe une séquence de règles de dérivation qui produit tous les symboles du mot, en ordre.

Remarque Nous utiliserons les conventions suivantes pour décrire une grammaire non contextuelle :
- On note par une lettre majuscule les symboles non terminaux et par une lettre minuscule les symboles terminaux.
- On note par ϵ le caractère vide.

Notation Pour parler des règles de dérivation et des dérivations possibles, nous utilisons les notations suivantes :

- $X \to xAx \mid \epsilon$ pour définir une règle de dérivation qui transforme X en xAx ou ϵ.
- $X \xrightarrow{\star} Y$ pour signifier que la chaîne $Y \in (\mathcal{N} \cup \Sigma)^*$ peut être dérivée à partir de X par une séquence de production de $\mathcal{P}$.

Finalement, pour éviter la confusion, la grammaire concernée sera ajoutée en indice à ses constituants lorsque cela n'est pas intuitivement clair. Ainsi, l'écriture $\mathcal{N}_\mathcal{G}$ signifie : les symboles non terminaux de la grammaire $\mathcal{G}$.

Exemple Soit l'alphabet $\Sigma = \{a, b\}$ auquel on y définit la grammaire suivante :

1. $S \to Y$
2. $X \to bYb \mid \epsilon$
3. $Y \to aXa$

Affirmation 1.1 *Les langages réguliers sont entièrement inclus dans la catégorie des grammaires non contextuelles.*

On peut normaliser les grammaires non contextuelles pour en faciliter l'utilisation dans des algorithmes et des preuves sans toutefois en réduire la puissance. On note cette normalisation : la forme normale de Chomsky.

Définition Une grammaire non contextuelle respectant la *forme normale de Chomsky* est décrite comme des règles de dérivation telles que le côté gauche ne possède qu'un symbole non terminal et que le côté droit possède soit deux symboles non terminaux soit un seul symbole terminal.

Exemple La grammaire $\mathcal{G}$ suivante est sous la forme normale de Chomsky, tandis que celle de l'exemple précédent ne l'est pas.

1. $S \to XY$
2. $X \to AX \mid AA$
3. $Y \to BB$
4. $A \to a$
5. $B \to b$

Voici quelques exemples de dérivation possibles [*] :
- $S \xrightarrow{1} XY \xrightarrow{2} AAY \xrightarrow{3} AABB \xrightarrow{4} aaBB \xrightarrow{5} aabb$
- $S \xrightarrow{1} XY \xrightarrow{2} AXY \xrightarrow{2} AAXY \xrightarrow{2} AAAY \xrightarrow{3} AAAABB \xrightarrow{4} aaaaBB \xrightarrow{5} aaaabb$

Ainsi, les mots *« aabb »* et *« aaaabb »*, ainsi que de nombreux autres, font partie du langage défini par $\mathcal{G}$.

Définition Les langages définis par une grammaire non contextuelle sont appelés *langage non contextuel*.

∗. Le numéro des dérivations utilisées est indiqué au-dessus du symbole de dérivation $\to$.

2 Contraintes Regular et Grammar

Les automates et les grammaires non contextuelles sont des outils très puissants pour modéliser une structure sur des séquences de symboles. Il est donc intéressant de pouvoir les utiliser comme contrainte dans un problème d'optimisation combinatoire pour imposer qu'une liste de variables possède cettedite structure.

Pour créer une contrainte d'optimisation, en plus d'en avoir son algorithme de vérification, il est préférable de posséder un algorithme qui en effectue le filtrage. Nous allons définir ces algorithmes pour les deux catégories de langages.

2.1 Regular

La contrainte Regular valide qu'une chaîne de caractères fasse partie du langage défini par un automate donné. Pour pouvoir l'utiliser, il faut lui fournir un automate et une chaîne de caractères de taille, disons, n. On représente cette dernière par n variables ayant des domaines définis par le problème d'optimisation. Cette contrainte sera donc vérifiée si et seulement si les variables possèdent une affectation valide constituant un mot accepté par l'automate et en respectant les domaines des variables.

Pour implémenter la contrainte Regular, on doit considérer les variables X_i qui représentent les caractères lus à chaque transition par l'automate et y ajouter les variables Q_i qui représentent l'état de l'automate après avoir lu i symboles. Ainsi, un triplet (Q_i, X_i, Q_{i+1}) correspond exactement à une transition de l'automate. On peut implémenter cette contrainte directement à l'aide d'une contrainte Tableau qui contient la liste de toutes les transitions. On dit qu'un triplet (Q_i, X_i, Q_{i+1}) est valide si une entrée du tableau lui correspond. La figure 2 correspond aux variables liées par une condition.

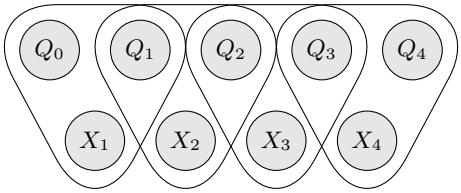


Figure 2 – Schéma des relations du filtrage d'un automate. Les variables entourées par un triangle représentent les triplets qui interviennent dans la contrainte Tableau.

La raison pour laquelle cette contrainte se traite facilement peut être déduite de la figure précédente. En effet, chaque variable appartient à moins de deux ensembles de contraintes. Ainsi, si l'on fixe une variable, on affecte la contrainte suivante et cela favorise un traitement efficace. On dit que cette structure est un *hyperarbre* car elle respecte la propriété que chaque nœud appartient à au plus deux contraintes et qu'il n'y a pas de cycle.

Avec ces constats, on obtient qu'il est nécessaire d'utiliser $n+1$ variables d'état Q_i pour n variables de symboles à détecter X_i. De plus, la contrainte TABLEAU s'exécute en temps linéaire selon la longueur du mot validé, soit en $O(n)$. Le filtrage se fait également en $O(n)$ par la même contrainte.

2.2 GRAMMAR

La contrainte GRAMMAR, quant à elle, valide qu'une chaîne de caractères appartienne à un langage non contextuel. À la manière de la contrainte REGULAR, il faut lui fournir une grammaire non contextuelle et une chaîne de caractères sous la forme de variables.

Pour le filtrage de cette contrainte, l'algorithme *Cocke-Younger-Kasami*, qui est communément abrégé en *CYK*, est le plus commun pour résoudre ce problème. L'algorithme permet de détecter si un mot appartient à une grammaire en construisant un tableau V tel que pour $X \in N_{\mathcal{G}}$, on a :

$$X \in V[i,j]$$
$$\Updownarrow$$
$$\begin{cases} X \xrightarrow{\star} A_i \dots A_{i+j-1} \mid A_k \in V[k,1] \quad \forall k \in \{i, \dots, i+j-1\} & \text{si } j > 1 \\ X \to A \mid \exists\, (A \to a) \in \mathcal{P}_{\mathcal{G}} \text{ et } a \in \Sigma_{\mathcal{G}} & \text{si } j = 1. \end{cases}$$

Cette caractérisation s'interprète par le fait que les symboles terminaux de la case $V[i,j]$ dérivent une séquence de symboles des cases $V[i,1], V[i+1,1], \dots, V[j,1]$. On donne ainsi l'algorithme suivant qui construit le tableau V en respectant cette propriété.

Algorithm 1 $CYK(\mathcal{G}, W)$

Ensure: $\mathcal{G}$ est une grammaire de symboles non terminaux $N_1, \dots, N_g \in \mathcal{N}_{\mathcal{G}}$ et de symbole de départ S.
Ensure: W est une chaîne de caractères de lettres $w_1, \dots, w_n$.

```
1:  P[n, n, g] ← un tableau de booléens initialisés à false
2:  for i = 1 to n do
3:      for (N_j → w_i) ∈ N_G do
4:          P[i, 1, j] ← true
5:  for i = 2 to n do
6:      for j = 1 to n − i + 1 do
7:          for k = 1 to i − 1 do
8:              for (N_a → N_b N_c) ∈ N_G do
9:                  if P[j, k, b] ∧ P[j + k, i − k, c] then
10:                     P[j, i, a] ← true
11: if P[1, n, S] then
12:     return « G accepte W »
13: else
14:     return « G refuse W »
```

Proposition 2.1 *L'algorithme CYK vérifie qu'un mot fait partie d'un langage défini par une grammaire non contextuelle en* $O(n^3 \cdot |\mathcal{G}|)$

Démonstration Pour effectuer l'analyse, comptons le nombre de fois que les opérations de la ligne 4 et de la ligne 9 sont exécutées. La ligne 4 est exécutée dans deux boucles imbriquées dont la première est exécutée n fois et la seconde $|\mathcal{G}|$ fois.

La ligne 9, quant à elle, est imbriquée dans 4 boucles. Les trois premières s'exécutent un nombre maximal de n fois. La quatrième itère sur les règles de dérivation de la grammaire en entier. Cette ligne sera donc exécutée de l'ordre de $O(n^3 \cdot |\mathcal{G}|)$ fois. Par la règle du maximum, on constate que le nombre d'exécutions de la ligne 9 majore clairement celle de la ligne 4 et on obtient le résultat. ∎

Comme l'on montré Quimper et Walsh (2006), on peut modifier cet algorithme pour effectuer le filtrage des contraintes à la manière de CYK.

Algorithm 2 *CYK-prop*$(\mathcal{G}, [X_1, \ldots, X_m])$

1: **for** $i = 1$ **to** m **do**
2: $\quad V[i,1] \leftarrow \{A \mid (A \leftarrow a) \in \mathcal{P}_\mathcal{G}, a \in dom(X_i)\}$
3: **for** $j = 2$ **to** m **do**
4: $\quad$ **for** $i = 1$ **to** $m - j + 1$ **do**
5: $\quad\quad V[i,1] \leftarrow \emptyset$
6: $\quad\quad$ **for** $k = 1$ **to** $j - 1$ **do**
7: $\quad\quad\quad V[i,j] \leftarrow V[i,j] \cup \{A \mid (A \leftarrow BC) \in \mathcal{P}_\mathcal{G}, B \in V[i,k], C \in V[i+k, j-k]\}$
8: **if** $S \notin V[1,m]$ **then**
9: $\quad$ **return** « *Non satisfiable* »
10: marquer$(1, m, S)$
11: **for** $j = m$ **downto** 2 **do**
12: $\quad$ **for** $i = 1$ **to** $m - j + 1$ **do**
13: $\quad\quad$ **for all** $(A \to BC) \in \mathcal{P}_\mathcal{G} \mid (i, j, A)$ est marqué **do**
14: $\quad\quad\quad$ **for** $k = 1$ **to** $j - 1$ **do**
15: $\quad\quad\quad\quad$ **if** $B \in V[i,k] \wedge C \in V[i+k, j-k]$ **then**
16: $\quad\quad\quad\quad\quad$ marquer(i, k, B)
17: $\quad\quad\quad\quad\quad$ marquer$(i + k, j - k, C)$
18: **for** $i = 1$ **to** m **do**
19: $\quad dom(X_i) \leftarrow \{a \in dom(X_i) \mid (A \leftarrow a) \in \mathcal{P}_\mathcal{G} \wedge (i, 1, A)$ est marqué$\}$
20: **return** « *Satisfiable* »

On ne prouvera pas la validité de l'algorithme, mais on traitera un exemple pour détailler le comportement de l'algorithme après avoir obtenu le tableau fourni par CYK qui est effectué en première partie de l'algorithme.

Exemple Soit la grammaire suivante :

- $S \to XY$
- $X \to AX \mid AA$
- $Y \to AB \mid BB$
- $A \to a$
- $B \to b$

On souhaite construire un mot de 5 caractères appartenant au langage de cette grammaire. Les lignes 1 à 7 de l'algorithme nous donnent le tableau V construit par CYK.

	1	2	3	4	5
1	{A, B}	{X, Y}	{X}	{X, S}	{X, S}
2	{A, B}	{X, Y}	{X}	{X, S}	
3	{A, B}	{X, Y}	{X}		
4	{A, B}	{X, Y}			
5	{A, B}				

TABLE 1 – Construction du tableau V par l'algorithme *CYK-prop*

On peut ensuite effectuer la propagation des contraintes dans ce même tableau en marquant les valeurs sélectionnées qui permettent d'obtenir S.

	1	2	3	4	5
1	{(A), B}	{X, Y}	{(X)}	{X, S}	{X, (S)}
2	{(A), B}	{(X), Y}	{X}	{X, S}	
3	{(A), B}	{X, Y}	{X}		
4	{(A), (B)}	{X, (Y)}			
5	{A, (B)}				

TABLE 2 – Marquage des éléments du tableau V

Ainsi, les seules valeurs possibles dans les domaines des variables X_i deviennent :

- $dom(X_1) = \{A\}$
- $dom(X_2) = \{A\}$
- $dom(X_3) = \{A\}$
- $dom(X_4) = \{A, B\}$
- $dom(X_5) = \{B\}$

On voit donc que cela permet de réduire les domaines des variables et donc l'espace de recherche au prix d'effectuer du filtrage en $O(n^3 \cdot |\mathcal{G}|)$.

2.3 Décomposition de la contrainte GRAMMAR

Nous pouvons aussi décomposer la contrainte que nous venons d'observer pour la représenter sous la forme d'opération binaire ET et OU. Ce genre de représentation permet de donner une meilleure analyse et permet surtout d'être implémenté dans des solveurs SAT. Nous allons effectuer la décomposition de cette deuxième contrainte. Celle-ci sera formulée sous la forme d'un *DAG*, c'est-à-dire un graphe orienté acyclique (de l'anglais *directed acyclic graph*), mais d'abord présentons les variables binaires que nous devrons utiliser.

Affirmation 2.2 *On peut modéliser une grammaire avec des contraintes SAT en introduisant les variables et les règles suivantes.*

Variables :

- $x(a, i, 1) = true \iff a \in dom(X_i)$.
- $x(A, i, j) = true \iff \exists A \to BC \in G$ *tel que* B *dérive les symboles* i *à* $i + k - 1$ *et* C *dérive les symboles* $i + k$ *à* $i + j - 1$.

Règles : *En commençant par le symbole de départ S, pour chaque choix admissible dans le tableau V tel que calculé par l'algorithme 2, on ajoute la règle*

$$x(A,i,j) \equiv \bigvee_{\substack{k\in[1,j-1] \\ V[i,k] \text{ est marqué} \\ V[i+k,j-k] \text{ est marqué}}} x(B,i,k) \wedge x(C,i+k,j-k).$$

Pour chaque règle terminale de type $A \to a_1 \,|\, a_2 \,|\, \ldots \,|\, a_r$, on ajoute la règle

$$x(A,i,1) \equiv x(a_1,i,1) \vee x(a_2,i,1) \vee \cdots \vee x(a_r,i,1).$$

Exemple Reprenons la grammaire de l'exemple précédent, on obtient évidemment le même tableau V avec les mêmes marquages. En utilisant l'affirmation précédente, on obtient les règles qui suivent.

	1	2	3	4	5
1	{(A), B}	{X, Y}	{(X)}	{X, S}	{X, (S)}
2	{(A), B}	{(X), Y}	{X}	{X, S}	
3	{(A), B}	{X, Y}	{X}		
4	{(A), (B)}	{X, (Y)}			
5	{A, (B)}				

Table 2 – Marquage des éléments du tableau V

- $x(S,1,5) \equiv x(X,1,3) \wedge x(Y,4,2)$
- $x(X,1,3) \equiv x(A,1,1) \wedge x(X,2,2)$
- $x(X,2,2) \equiv x(A,2,1) \wedge x(A,3,1)$
- $x(Y,4,2) \equiv (x(A,4,1) \wedge x(B,5,1)) \vee (x(B,4,1) \wedge x(B,5,1))$
- $x(A,1,1) \equiv x(a,1,1)$
- $x(A,2,1) \equiv x(a,2,1)$
- $x(A,3,1) \equiv x(a,3,1)$
- $x(A,4,1) \equiv x(a,4,1)$
- $x(B,4,1) \equiv x(b,4,1)$
- $x(B,5,1) \equiv x(b,5,1)$

Partant de ces règles, on peut concevoir le graphe orienté acyclique suivant qui représente cette contrainte.

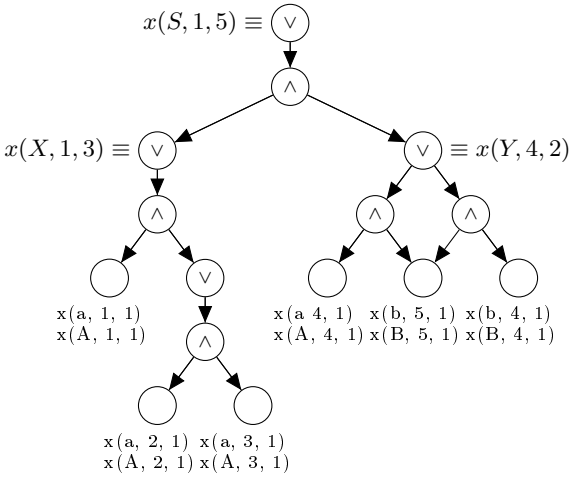


FIGURE 3 – DAG des règles retournées par l'algorithme *CYK-prop*

Avec le DAG produit, on peut résoudre le problème de reconnaissance de la grammaire en résolvant plutôt le problème de satisfaction des clauses SAT inscrite dans le graphe. Or, même s'il s'agit d'un problème $\mathcal{NP}$-complet, le problème est plus simple à conceptualiser et les solveurs SAT sont très performants. Toutefois, la conception de ce graphe s'effectue au moins en un temps de l'ordre de $O(n^3 \cdot |\mathcal{G}|)$ puisqu'il faut exécuter l'algorithme *CYK-prop*. De plus, la forme particulière des clauses SAT permet à tout solveur SAT appliquant la cohérence de domaine de trouver une solution en temps $O(n^3 \cdot |\mathcal{G}|)$.

3 Modélisation sous la forme d'un MIP

Maintenant que nous avons énoncé une manière de modéliser les contraintes REGULAR et GRAMMAR en programmation par contraintes, nous pouvons aussi modéliser le problème dans un contexte de programmation linéaire sous la forme de MIP.

Définition Un problème d'*optimisation linéaire mixte* (de l'anglais *mixed integer programming*, d'où l'acronyme *MIP*), est un problème d'optimisation linéaire dont certaines variables sont entières et d'autres peuvent être réelles.

La manière standard d'écrire un problème de ce genre est la suivante.

$$\begin{aligned} \min c^t \begin{bmatrix} x & y \end{bmatrix} & \\ \text{t.q.} \quad A \begin{bmatrix} x & y \end{bmatrix} & \leq b, \\ x & \geq 0, \\ y & \in \mathbb{Z} \end{aligned}$$

Il est aussi nécessaire d'utiliser des graphes de flot, que nous définissons tout de suite.

Définition Un *graphe de flot* est un graphe orienté dans lequel chaque arête $(u, v) \in E$ a une capacité non négative $c(u, v)$. On impose que s'il existe une arête (u, v), alors il n'existe pas d'arête (v, u). On appelle *flot* la circulation d'une certaine quantité dans chaque arête du graphe. On nomme deux sommets particuliers :

- La *source* est le nœud d'où provient un flot initial qui traverse le graphe.
- Le *puits* est le nœud d'où le flot qui circule dans le graphe doit s'échapper.

3.1 Contrainte REGULAR

Avant de concevoir une interprétation d'un langage régulier sous la forme d'un MIP, observons une propriété d'un langage régulier.

Proposition 3.1 *Soit L_1 et L_2, deux langages réguliers, alors $L_1 \cap L_2$ est aussi un langage régulier.*

Démonstration Montrons d'abord deux résultats intermédiaires.

Lemma 3.2 *L'union de deux langages réguliers est un langage régulier.*

Démonstration Soit $\mathcal{A}_1$ et $\mathcal{A}_2$, les deux automates issues des langages réguliers. On peut construire $\mathcal{A}_\cup$ en considérant l'état initial des deux automates comme étant le même et en effectuant l'union de toutes les autres composantes de l'automate. Si une transition est répétée de sorte que l'automate n'est plus déterministe, on fusionne plutôt les deux états correspondants. L'automate résultant représente l'union des deux langages, puisqu'il accepte clairement les chaînes de $\mathcal{A}_1$ et de $\mathcal{A}_2$, alors $\mathcal{A}_\cup$ est régulier. ∎

Lemma 3.3 *Le complément d'un langage régulier est régulier.*

Démonstration Puisque le langage initial est régulier, il existe une représentation sous la forme d'un automate, disons $\mathcal{A}$. Or, remarquons que le langage complémentaire de $\mathcal{A}$ possède les mêmes états et que seuls les états terminaux sont inversés tel que $\mathcal{T}_{\overline{\mathcal{A}}} = \mathcal{Q}_{\mathcal{A}} \setminus \mathcal{T}_{\mathcal{A}}$. Clairement, le langage engendré par cet automate est le langage complémentaire et celui-ci est régulier. ∎

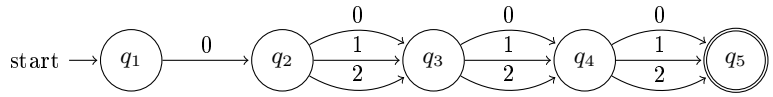
Maintenant, pour la preuve principale, puisque les langages sont essentiellement une série d'ensembles, toutes les propriétés des ensembles sont héritées. Par les lois de De Morgan, on obtient que $A_1 \cap A_2 = \overline{\overline{A_1} \cup \overline{A_2}}$ et le résultat suit immédiatement des lemmes précédents. ∎

Concrètement, il est facile de construire l'intersection de deux langages donnés, il suffit de construire un nouvel automate fixé sur une grille de $n \times m$ états où n est le nombre d'états du premier langage et m celui du second. Partant de l'état initial des deux langages, on trace l'ensemble des chemins possibles qui respectent les deux automates en même temps en appliquant la règle suivante.

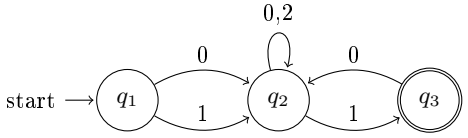
Lorsqu'un symbole σ est admissible conjointement par les automates $\mathcal{A}_1$ et $\mathcal{A}_2$ tel que $(q_{i_1}, \sigma, q_{i_2}) \in \mathcal{F}_{\mathcal{A}_1}$ et $(p_{j_1}, \sigma, p_{j_2}) \in \mathcal{F}_{\mathcal{A}_2}$, alors on représente ce choix par la transition $(r_{i_1,j_1}, \sigma, r_{i_2,j_2}) \in \mathcal{F}_{\mathcal{A}_\cap}$.

On peut par la suite enlever toutes les transitions qui ne sont pas accessibles par une séquence de transitions partant de l'état initial.

Exemple L'intersection des deux premiers automates peut s'exprimer comme le troisième dans cet exemple.



(a) Automate reconnaissant les mots de 4 symboles commençant par le symbole 0.



(b) Automate reconnaissant un langage quelconque.

FIGURE 4 – Automates à intersecter.

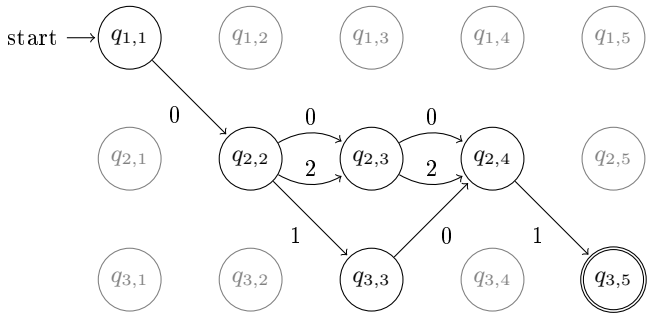


FIGURE 5 – Automate résultant de l'intersection des deux premiers.

Comme dans notre problème de reconnaissance de langage, la taille du mot est fixée, nous allons utiliser l'automate de la contrainte REGULAR et un deuxième langage qui impose une taille fixe de n caractères. Par la propriété précédente, l'intersection des deux langages sera un langage régulier.

On peut considérer le graphe issu de la création de l'intersection des deux langages. À ce graphe, on ajoute un nœud f qui est lié par tous les états acceptants de l'automate. En interprétant ce graphe comme un graphe de flot, on peut obtenir une formulation d'équations linéaires pour former un MIP. Pour ce faire, introduisons quelques variables pour poser le problème de flot correctement :

- N^i est l'ensemble des états de la colonne i ;
- $f_{(i_1,j_1)\xrightarrow{\sigma}(i_2,j_2)}$ est la variable de flot qui, partant du nœud à la position (i_1, j_1), lit le caractère σ et termine sa transition au nœud (i_2, j_2) ;
- $f_{(n,j_1)\to(n+1,j_2)}$ est une variable de flot qui, partant du nœud terminal (n, j_1) se dirige au nœud $(n+1, j_2)$ sans lire aucun caractère ;
- Δ^+_{ij} et Δ^-_{ij} représentent respectivement l'ensemble des arêtes sortantes et des arêtes entrantes du nœud à la position (i, j) ;
- s représente la source et t, le puits.

Propriétés

- Comme l'automate $\mathcal{A}_2$ impose la taille de la chaîne, il parcourt ses états linéairement, les transitions seront toujours d'un nœud appartenant à N^i vers un nœud de N^{i+1}.
- Toutes les variables de flot $f_{(i_1,j_1)\xrightarrow{\sigma}(i_2,j_2)}$ respectent la propriété $i_1+1 = i_2$.

La proposition suivante donne la formulation du MIP à partir de l'automate $\mathcal{A}_\cap := \mathcal{A}_1 \cap \mathcal{A}_2$ et la remarque qui suit donne une interprétation des formules du MIP.

Proposition 3.4 *Les formules suivantes permettent d'encoder la contrainte* REGULAR *dans un MIP.*

$$\sum_{\substack{(\sigma,k)\\ \langle s\xrightarrow{\sigma}(2,k)\rangle\in\Delta^+_{is}}} f_{s\xrightarrow{\sigma}(2,k)} = w \tag{3.1}$$

$$\sum_{\substack{(j,\sigma)\\ \langle (i-1,j)\xrightarrow{\sigma}(i,k)\rangle \in \Delta^-_{ik}}} f_{(i-1,j)\xrightarrow{\sigma}(i,k)} = \sum_{\substack{(j,\sigma)\\ \langle (i,j)\xrightarrow{\sigma}(i+1,k)\rangle \in \Delta^+_{ik}}} f_{(i,j)\xrightarrow{\sigma}(i+1,k)} \quad \forall\, i \in \{2,\dots,n\}, k \in N^i \tag{3.2}$$

$$\sum_{\substack{(\sigma,k)\\ \langle (n,j)\xrightarrow{\sigma}(n+1,k)\rangle \in \Delta^-_{(n+1)k}}} f_{(n,j)\xrightarrow{\sigma}(n+1,k)} = f_{(n+1,j)\to t} \quad \forall\, k \in N^{n+1} \tag{3.3}$$

$$\sum_{k \in N^{n+1}} f_{(n+1,j)\to t} = w \tag{3.4}$$

On impose aussi que le réseau soit modélisé par l'automate $\mathcal{A} = (\mathcal{Q}, \mathcal{F}, \mathcal{I}, \mathcal{T})$:

$$f_{(i,j)\xrightarrow{\sigma}(i+1,k)} \in \{0,1\} \quad \forall\, i \in \{1,\dots,n\}, \langle j,\sigma,k\rangle \in \mathcal{F} \mid j \in N^i \tag{3.5}$$

$$f_{(n+1,j)\to t} \in \{0,1\} \quad \forall j \in N^{n+1} \tag{3.6}$$

Une solution à ce problème de flot correspond à un mot reconnu par $\mathcal{A}$.

Remarque On peut interpréter les équations (3.1)-(3.4) comme ceci :

(3.1) La somme des flots en sortie de la source est de w.

(3.2) À chaque variable, la somme des flots en entrée est égale à la somme des flots en sortie.

(3.3) Le puits récolte tous les flots des nœuds précédents.

(3.4) La somme des flots en entrée au puits est de w.

Remarque On peut vouloir utiliser les variables des symboles du mot dans d'autres contraintes. Pour ce faire, on encode les variables $x_{i,\sigma}$ comme suit :

$$x_{i,\sigma} = \begin{cases} 1 & \text{si le symbole à la position i la valeur } \sigma, \\ 0 & \text{sinon.} \end{cases}$$

Pour lier ces variables avec les variables de flots dans le MIP, on doit ajouter les contraintes suivantes :

$$x_{i\sigma} = \sum_{\substack{(j,k)\\ \langle j,\sigma,k\rangle \in \mathcal{F}\\ j \in N^i}} f_{(i,j)\xrightarrow{\sigma}(i+1,k)} \quad \forall\, i \in \{1,\dots,n\}, \sigma \in dom(X_i) \tag{3.7}$$

Démonstration En fixant $w = 1$ et par le fait que chaque variable est binaire, on sait qu'une solution au problème de flot forme un chemin $\mathcal{C}_\mathcal{G}$ dans le graphe. Puisque le chemin dans le graphe est équivalent à un chemin dans l'automate $\mathcal{A}_\cap$, on obtient le chemin $\mathcal{C}_{\mathcal{A}_\cap}$ qui lit chaque symbole le long du chemin $\mathcal{C}_\mathcal{G}$. Comme le chemin $\mathcal{C}_\mathcal{G}$ se termine au puits, le chemin $\mathcal{C}_{\mathcal{A}_\cap}$ se termine sur l'état précédent qui est un état final de l'automate. Par conséquent, la chaîne de caractères reconnue par le chemin est acceptée par $\mathcal{A}_\cap$.

Inversement, une chaîne acceptée dans l'automate forme un chemin $\mathcal{C}$ dans le graphe. Puisque notre automate a été conçu à partir de $\mathcal{A}_2$, il ne peut y avoir de boucle dans l'automate et on ne peut avancer que d'une colonne à la fois. Par conséquent, on peut assigner la valeur 1 aux variables de flot correspondant aux transitions empruntées par la chaîne acceptante de $\mathcal{A}_\cap$ et la valeur de 0 aux autres. Ces assignations de valeurs forment une solution au problème de flot telle que la source, c'est-à-dire l'état initial de l'automate, forme un chemin jusqu'au puits, soit l'état final, en respectant les équations (3.1)-(3.4) de la proposition. Celles-ci sont trivialement vérifiées par l'existence du chemin $\mathcal{C}$. ∎

Remarque Le nombre de variables créées dans le MIP par ces équations est de l'ordre de $O(n \cdot |\mathcal{T}_{\mathcal{A}_1}|)$ où n est la longueur du mot. Le nombre de contraintes est de l'ordre de $O(n \cdot |\mathcal{Q}_{\mathcal{A}_1}|)$.

3.2 Contrainte Grammar

Pour énoncer les équations linéaires décrivant une grammaire sous la forme d'un MIP, nous utilisons la décomposition sous forme de DAG par des nœuds *and* et *or*.

Proposition 3.5 *Soient N_{or} et N_{and}, l'ensemble des nœuds du graphe issu de la décomposition de la contrainte* Grammar *de type* or *et de type* and *respectivement et soit $X_{or,u}$ la variable correspondante à la valeur du nœud $u \in N_{or}$ et $X_{and,v}$ la variable correspondante à la valeur du nœud $v \in N_{and}$. Notons $enfant(u)$ les identifiants des enfants du nœud u et $parent(u)$ ceux de ses parents.*

Alors, la formulation suivante permet d'encoder la contrainte Grammar *dans un MIP.*

$$\sum_{v \in enfant(u)} X_{and,v} = X_{or,u} \quad \forall\, u \in N_{or} \tag{3.8}$$

$$\sum_{v \in parent(u)} X_{and,v} = X_{or,u} \quad \forall\, u \in N_{or} \tag{3.9}$$

$$X_{and}(S_0, 1, n) = 1 \tag{3.10}$$

$$X_{or,u}, X_{and,v} \in \{0,1\} \quad \forall\, u \in N_{or}, v \in N_{and} \tag{3.11}$$

Remarque Comme la contrainte (3.11) impose que les variables $X_{or,u}$ et $X_{and,v}$ soient binaires, alors les contraintes peuvent s'exprimer comme suit :

- La contrainte (3.8) impose qu'une variable associée à un nœud *or* vaille 1 si exactement une des variables associées à ses enfants est égale à 1 ;
- La contrainte (3.9) impose qu'une variable associée à un nœud *or* vaille 1 si exactement une des variables associées à ses parents est égale à 1 ;
- La contrainte (3.10) impose que la racine soit égale à 1 pour que le mot soit reconnu (le cas contraire n'est habituellement pas intéressant).

On note que les contraintes sur les nœuds *and* peuvent être déduites des contraintes (3.8) et (3.9). En effet, si un nœud $v \in N_{and}$ est vrai ($X_{and,v} = 1$), alors on veut que ses enfants soient aussi vrais. Or, pour chaque enfant $u \in enfant(v)$, on a

$$X_{or,u} = \sum_{w \in \text{parent}(u)} X_{and,w} \geq X_{and,v}$$

et tous les enfants du nœud u valent 1, respectant la contrainte sur les nœuds *and*.

Lemma 3.6 *Les contraintes* (3.8), (3.9) *et* (3.10) *sont satisfaites si et seulement si le graphe* and/or *de décomposition de la contrainte* GRAMMAR *est vérifié.*

Démonstration Avec la construction du DAG, on sait que les nœuds sont en alternance *and* et *or*. De plus, les nœuds *and* ont toujours un seul parent.

$\Longrightarrow$) On suppose que les équations sont respectées. Chaque variable assignée à 1 dans le MIP possède une variable booléenne associée dans l'arbre que l'on met à *vrai*. Par les contraintes (3.8) et (3.9), chaque nœud *or* possède un enfant à l'état *vrai* et un parent à l'état *vrai*. Le parent est un nœud *and* par l'observation plus haut et celui-ci a soit un ou deux enfants mis à *vrai*. Ainsi, tous les enfants d'un nœud *and* mis à *vrai* sont eux aussi *vrais*. De plus, la racine est *vraie* par la contrainte (3.10). Ainsi, cela montre que toutes les variables de valeur 1 dans le MIP ont pour valeur *vraie* dans le graphe. Maintenant, montrons que les autres variables ont aussi une valeur booléenne. En effet, on peut placer toutes les autres à *faux*. En partant des feuilles de l'arbre et en remontant selon les règles mentionnées, on remarque que les variables booléennes forment une affectation valide selon les règles du graphe *and/or* et la propriété est démontrée.

$\Longleftarrow$) Supposons maintenant que la contrainte GRAMMAR est vérifiée, c'est-à-dire que le graphe *and/or* a un affectation valide et montrons que le MIP engendré par les contraintes (3.8)-(3.10) est réalisable. Puisque la contrainte est vérifiée, le graphe *and/or* possède un arbre de dérivation partant de la racine qui est assignée à *vrai*. En sélectionnant arbitrairement un tel arbre de dérivation et en choisissant arbitrairement tout descendant d'un nœud *or* parmi tous ceux étant *vrai*, on affecte toutes les variables correspondantes sur le chemin arbitraire formée comme valant un et les autres zéros. La contrainte (3.8) est vérifiée puisque tous les enfants d'un nœud *or* de valeur un à exactement un enfant de valeur un par le choix arbitraire de l'enfant dans la construction. Chaque nœud de l'arbre de dérivation n'a qu'un seul parent, puisque c'est un arbre, ce qui vérifie la contrainte (3.9). La contrainte (3.10) est trivialement vérifiée puisque la racine était mise à *vrai*, ainsi la variable correspondante à obligatoirement pour valeur un.

Ainsi, la satisfaction des assignations du graphe *and/or* est valide si et seulement si les équations du MIP sont vérifiées. ∎

Remarque Le nombre de variables créées dans le MIP par ces équations est de l'ordre de $O(n^3 \cdot |\mathcal{P}_\mathcal{G}|)$ où n est la longueur du mot. Le nombre de contraintes est de l'ordre de $O(n^2 \cdot |\mathcal{P}_\mathcal{G}|)$.

Remarque Sans démonstration, il est intéressant de noter que le modèle MIP d'une grammaire non contextuelle encodant un langage régulier donne un MIP de même taille, en nombre de variables et en nombre de contraintes, que le modèle MIP d'un langage régulier décrit la section précédente donnée par le même automate.

4 Réduction à la multiplication matricielle

Maintenant que nous avons vu différentes techniques de reconnaissance de grammaire, on peut s'intéresser à essayer de les optimiser. Valiant réduit le problème de reconnaissance à un problème de multiplications de matrices, ce qui permet de résoudre le problème de reconnaissance avec une complexité sous-cubique. Dans cette section, nous ferrons la réduction qui permet d'en arriver à cette conclusion.

La réduction s'effectue en quatre étapes, on montre d'abord une réduction du problème de reconnaissance $P_R(n)$ vers un problème de fermeture transitive $P_{FT}(n)$, qui, à son tour, se réduit en un problème de multiplication de matrice $P_M(n)$ et finalement se réduit à un problème de multiplication de matrices booléennes $P_{MB}(n)$.

4.1 Préalables

Pour accomplir cette réduction, nous allons utiliser la forme normale de Chomsky d'une grammaire non contextuelle $\mathcal{G} = (\mathcal{N}, \Sigma, A_1, P)$. Nous définissons une opération binaire multiplicative sur les sous-ensembles $N_1, N_2 \subseteq \mathcal{N}_\mathcal{G}$:

$$N_1 \star N_2 := \{A_i \mid \exists\, A_j \in N_1, A_k \in N_2 \text{ tel que } (A_i \to A_j A_k) \in \mathcal{P}_\mathcal{G}\}. \tag{4.1}$$

Auquel on peut y définir une multiplication de matrices ayant pour valeur des sous-ensembles de $\mathcal{N}_\mathcal{G}$.

Définition Soit $a, b \in M_n(\mathcal{P}(N))$, alors on définit la multiplication $a \star b = c$ [†] comme suit :

$$c_{i,k} = \bigcup_{j=1}^{n} a_{i,j} \star b_{j,k} \tag{4.2}$$

Introduisons maintenant la notion de clôture qui est cruciale à la réduction.

Définition Un ensemble E est dit *clos* sur une relation binaire si cette opération, appliquée à des éléments de E, produit un élément de E. On nomme la *fermeture* d'un ensemble $S \subseteq E$, le plus petit sous-ensemble de E qui contient S qui est clos sur la relation binaire.

Exemple

- Les naturels sont clos sur l'addition, mais ne le sont pas sur la soustraction (car $1 - 2 = -1 \notin \mathbb{N}$).
- La fermeture de $E := \{1, 2\}$ sur la multiplication dans les entiers $\mathbb{Z}$ est l'ensemble E lui-même.

Définition La *fermeture transitive* d'une relation R sur un ensemble E est une relation R^+ sur E tel que $R \subseteq R^+$ et que R^+ est minimale.

†. On note les matrices avec des lettres minuscules pour éviter la confusion avec les symboles non terminaux de la grammaire.

Exemple Soit E un ensemble de villes et R une relation binaire telle que x est en relation avec y si et seulement s'il existe un vol direct qui permet de relier les villes x et y par avion. Alors la fermeture transitive de R forme R^+ qui met en relation le couple (x, y) si et seulement s'il existe une séquence de vol permettant de voler de la ville x à la ville y.

Affirmation 4.1 *La fermeture transitive d'une matrice carrée a sur notre relation multiplicative peut être définie comme :*

$$a^+ = a^{(1)} \cup a^{(2)} \cup \dots \tag{4.3}$$

avec

$$a^{(i)} = \begin{cases} \bigcup_{j=1}^{i-1} a^{(j)} \star a^{(i-j)} & si\ i > 1 \\ a & si\ i = 1. \end{cases}$$

Notons la complexité algorithmique des différentes problèmes comme suit :
- $R(n)$ la complexité de l'algorithme de reconnaissance d'une chaîne de caractères de longueur n par une grammaire non contextuelle ;
- $FT(n)$ la complexité de l'algorithme de fermeture transitive de matrice triangulaire supérieure $n \times n$;
- $M(n)$ la complexité de l'algorithme de multiplication de matrices triangulaire supérieure $n \times n$;
- $MB(n)$ la complexité de l'algorithme de multiplication de matrices booléennes triangulaires supérieures par bloc de $|\mathcal{N}_\mathcal{G}| \times 1$ ou $1 \times |\mathcal{N}_\mathcal{G}|$ dont la taille est de $|\mathcal{N}_\mathcal{G}|n \times n$ ou $n \times |\mathcal{N}_\mathcal{G}|n$.

4.2 Réduction $P_R(n) \preceq P_{FT}(n)$

Montrons que la reconnaissance d'une chaîne de caractères $x_1 x_2 \dots x_n \in \Sigma^*$ par la grammaire $\mathcal{G}$ se réduit à un calcul de fermeture transitive d'une relation binaire.

Soit la matrice b définie comme suit :

$$b_{i,j} = \begin{cases} A_k \mid (A_k \to x_i) \in \mathcal{P}_\mathcal{G} & \text{si } j = i + 1 \\ \emptyset & \text{sinon.} \end{cases}$$

Proposition 4.2 *Les éléments de la fermeture transitive b^+ sont uniquement ceux ayant la propriété*

$$A_k \in b^+_{i,j} \iff A_k \xrightarrow{\star} x_i \dots x_{j-1}.$$

Démonstration Effectuons la preuve par induction généralisée sur les unions qui définissent la fermeture transitive b^+.

Cas de base : Soit $b^+_{i,j} \in b^{(1)} = b$, le premier terme de l'union. Il est clair par la définition de b que

$$A_k \in b_{i,i+1} \iff A_k \xrightarrow{\star} x_i,$$

ce qui valide le cas de base.

Étape d'induction : On prend comme hypothèse que

$$A_k \in (b^{(1)} \cup b^{(2)} \cup \cdots \cup b^{(l)})_{i,j} \iff A_k \overset{\star}{\to} x_i \dots x_{j-1}$$

et il faut montrer la relation

$$A_k \in (b^{(1)} \cup b^{(2)} \cup \cdots \cup b^{(l+1)})_{i,j} \iff A_k \overset{\star}{\to} x_i \dots x_{j-1}.$$

C'est-à-dire qu'il faut montrer le cas $A_k \in b_{i,j}^{(l+1)}$.

$\Longrightarrow$: On suppose que $A_k \in b_{i,j}^{(l+1)}$ et de la définition, on a :

$$b_{i,j}^{(l+1)} = \bigcup_{m=1}^{l} b^{(m)} \star b^{(l+1-m)}.$$

Comme $m, l+1-m < l+1$, tous les éléments de l'union satisfont la propriété par l'hypothèse d'induction. Il faut montrer que la propriété se conserve en appliquant la relation multiplicative. Soit $N_1 \in b_{i,j}^{(m)}$ et $N_2 \in b_{j,k}^{(l+1-m)}$ pour certains indices i, j, k impliqués dans la multiplication matricielle. Deux cas peuvent survenir dans la relation :

1. Si $N_1 = \emptyset$ ou $N_2 = \emptyset$. Alors le produit $N_1 \star N_2 = \emptyset$, ce qui découle directement de la définition de $\star$.
2. Si N_1 et N_2 ne sont pas l'ensemble vide, alors, on prend chaque élément de ces deux ensembles deux à deux, disons A_{q_1} et A_{q_2}, et l'on vérifie s'ils sont engendrés par une certaine production $A_q \to A_{q_1} A_{q_2}$. Pour chacun de ces cas vérifiés, on ajoute A_q dans la cellule de la matrice à la position $b_{i,k}^{(l+1)}$ et on doit vérifier que la propriété est respectée. Or, celle-ci est valide pour $A_{q_1} \in b_{i,j}^{m}$ et $A_{q_2} \in b_{j,k}^{l+1-m}$ et le calcul suivant nous donne la conclusion.

$$\begin{aligned} A_q &\to A_{q_1} A_{q_2} \\ &\overset{\star}{\to} (x_i \dots x_{j-1})(x_j \dots x_{k-1}) \\ &\overset{\star}{\to} x_1 \dots x_{k-1} \end{aligned}$$

$\Longleftarrow$: Maintenant si $\exists A_q \in \mathcal{N}_\mathcal{G}$ tel que $A_q \overset{\star}{\to} x_i \dots x_{k-1}$, on veut montrer que $A_q \in b_{i,k}^{(l+1)}$ pour certains indices i et k. Comme $k-1 > i$ et que la grammaire $\mathcal{G}$ est sous la forme normale de Chomsky, il existe A_{q_1} et A_{q_2} tel que $(A_q \to A_{q_1} A_{q_2}) \in \mathcal{P}_\mathcal{G}$ et donc que $A_{q_1} A_{q_2} \overset{\star}{\to} x_i \dots x_{k-1}$. En particulier, il existe j tel que $A_{q_1} \overset{\star}{\to} x_i \dots x_{j-1}$ et $A_{q_2} \overset{\star}{\to} x_j \dots x_{k-1}$. Or, ces deux termes font partie de l'union $b^{(1)} \cup b^{(2)} \cup \cdots \cup b^{(l)}$ et possèdent la propriété d'induction d'où $A_{q_1} \in b_{i,j}^{(m)}$ et $A_{q_2} \in b_{j,k}^{(l+1-m)}$. Par notre définition de $\star$, on a donc que $A_q \in b_{i,k}^{(l+1)}$.

Théorème 4.3 $O\big(R(n)\big) \subseteq O\big(FT(n+1)\big)$.

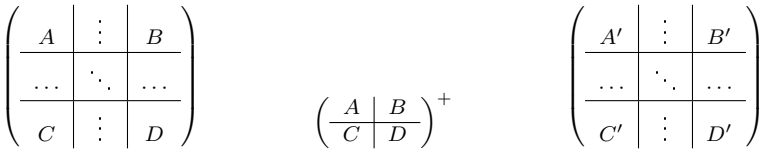
Démonstration Soit S, le symbole de départ de la grammaire. Par la proposition précédente, on trouve que pour que $S \stackrel{\star}{\rightarrow} x_1 \dots x_n$, il est nécessaire que $S \in b_{1,n+1}$. Ainsi, plutôt que de calculer les dérivations de S, on peut plutôt construire la matrice b, effectuer la fermeture transitive de b et obtenir l'information. Construire la matrice b nécessite $\frac{(n+1)^2}{2} \in O(n^2)$ opérations, ce qui est inférieur au minorant de l'opération de multiplication de matrices. ∎

4.3 Réduction $P_{FT}(n) \preceq P_M(n)$

La stratégie de Valiant (1974) est de décrire une procédure récursive pour calculer la fermeture transitive qui possède la même complexité qu'un algorithme de multiplication de matrices. D'abord, nous devons introduire une opération.

Définition Soit b une matrice triangulaire supérieure $n \times n$. On définit $b^{+(r:s)}$ comme la matrice résultante des opérations suivantes :

1. Supprimer toutes les cellules $b_{i,j}$ de b tel que $r \leq i \leq s$ où $r \leq j \leq s$.
2. Calculer la fermeture transitive de la matrice $(n + r - s) \times (n + r - s)$ résultante.
3. Restaurer la matrice à sa taille originale en réinsérant les éléments de b qui ont été supprimés à l'étape 1.



(a) Étape 1 (b) Étape 2 (c) Étape 3

FIGURE 6 – Schéma de la construction de $b^{+(r:s)}$

Lemma 4.4 *Soient $s_1 := b_{[1 \leq i,j \leq s]}$ et $s_2 := b_{[r < i,j \leq n]}$ deux sous-matrices de b. Si s_1 et s_2 sont transitivement fermés et que $r \leq s$, alors :*

$$b^+ = (b \cup b \star b)^{+(r:s)} \tag{4.4}$$

Démonstration Notons que peu importe la valeur de r et de s, il existe toujours au moins une case $b_{i,j}$ qui appartient à s_1 et à s_2 simultanément.

Soit aussi les deux autres blocs $s_3 := \{b_{i,j} \mid i > s \text{ et } j < r\}$ et $s_4 := \{b_{i,j} \mid i < r \text{ et } j > s\}$. Comme s_3 ne contient que des éléments sous la diagonale principale, chaque case ne contient que l'ensemble vide. On a donc que s_1, s_2 et s_3 sont transitivement clos. Il ne reste qu'à calculer la fermeture transitive de s_4 pour obtenir b^+.

Si l'on entreprend la démarche usuelle de calcul de la fermeture transitive $(\bigcup_{j=1}^{i-1} b^{(j)} \star b^{(i-j)})$, on doit d'abord calculer $b^{(2)}$. De par la configuration de r et s, on peut calculer tous les éléments de s_4 par le simple calcul de $b^{(2)}$. En effet, soit $b_{i,k} \in s_4$, alors on peut l'avoir calculé par la multiplication de $b_{i,k} = \bigcup_{j=1}^{n} b_{i,j} \star b_{j,k}$. Puisque tous les $b_{i,j}$ et les $b_{j,k}$ qui interviennent dans la relation $\star$ appartiennent soit aux groupes déjà clos s_1 et s_2, soit au groupe s_4 que l'on tente de clore. On remplit donc toutes les valeurs possibles de s_4 et on peut en calculer la clôture définitive par l'opération $+(r : s)$ puisque tout le reste de la matrice est déjà sous forme close. Par conséquent, s_4 devient clos après cette opération et l'ensemble de la matrice est transitivement close, montrant que $b^+ = (b \cup b \star b)^{+(r:s)}$. ∎

Exemple Examinons un exemple de l'opération $(b \cup b \star b)^{+(r:s)}$ sur une matrice issue de la grammaire suivante :

- $S \to XY$
- $X \to AB$
- $Y \to BX$
- $A \to a$
- $B \to b$

Prenons le cas où l'on veut reconnaître un mot de 5 caractères. Ainsi, la matrice b devra être de taille 6×6. On peut donc fixer $r := 2$ et $s := 4$ par exemple. La matrice qui est formée par cette grammaire et dont les deux blocs s_1 et s_2 sont transitivement clos est celle-ci :

$$b = \begin{pmatrix} \emptyset & A & X & Y & \emptyset & \emptyset \\ \emptyset & \emptyset & B & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & B & \emptyset & Y \\ \emptyset & \emptyset & \emptyset & \emptyset & A & X \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & B \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}.$$

Le bloc bleu représente la sous-matrice s_1 et le bloc rouge la sous-matrice s_2. On peut observer que chacune d'elle est transitivement close. On calcule facilement

$$b \star b = \begin{pmatrix} \emptyset & \emptyset & X & \emptyset & \emptyset & S \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & X \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}.$$

Par la suite, le calcul de $(b \cup b \star b)^{+(r:s)}$ mène au calcul de la sous-matrice suivante :

$$\begin{pmatrix} \emptyset & A & \emptyset & S \\ \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & B \\ \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}^+ = \begin{pmatrix} \emptyset & A & \emptyset & S \\ \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & B \\ \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}.$$

En recollant la matrice comme demandé par l'opérateur $+(r : s)$, on obtient

$$(b \cup b \star b)^{+(r:s)} = \begin{pmatrix} \emptyset & A & X & Y & \emptyset & S \\ \emptyset & \emptyset & B & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & B & \emptyset & Y \\ \emptyset & \emptyset & \emptyset & \emptyset & A & X \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & B \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}$$

et il est facilement vérifiable que cette matrice est la même que b^+.

Avec la proposition précédente, on peut former la réduction en analysant minutieusement une fonction triplement récursive. Celle-ci a été choisie pour obtenir la complexité souhaitée.

Notons P_k l'opération de la fermeture transitive d'une matrice b pour laquelle les sous-matrices $b_{[1 \leq i,j \leq n-\frac{n}{k}]}$ et $b_{[\frac{n}{k} < i,j \leq n]}$ sont déjà closes. On remarque qu'il s'agit là du cas $r = \frac{n}{k}$ et $s = n - \frac{n}{k} = \frac{n(k-1)}{k}$ de la proposition précédente.

On applique les fonctions suivantes :

- P_2 :
  - (i) Appliquer P_2 à la sous-matrice $[\frac{n}{4} < i,j \leq \frac{3n}{4}]$.
  - (ii) Appliquer P_3 aux sous-matrices $[1 \leq i,j \leq \frac{3n}{4}]$ et $[\frac{n}{4} < i,j \leq n]$ au résultat de (i).
  - (iii) Appliquer P_4 au résultat de (ii).
- P_3 :
  - (i) Calculer $b \cup b \star b$.
  - (ii) Appliquer $+(\frac{n}{3} : \frac{2n}{3})$ au résultat de (i) en utilisant P_2.
- P_4 :
  - (i) Calculer $b \cup b \star b$.
  - (ii) Appliquer $+(\frac{n}{4} : \frac{3n}{4})$ au résultat de (i) en utilisant P_2.

Pour fermer une matrice dont aucun bloc n'est transitivement clos, il faut appeler une fonction P_1 comme celle-ci :

- P_1 :
  - (i) Appliquer P_1 aux sous-matrices $[1 \leq i,j \leq \frac{n}{2}]$ et $[\frac{n}{2} < i,j \leq n]$.
  - (ii) Appliquer P_2 au résultat de (i).

4.3.1 Vérification de la validité de l'algorithme

Démonstration Puisque l'algorithme est récursif (et même triplement récursif!), la vérification de sa validité se fait par induction généralisée sur la taille de la matrice.

Cas de base : Comme les trois procédures servent à fermer une matrice, tous les cas de base sont trivialement vérifiés, puisqu'une matrice 1×1 est nécessairement transitivement fermée.

Étape d'induction : On suppose que les procédures P_2, P_3 et P_4 fonctionnent pour des matrices de taille $i \times i \quad \forall i < n$ et nous allons vérifier que cela fonctionne pour des matrices $n \times n$.

- P_2 **:** L'opération P_2 prend comme hypothèse que les deux blocs $A := b_{[1 \leq i,j \leq \frac{n}{2}]}$ et $B := b_{[\frac{n}{2} < i,j \leq n]}$ sont transitivement clos et retourne une matrice entièrement fermée. Le travail de P_2 consiste donc à fermer la section $[1 \leq i < \frac{n}{2} \leq j \leq n]$.

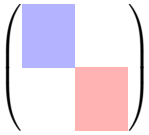


FIGURE 7 – Représentation des blocs A et B qui sont transitivement clos.

Analysons les étapes effectuées par P_2. À l'étape (i), l'hypothèse d'induction nous permet de conclure que l'appel à la fonction P_2 sur la sous-matrice centrale $C := b_{[\frac{n}{4} \leq i,j \leq \frac{3n}{4}]}$ retourne une matrice transitivement close, on obtient la situation illustrée sur la figure 8.

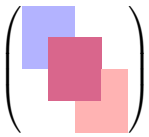


FIGURE 8 – Représentation des blocs A, B et de la partie centrale C, en mauve, qui sont transitivement clos.

La deuxième étape, soit (ii), effectuée par la procédure P_2 requiert d'utiliser la procédure P_3 sur les sous-matrices $[1 \leq i,j \leq \frac{3n}{4}]$ et $[\frac{n}{4} < i,j \leq n]$. Soit m la taille de la sous-matrice envoyée à P_3 et n la taille de la matrice de P_2, on peut vérifier que les hypothèses de fermeture des deux sous-matrices sont vérifiées :

- **De la première sous-matrice :** Puisque $\frac{(k-1)m}{k} = \frac{2(\frac{3n}{4})}{3} = \frac{n}{2}$ et $\frac{m}{k} = \frac{(\frac{3n}{4})}{3} = \frac{n}{4}$, les sections $[1 \leq i,j \leq \frac{2m}{3}] = [1 \leq i,j \leq \frac{n}{2}]$ et $[\frac{m}{3} < i,j \leq m] = [\frac{n}{4} < i,j \leq \frac{3n}{4}]$ sont fermées par les hypothèses de P_2 et par le résultat de la première étape respectivement.
- **De la deuxième sous-matrice :** Les sous-sections concernées par la deuxième sous-matrice ne sont qu'un décalage des matrices trouvées plus haut : $[\frac{n}{4} \leq i,j \leq \frac{3n}{4}]$ et $[\frac{n}{2} < i,j \leq n]$ dont la

première est fermée par l'étape (i) et la seconde par l'hypothèse sur P_2.

Ainsi, comme la matrice envoyée à P_3 est de taille inférieure à $n \times n$ et que les hypothèses sont vérifiées, on peut supposer que les matrices retournées soient transitivement fermées. On représente alors la situation par la figure 9.

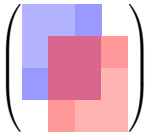


FIGURE 9 – Représentation des blocs fermés après l'étape (ii).

On remarque alors qu'il ne reste que la section $[1 \leq i < \frac{3n}{4} \leq j \leq n]$ à fermer. Finalement, on vérifie que les hypothèses de P_4 sont vérifiées, c'est-à-dire que les blocs $[1 \leq i, j \leq \frac{3n}{4}]$ et $[\frac{n}{4} < i, j \leq n]$ sont fermés, ce qui est trivial par l'observation précédente. Ainsi, la procédure P_4 va correctement fermer l'ensemble de la matrice. On obtient alors, qu'en supposant le fonctionnement de P_3 et de P_4, on a vérifié par induction que la procédure P_2 fonctionnait. Nous allons maintenant vérifier les autres procédures.

- P_3 **:** Comme précédemment, cette procédure prend comme hypothèse que les sous-matrices $[1 \leq i, j \leq \frac{2n}{3}]$ et $[\frac{n}{3} < i, j \leq n]$ sont fermés. Or, on remarque que le processus effectué par P_3 est le même que celui analysé par la proposition 4.4. La seule différence est qu'on utilise P_2 pour calculer la fermeture transitive, mais puisque nous avons déjà vérifié la validité de P_2, cela ne pose pas de problème. Ainsi, cette étape est déjà vérifiée.
- P_4 **:** Similairement, on obtient la même conclusion pour P_4 puisque seules les bornes changent, mais que les hypothèses tiennent toujours. Ainsi, sa vérification se fait aussi avec la proposition 4.4. ∎

Ainsi, la démonstration de la validité montre que, peu importe la taille de la matrice triangulaire, nous pourrons fermer transitivement la matrice par la procédure décrite en ayant les hypothèses de P_2. Dans les cas où la matrice n'aurait pas les bonnes dimensions pour effectuer les divisions entières, on peut tout simplement ajouter des lignes et des colonnes vides pour se ramener à la taille divisible la plus près.

Observons la démonstration que la procédure P_1 permet d'effectuer la fermeture transitive de toute la matrice sans avoir recours aux hypothèses de sous-matrices closes comme avec P_2.

Démonstration Vérifions le résultat par récurrence sur la taille de la matrice.

Cas de base : Comme mentionnée plus haut, la fermeture transitive d'une matrice 1×1 ne nécessite aucune opération.

Étape d'induction : Supposons que P_1 peut fermer une matrice $i \times i$ pour $i < n$ et montrons le résultat pour une matrice $n \times n$. Or ceci est clairement vérifié puisque P_1 effectue deux appels récursifs avec des matrices de taille $\frac{n}{2} \times \frac{n}{2}$. Ces matrices vérifient l'hypothèse d'induction et on suppose que P_1 a fermé ceux-ci. La deuxième étape consiste à appeler P_2 avec la matrice entière, ce qui est valide puisque les hypothèses de P_2 sont vérifié par l'étape (i). L'analyse précédente nous donne ainsi que la fermeture de la matrice en entier est valide et que la procédure P_1 est elle aussi valide. ∎

4.3.2 Analyse de la complexité de l'algorithme

Pour les procédures définies plus haut, nous noterons le temps de calcul $T_i(n)$ pour la procédure P_i d'une matrice $n \times n$.

$$\begin{aligned} T_2(n) &\leq T_2(n/2) + 2T_3(3n/4) + T_4(n) \\ T_3(n) &\leq M(n) \quad + T_2(2n/3) \; + O(n^2) \\ T_4(n) &\leq M(n) \quad + T_2(n/2) \quad + O(n^2) \end{aligned}$$

On voit que toutes les formules peuvent s'exprimer en terme de T_2 :

$$\begin{aligned} T_2(n) &\leq T_2(n/2) + 2T_3(3n/4) + T_4(n) \\ &\leq T_2(n/2) + 2\left[M(3n/4) + T_2(n/2) + O((3n/4)^2)\right] + \left[M(n) + T_2(n/2) + O(n^2)\right] \\ &= 4T_2(n/2) + M(n) + 2M(3n/4) + O(n^2) \\ &\leq 4T_2(n/2) + 3M(n) + O(n^2). \end{aligned}$$

On prend alors des matrices de taille égale à une puissance de deux, on a $n = 2^k$, et on suppose que l'algorithme de multiplication est au moins d'ordre $\Omega(n^d)$ avec $d \geq 2$, c'est-à-dire que $M(2^{k+1}) \geq 2^d M(2^k)$. Cette supposition est tout à fait raisonnable, car il y a un minorant trivial sur le nombre de valeurs dans chacune des matrices multipliées qui justifie $d \geq 2$. On résout la récurrence comme suit :

$$\begin{aligned} T_2(n) &\leq 4T_2(n/2) + 3M(n) + O(n^2) \\ &\leq 4\left[4T_2(n/2^2) + 3M(n/2) + O\left(\left(\frac{n}{2}\right)^2\right)\right] + 3M(n) + O(n^2) \\ &= 4^2 T_2(n/2^2) + 3 \cdot 4^1 M(n/2) + 3M(n) + O(2n^2) \\ &\leq 4^2\left[4T_2(n/2^3) + 3M(n/2^2) + O\left(\left(\frac{n}{2^2}\right)^2\right)\right] + 12M(n/2) + 3M(n) + O(2n^2) \\ &= 4^3 T_2(n/2^3) + 3 \cdot 4^2 M(n/2^2) + 3 \cdot 4^1 M(n/2) + 3M(n) + O(3n^2) \\ &\leq \dots \\ &\leq 4^k T_2(1) + 3\sum_{j=0}^{k-1} 2^{2j} M(n/2^j) + O(kn^2). \end{aligned}$$

Ensuite, puisque le calcul de la fermeture transitive d'une matrice 1×1 est toujours la matrice elle-même, on utilise $T_2(1) = 0$. De plus, on utilise le fait que $\log_2 n = k$ et on applique notre borne inférieure sur la multiplication matricielle.

$$
\begin{aligned}
T_2(n) &\leq 4^k T_2(1) + 3\sum_{j=0}^{k-1} 2^{2j} M(n/2^j) + O(kn^2) \\
&= 3\sum_{j=0}^{\log_2 n - 1} 2^{2j} M(n/2^j) + O(n^2 \log_2 n) \\
&= 3\left[M(n) + \sum_{j=1}^{\log_2 n - 1} 2^{2j} M(n/2^j)\right] + O(n^2 \log_2 n) \\
&\leq 3\left[M(n) + \sum_{j=1}^{\log_2 n - 1} 2^{2j-d} M(n/2^{j-1})\right] + O(n^2 \log_2 n) \\
&= 3\left[M(n) + 2^{(2-d)} M(n) + \sum_{j=2}^{\log_2 n - 1} 2^{2j-d} M(n/2^{j-1})\right] + O(n^2 \log_2 n) \\
&\leq 3\left[M(n) + 2^{(2-d)} M(n) + \sum_{j=2}^{\log_2 n - 1} 2^{2j-2d} M(n/2^{j-2})\right] + O(n^2 \log_2 n) \\
&= 3\left[M(n) + 2^{(2-d)} M(n) + 2^{2(2-d)} M(n) + \sum_{j=3}^{\log_2 n - 1} 2^{2j-2d} M(n/2^{j-2})\right] + O(n^2 \log_2 n) \\
&\leq \dots \\
&\leq 3\left(M(n) + 2^{(2-d)} M(n) + \dots + 2^{(\log_2 n - 1)(2-d)} M(n)\right) + O(n^2 \log_2 n) \\
&= 3M(n) \sum_{j=0}^{\log_2 n - 1} 2^{(2-d)j} + O(n^2 \log_2 n)
\end{aligned}
$$

Au final, on trouve avec la complexité suivante pour des puissances de 2 :

$$T_2(n) \leq 3M(n) \cdot \sum_{j=0}^{\log_2 n - 1} 2^{(2-d)j} + O(n^2 \log_2 n). \tag{4.5}$$

Puisque l'on suppose que notre algorithme de multiplication de matrice est sous-cubique, $2 \leq d < 3$, deux cas sont à vérifier.

- $2 < d < 3$ **:** Alors la série $\sum_{j=0}^{\log_2 n - 1} 2^{(2-d)j}$ est une série géométrique de raison, en valeur absolue, strictement inférieure à 1. Elle converge donc vers une constante indépendante de n.
- $d = 2$ **:** L'exposant s'annule et l'on obtient $\sum_{j=0}^{\log_2 n - 1} 2^0 = \log_2 n$.

Ce qui permet de conclure la complexité suivante :

$$O\big(T_2(n)\big) \subseteq \begin{cases} O\big(M(n)\big) & \text{si } d > 2 \\ O\big(M(n) \cdot \log_2(n)\big) & \text{si } d = 2. \end{cases}$$

On peut ensuite réduire la fermeture transitive $P_{FT}(n)$ de la procédure P_1 en utilisant notre calcul pour $T_2(n)$. On fera les calculs sur le temps de la fermeture transitive, noté $FT(n)$.

$$FT(n) \leq 2FT(n/2) + T_2(n) + O(n^2) \tag{4.6}$$

On peut soit effectuer le même procédé avec les inégalités tel qu'effectué avec T_2 précédemment, ou bien, on peut directement utiliser le théorème général de l'analyse des algorithmes. Nous allons effectuer ce deuxième choix en supposant que la complexité de $T_2(n)$ est de n^d. La forme générale est $R(n) \leq aR(n/b) + O(n^c)$ et on est dans le cas où $\log_b a < c$ puisque $\log_2 2 = 1 < d = \max(2, d)$. Le cas du théorème général associé à cette inégalité nous donne la complexité de $FT(n) \in \Theta(T_2(n))$, ce qui montre que la fermeture transitive de toute la matrice se fait en même complexité que la fermeture avec l'hypothèse de P_2.

$$O\big(FT(n)\big) \subseteq O\big(T_2(n)\big) \subseteq O\big(M(n)\big) \tag{4.7}$$

Si la taille de la matrice n'est pas une puissance de deux, on peut toujours agrandir la matrice à la prochaine puissance de deux et combler les cellules par des ensembles vides.

À la suite de tous les calculs précédents, on peut donc énoncer le théorème suivant.

Théorème 4.5 *Si $M(n)$ et $T_2(n)$ possède la propriété $F(2^{k+1}) \geq 2^d F(2^k)$, où F est soit M ou T_2, alors s'il y a un degré $2 < d < 3$, on a :*

$$P_{FT}(n) \preceq P_M(n). \quad \blacksquare$$

4.4 Réduction $P_M(n) \preceq P_{MB}(n)$

Considérons les multiplications des matrices a et b de résultat c. On rappelle la multiplication $\star$:

$$c_{i,k} = \bigcup_{j=1}^{n} a_{i,j} \star b_{j,k}$$

Maintenant, rendons nos matrices booléennes. Pour ce faire, nous allons allonger les matrices en ajoutant une case par choix possibles parmi l'ensemble $\mathcal{N}_\mathcal{G}$. Ainsi, si $h := |\mathcal{N}_\mathcal{G}|$, nos matrices a et b de taille $n \times n$ deviendront des matrices a' et b' de taille respectivement $hn \times n$ et $n \times hn$. On devra donc effectuer un décalage pour accéder aux éléments de a au travers a' comme suit :

$$a'_{x,y} = 1 \iff A_{x \bmod h} \in a_{\lceil x/h \rceil, y}$$
$$b'_{x,y} = 1 \iff A_{y \bmod h} \in b_{x, \lceil y/h \rceil}$$

Par le produit matriciel usuel, la variable $c_{x,y}$ aura pour valeur 1 s'il y a un $a'_{x,z} = 1$ et un $b'_{z,y} = 1$. En terme de dérivation, cela ne se produira que s'il y a une dérivation $A \to A_i A_j$ tel que :

$$A_{x \bmod h} \in a_{\lceil x/h \rceil, z}$$
$$A_{y \bmod h} \in b_{z, \lceil y/h \rceil}$$

On définit finalement la matrice c en fonction du résultat de la multiplication de la matrice booléenne c' :

$$c_{x,y} = \bigcup_{(A_k \to A_i A_j) \in \mathcal{P}_\mathcal{G}} \{A_k \mid c'_{xh-h+i, yh-h+j} = 1\}. \tag{4.8}$$

Puisque les indices i et j sont utilisées comme des entiers, il suffit de donner une numérotation arbitraire de 1 à h aux symboles non terminaux de la grammaire $\mathcal{G}$.

Théorème 4.6 $O\big(M(n)\big) \subseteq O\big(MB(n) + n^2 \cdot |\mathcal{N}_\mathcal{G}|\big)$. ∎

Ainsi, cette technique nous permet de calculer c par la multiplication des matrices booléennes a' et b' et le corollaire suivant fait le lien final avec le problème initial.

Corollaire 4.7 *Lorsque $M(n) \in O(n^d)$ avec $d > 2$, on obtient :*

$$P_R(n) \preceq P_{FT}(n) \preceq P_M(n) \preceq P_{MB}(n).$$

5 Multiplication matricielle réduite à la reconnaissance

Nous venons de réduire le problème de reconnaissance d'une chaîne de caractères avec une grammaire non contextuelle à un problème de multiplication matricielle. Or, il est aussi possible de faire la réduction inverse ! Voyons celle-ci pour caractériser la complexité de ces deux problèmes.

Pour commencer, définissons la multiplication de matrice booléenne.

Définition Soit A, B et C trois matrices booléennes $m \times m$ telles que $C = A \cdot B$. On définit alors la multiplication comme :

$$c_{i,j} = \bigvee_{k=1}^{m} (a_{i,k} \wedge b_{k,j}).$$

Nous avons aussi besoin d'une définition plus forte de dérivation d'un symbole non terminal.

Définition Définissons que $A \in \mathcal{N}_\mathcal{G}$ *c-dérive* $w_i \dots w_j$ si et seulement si

1. $A \xrightarrow{\star} w_i \dots w_j$ et
2. $S \xrightarrow{\star} w_1 \dots w_{i-1} A w_{j+1} \dots w_n$.

Définition Un *c-analyseur* est un algorithme qui prend une grammaire non contextuelle $\mathcal{G}$ et une chaîne de caractères w et retourne un oracle $\mathcal{F}_{\mathcal{G},w}$ qui possède les trois propriétés suivantes pour tout $A \in \mathcal{N}_\mathcal{G}$.

1. A c-dérive $w_i \dots w_j \implies \mathcal{F}_{\mathcal{G},w} =$ « oui » ;
2. A ne c-dérive pas $w_i \dots w_j \implies \mathcal{F}_{\mathcal{G},w} =$ « non » ;
3. Dans tous les cas, $\mathcal{F}_{\mathcal{G},w}$ donne une réponse en temps constant.

Remarque L'algorithme CYK, que nous avons observé plus tôt, est un exemple de c-analyseur. Le tableau de programmation dynamique que l'algorithme calcule devient un oracle, puisqu'on peut y retrouver l'information nécessaire pour vérifier si un symbole non terminal c-dérive une chaîne ou non.

Nous effectuons la réduction en encodant l'information des deux matrices booléennes dans une grammaire non contextuelle. Nous allons donc exhiber une chaîne de caractères et une grammaire à partir des deux matrices booléennes pour ainsi former le problème de reconnaissance d'un mot à un langage.

5.1 Réduction

Fixons le produit de deux matrices booléennes et leur résultat, $AB = C$, où A, B et C sont des matrices de taille $m \times m$.

En premier temps, nous allons interpréter les matrices A et B comme une grammaire $\mathcal{G}$. L'idée clé est d'introduire les interprétations suivantes :

- **Règles-A :** Pour chaque cellule $a_{i,j} = 1$ de la matrice A, on considère la production $A_{i,j} \to w_i W w_j$ où W est une dérivation quelconque qui produit n'importe quelle chaîne de caractères non nulle.
- **Règles-B :** Pour chaque cellule $b_{i,j} = 1$ de la matrice B, on considère la production $B_{i,j} \to w_{i+1} W w_j$.
- **Règles-C :** Pour chaque cellule $c_{i,j}$ de la matrice C, on associe les productions $C_{i,j} \to A_{i,k} B_{k,j} \quad \forall 1 \leq k \leq m$.

Ainsi, la règle $C_{i,j}$ c-dérive $A_{i,k}B_{k,j}$, qui elle-même c-dérive $w_i W w_k w_{k+1} W w_j$ et ce, si et seulement s'il existe une production $A_{i,k}$ et une production $B_{k,j}$, qui par définition des règles-A et des règles-B signifie qu'il existe $a_{i,k} = 1$ et $b_{k,j} = 1$ pour un certain k. On considère alors que $c_{i,j} = 1$ seulement si $C_{i,j}$ c-dérive $w_i \dots w_j$, ce qui correspond à la multiplication booléenne définit plus haut, mais avec une interprétation liée aux grammaires.

Or, en calculant la taille de la grammaire on trouve qu'il y aura m^2 règles-A, m^2 règles-B et m^3 règles-C pour une taille asymptotique de l'ordre de $O(m^3)$ ce qui est trop imposant pour obtenir le résultat souhaité. Il faut alors utiliser une astuce pour réduire cette taille. Pour ce faire, il faut séparer les indices i, j et k en deux indices chacun. Les premiers représenteront les indices de la matrice et les seconds les indices dans le mot w.

Nous allons donc utiliser les fonctions f_1 et f_2 suivantes pour calculer les nouveaux indices.

$$\begin{aligned} l_1 &:= f_1(l) := \lfloor l/n \rfloor \\ l_2 &:= f_2(l) := (l \mod n) + 2 \end{aligned} \tag{5.1}$$

On prend $n = \lceil m^{1/3} \rceil$, car, comme on le verra, cela donne la borne souhaitée. En calculant le résultat de ces fonctions pour $i_1, i_2, j_1, j_2, k_1, k_2$, on obtient six indices qui respectent les inégalités suivantes :

$$\begin{aligned} &0 \leq i_1, j_1, k_1 \leq \left\lfloor \frac{m}{\lceil m^{1/3} \rceil} \right\rfloor \leq m^{2/3} \leq n^2 \\ &2 \leq i_2, j_2, k_2 \leq n + 1. \end{aligned} \tag{5.2}$$

Il est donc clair que i_1 et i_2 représentent essentiellement la partie entière et le reste de la division euclidienne et qu'avec ces deux indices, on peut reconstruire i. Le choix d'additionner 2 dans la fonction f_2 permet d'éviter la création de règles de dérivation nulle, ce que l'on souhaite éviter dans notre grammaire. Pour vérifier l'égalité de i avec un i', il suffira d'avoir l'égalité $i_1 = i'_1$ qui sera vérifié par la grammaire et d'avoir l'égalité $i_2 = i'_2$ qui sera vérifié par le mot à reconnaître.

Il faut maintenant définir une grammaire et une chaîne de caractères qui encode l'information de la multiplication $AB = C$. Nous choisissons donc une grammaire ayant comme symboles non terminaux $\mathcal{N}_\mathcal{G} := \{w_l \mid 1 \leq l \leq 3n + 6\}$. Le mot à reconnaître sera $w = w_1 w_2 \dots w_{3n+6}$. En posant $\delta := n + 2$, w sera de taille 3δ ce qui sépare le mot en trois parties :

$$\underbrace{w_1 w_2 \dots w_{n+2}}_{\text{partie 1}} \underbrace{w_{n+3} \dots w_{2n+4}}_{\text{partie 2}} \underbrace{w_{2n+5} \dots w_{3n+6}}_{\text{partie 3}}.$$

Les symboles d'indices i_2, $i_2+\delta$ et $i_2+2\delta$ appartiendront chacun à une partie différente du mot.

On crée alors la grammaire avec les productions suivantes basées sur la première grammaire que nous avons accomplie. On doit modifier celle-ci pour tenir compte des nouveaux indices dans les règles-{A,B,C}. On ajoute aussi les règles-W, qui formalisent le choix de W, et finalement les règles-S qui donne les dérivations du symbole de départ S.

- **Règles-W :** Pour chaque symbole terminal de w, on ajoute $W \to w_l W \mid w_l, \quad 1 \leq l \leq 3n+6$.
- **Règles-A :** Pour chaque cellule $a_{i,j} = 1$ de la matrice A, on considère la production $A_{i_1,j_1} \to w_{i_2} W w_{j_2+\delta}$.
- **Règles-B :** Pour chaque cellule $b_{i,j} = 1$ de la matrice B, on considère la production $B_{i_1,j_1} \to w_{i_2+\delta+1} W w_{j_2+2\delta}$.
- **Règles-C :** Pour chaque cellule $c_{i,j}$ de la matrice C, on associe les productions $C_{p,q} \to A_{p,r} B_{r,q} \quad \forall\, 1 \leq p,q,r \leq n^2$.
- **Règles-S :** Finalement, on impose le symbole de départ avec les dérivations $S \to W C_{p,q} W \quad 1 \leq p,q \leq n^2$.

Cela signifie qu'une sous-chaîne c-dérivée de A_{i_1,k_1} débute dans la partie 1 du mot et se termine dans la partie 2. De même, une sous-chaîne c-dérivée de B_{k_1,j_1} commence à la suite de la précédente, dans la partie 2, et se termine dans la partie 3. On l'exprime par les expressions : A_{i_1,k_1} c-dérive $w_{i_2} \dots w_{k_2+\delta}$ et B_{k_1,j_1} c-dérive $w_{k_2+\delta+1} \dots w_{j_2+2\delta}$.

Avec cette modification, le nombre de productions engendré par la règle-C est de $(n^2)^3 \approx m^2$. L'ajout des dérivations de S reste inférieur avec $n^4 \approx m^{4/3} \in O(m^2)$. Par conséquent, le nombre total de règles dans la grammaire est maintenant de l'ordre de $O(m^2)$. Bien que cela réduise la taille de la matrice, il faudra faire attention à cette modification, car elle impose de connaître i_2 pour reconstruire i et cette information est maintenant contenue dans le mot w.

Pour montrer que la réduction des matrices booléennes A et B à la grammaire $\mathcal{G}$ et du mot w fonctionne, nous avons besoin du théorème suivant.

Théorème 5.1 *Pour $1 \leq i,j \leq m$, les cellules $c_{i,j}$ de la matrice C sont non-nulle si et seulement si C_{i_1,j_1} c-dérive $w_{i_2} \dots w_{j_2+2\delta}$.*

Démonstration Soit $1 \leq i,j \leq m$ quelconque.

$\Longrightarrow$: Avec l'hypothèse que $c_{i,j} = 1$, montrons que C_{i_1,j_1} c-dérive $w_{i_2} \dots w_{j_2+2\delta}$. Pour ce faire, il faut vérifier les deux propriétés de la c-dérivation. Remarquons que puisque $c_{i,j} = 1$, il existe un k tel que $a_{i,k} = b_{k,j} = 1$.

1. Montrons que $C_{i_1,j_1} \overset{\star}{\to} w_{i_2} \dots w_{j_2+2\delta}$.

$$
\begin{aligned}
C_{i_1,j_1} &\to A_{i_1,k_1} B_{k_1,j_1} && \text{(Par les règles-C)} \\
&\overset{\star}{\to} \underbrace{w_{i_2} W w_{k_2+\delta}}\, \underbrace{w_{k_2+\delta+1} W w_{j_2+2\delta}} && \text{(Par l'existence d'un } k\text{)} \\
&\overset{\star}{\to} w_{i_2} \dots w_{j_2+2\delta} && \text{(Par les règles-W)}
\end{aligned}
$$

La seule vérification qui reste à faire est de vérifier que les règles-W génèrent des chaînes qui contiennent au moins un symbole.

- Pour le premier cas, celui de la section $w_{i_2+1} \dots w_{k_2+\delta-2}$, il suffit de montrer que $i_2 + 1 < k_2 + \delta - 1$. Or, on sait que i_2 se trouve dans la partie 1 du mot et que $k_2 + \delta$ se trouve dans la partie 2 et par les inégalités (5.2), on trouve facilement que

$$i_2 + 1 \leq (n+1) + 1 = \delta < \delta + 1 = 2 + \delta - 1 \leq k_2 + \delta - 1.$$

- Le deuxième cas est presque identique avec la section générée par la règle-B. On doit montrer que $k_2 + \delta + 1 < j_2 + 2\delta - 1$. Par un raisonnement identique sur les parties 2 et 3, on trouve que

$$k_2 + \delta + 1 \leq (n+1) + \delta + 1 = 2\delta < 2\delta + 1 = 2 + 2\delta - 1 \leq j_2 + 2\delta - 1.$$

Par conséquent, dans les deux cas, W dérive au moins un symbole dans la règle-A et la règle-B, ce qui montre que la dérivation est valide pour $C_{i,j}$.

2. Montrons que $S \xrightarrow{\star} w_1 \dots w_{i_2-1} C_{i_1,j_1} w_{j_2+2\delta+1} \dots w_{3\delta}$. Cela est évident par la dérivation des règles-S $S \to W C_{i_1,j_1} W$, mais il faut tout de même vérifier que les W contiennent au moins un symbole.
   - Dans le premier cas, on doit montrer que les symboles qui précédent w_{i_2} contiennent au moins un élément, ce qui signifie que $1 < i_2$. Or, ceci est évident puisque $i_2 \geq 2$.
   - De même, on doit montrer qu'il y a au moins un symbole qui suit w_{j_2}. Or, $j_2 + 2\delta \leq n + 1 + 2\delta = 3\delta - 1 < 3\delta$.

   Ainsi, C_{i_1,j_1} peut être dérivé par S et la deuxième propriété est vérifiée.

$\Longleftarrow$: On veut maintenant montrer que $c_{i,j} = 1$ sachant que C_{i_1,j_1} c-dérive $w_{i_2} \dots w_{j_2+2\delta}$. Comme C_{i_1,j_1} c-dérive la chaîne $w_{i_2} \dots w_{j_2+2\delta}$, on sait qu'il existe une règle-C qui dérive la règle suivante pour un certain k_1 et un certain k_2.

$$\begin{aligned} C_{i_1,j_1} &\to A_{i_1,k_1} A_{k_1,j_1} \\ &\xrightarrow{\star} \underbrace{w_{i_2} W w_{k_2+\delta}} \underbrace{w_{k_2+\delta+1} W w_{j_2+2\delta}} \\ &\xrightarrow{\star} w_{i_2} \dots w_{j_2+2\delta}. \end{aligned}$$

Sachant k_1 et k_2, on retrouve $k = k_1 n + (k_2 - 2)$ qui a causé cette décomposition. Ainsi, les deux cellules $a_{i,k} = 1$ et $b_{k,j} = 1$ ce qui nous permet de conclure que $c_{i,j} = 1$ par notre définition de multiplication de matrices booléennes.

Puisque i et j étaient quelconque, l'énoncé est vrai $\forall\, 1 \leq i, j \leq m$. ∎

Les deux corollaires suivants découlent directement de la preuve précédente.

Corollaire 5.2 *Pour $1 \leq i, j \leq m$, on a que $c_{i,j} = 1$ si et seulement si C_{i_1,j_1} dérive $w_{i_2} \dots w_{j_2+2\delta}$. Autrement dit, la dérivation implique la c-dérivation pour la grammaire proposée.*

Corollaire 5.3 *S dérive w si et seulement si C n'est pas une matrice nulle.*

En récapitulant les règles et les éléments de la grammaire, on utilise :
- $|\mathcal{G}| \cdot (3n + 6) \in O(n \cdot |\mathcal{G}|) \approx O(m^{1/3} \cdot |\mathcal{G}|)$ règles-W ;
- $O(m^2)$ règles-A, puisqu'au plus chaque cellule de la matrice est un 1 ;
- $O(m^2)$ règles-B pour la même raison ;
- $(n^2)^3 \in O(m^2)$ règles-C ;
- $(n^2)^2 \in O(m^{4/3})$ règles-S.

La taille totale de notre matrice est alors de l'ordre de $O(m^2 \cdot |\mathcal{G}|)$ ce qui est la taille optimale puisque la borne informationnelle des valeurs d'une matrice $m \times m$ est de m^2 éléments.

Remarque Avec la grammaire produite par cette procédure, on voudrait probablement la transformer en forme normale de Chomsky. Il existe des algorithmes pour effectuer cette transformation, mais on souhaite conserver la même taille asymptotique optimale de notre grammaire. Or l'algorithme standard pour effectuer cette transformation procède en quatre étapes :

1. Créer un nouveau symbole de départ qui pointe sur l'ancien $S_0 \to S$.
2. Éliminer les règles de la forme $A \to \epsilon$. Par exemple :
   - $S \to AbA \mid B$ $\quad$ - $B \to b|c$ $\quad$ - $A \to \epsilon$

   va devenir après avoir éliminé les règles ϵ :
   - $S \to AbA \mid Ab \mid bA \mid b \mid B$ $\quad$ - $B \to b|c$.
3. Éliminer les règles unitaires de la forme $A \to B$.
4. Ajouter des règles intermédiaires pour obtenir la forme normale de Chomsky. Par exemple, pour la règle :
   - $A \to abcd$

   on ajoutera des règles intermédiaires pour obtenir :
   - $A \to aA_1$ $\quad$ - $A_1 \to bA_2$ $\quad$ - $A_2 \to cA_3$ $\quad$ - $A_3 \to d$.

Maintenant, dans notre cas, avec la grammaire que nous avons construite, nous n'avons pas de règle ϵ, aussi toutes nos règles-C sont de la forme $C \to AB$, ce qui est déjà en forme normale de Chomsky. Les règles-A et les règles-B sont de la forme $A \to w_i W w_j$ que l'on peut modifier avec l'ajout d'une seule variable intermédiaire en :
- $A \to w_i A_1$ $\quad$ - $A_1 \to W w_j$.

On effectue la même opération pour les règles-S, tandis que les règles-W sont déjà dans la bonne forme.

Essentiellement, on obtient donc la même taille de grammaire asymptotiquement puisque l'on ne fait que doubler certaines règles. On peut alors considérer la grammaire comme si elle était en forme normale de Chomsky en conservant l'optimalité en taille de celle-ci.

5.2 Borne de temps

Théorème 5.4 *Soit un analyseur qui reconnaît un mot w de taille n d'une grammaire $\mathcal{G}$ et soit m la taille de deux matrices booléennes carrées A et B. Alors, si l'analyseur s'exécute en $O(T(|\mathcal{G}|) \cdot t(n))$, il peut être converti en un algorithme de multiplication de matrices booléennes en temps $O(m^2 + T(m^2) \cdot t(m^{1/3})$.*

Démonstration Avec l'algorithme de multiplication de matrice $\mathcal{M}$ qui est issu de la réduction précédente, on produit l'oracle $\mathcal{F}_{\mathcal{G},w}$. On peut alors utiliser cet oracle pour effectuer l'algorithme $\mathcal{M}$ en demandant à l'oracle si C_{i_1,j_1} dérive la chaîne $w_{i_2} \dots w_{j_2+2\delta}$, ce qui permet de déterminer la valeur de $c_{i,j}$. Comme l'oracle répond en temps constant et qu'il faut demander la valeur de chaque case de C, cela prend de l'ordre de $O(m^2)$ opérations. On prend alors le maximum entre le temps de construire l'oracle ($O(T(m^2) \cdot t(m^{1/3}))$ puisque $\mathcal{G}$ est de taille m^2 comme analysée précédemment et que le mot w est de taille $O(m^{1/3})$) et celui de demander si la grammaire dérive chaque expression associée à chaque case. Ainsi il prend effectivement un temps de l'ordre de $O(m^2+T(m^2)\cdot t(m^{1/3}))$ pour multiplier des matrices booléennes. ∎

Corollaire 5.5 *En particulier, si l'analyseur prend $O(n^d \cdot |\mathcal{G}|)$, alors l'algorithme de multiplication de matrices associé prendra $O(m^{2+d/3})$.*

Démonstration Puisque dans notre cas $T(|\mathcal{G}|) = |\mathcal{G}|$ et $t(n) = n^d$, où n^d représente la complexité d'un algorithme de reconnaissance de langage par une grammaire non contextuelle. On sait qu'il existe au moins un algorithme tel que $d < 3$ tel que montré par Valiant. Ainsi, les calculs suivants suivent immédiatement du théorème précédent :

$$\begin{aligned} O\left(m^2 + T(m^2) \cdot t(m^{1/3})\right) &= O\left(m^2 + m^2 \cdot (m^{1/3})^d\right) \\ &= O\left(m^{2+d/3}\right). \end{aligned}$$

Avec $d < 3$ on obtient une complexité sous-cubique. ∎

Au final, ce théorème exprime une relation entre le temps de calcul du problème de multiplication de matrices et du problème de reconnaissance de chaînes de caractères par une grammaire non contextuelle. Ainsi, si l'on trouve un algorithme de reconnaissance de chaîne de caractères de complexité sous-cubique, on peut en déduire un algorithme de multiplication de matrice sous-cubique. De même, si l'on trouve une borne minimale pour le problème de multiplication de matrices booléennes de la forme $\Omega(n^{3-d})$, on obtient immédiatement une borne minimale $\Omega(n^{3-3d} \cdot |\mathcal{G}|)$ pour le problème de reconnaissance.

6 Modèle mathématique

On peut utiliser la réduction de Valiant de la section 4.3 pour concevoir un modèle de résolution du problème de reconnaissance dans les temps calculés. Dans cette section, nous énoncerons un modèle qui utilise la réduction de Valiant et l'algorithme de multiplication de matrice Strassen. Les deux algorithmes sont détaillés ci-bas.

6.1 Algorithme de réduction

Cet algorithme sert à produire un modèle qui reconnaît une chaîne de caractères. Étant donné que le modèle dépend hautement du problème source, il est plus judicieux de fournir une fonction qui prend les variables du problème en entrées et qui retourne un ensemble de variables intermédiaires V et un ensemble de contraintes $\mathcal{C}$.

Remarquons cependant que le modèle est directement basé sur les affectations effectuées par la procédure des trois fonctions récursives de Valiant. L'idée pour la conception du modèle est d'implémenter cet algorithme, mais de remplacer toutes les affectations de variables par l'ajout de contraintes à l'ensemble $\mathcal{C}$, en ajoutant aussi toutes les variables intermédiaires à la collection V.

Cet algorithme est implémenté dans l'annexe A. Il crée un modèle avec de l'ordre de $O(M(n))$ variables avec autant de contraintes.

6.2 Algorithme de Strassen

Cette section vise à faire le modèle de l'algorithme de Strassen. Toutefois, avant d'en examiner les modèles, rappelons les éléments de l'algorithme sous forme mathématique.

Proposition 6.1 *On définit l'algorithme de Strassen pour la multiplication $AB = C$ comme suit :*

Soit les partitions des matrices A, B et C suivantes :

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}, B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}, C = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix}.$$

Alors on peut effectuer la multiplication matricielle en calculant les multiplications suivantes :

$$\begin{aligned} M_1 &:= (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2}) \\ M_2 &:= (A_{2,1} + A_{2,2})B_{1,1} \\ M_3 &:= A_{1,1}(B_{1,2} - B_{2,2}) \\ M_4 &:= A_{2,2}(B_{2,1} - B_{1,1}) \\ M_5 &:= (A_{1,1} + A_{1,2})B_{2,2} \\ M_6 &:= (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2}) \\ M_7 &:= (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2}). \end{aligned}$$

et en additionnant ces nouvelles variables de la manière suivante, on obtient C :

$$\begin{aligned} C_{1,1} &:= M_1 + M_4 - M_5 + M_7 \\ C_{1,2} &:= M_3 + M_5 \\ C_{2,1} &:= M_2 + M_4 \\ C_{2,2} &:= M_1 - M_2 + M_3 + M_6. \end{aligned}$$

Démonstration La preuve se déduit facilement des relations trouvées, il suffit de remplacer les valeurs des $C_{i,j}$ et des M_k et on remarque que la multiplication de Strassen effectue le même résultat que l'algorithme de multiplication naïf en seulement sept multiplications.

Affirmation 6.2 *L'algorithme de Strassen à une complexité de* $O(n^{\log_2 7}) \approx O(n^{2.80735})$.

On construit le modèle comme on a construit celui de Valiant, c'est-à-dire qu'on exhibe une fonction qui va construire le modèle en prenant les variables initiales en entrées. Cette fonction est basée sur la même astuce que les précédentes, les affectations sont remplacées en contraintes dans le modèle et les variables intermédiaires sont conservées. La fonction est définie dans l'annexe B. Cette fonction produit un modèle qui comprend de l'ordre de $O(n^{\log_2 7})$ variables et autant de contraintes.

7 Conclusion

En somme, ce rapport présente diverses façons d'utiliser les automates et les grammaires non contextuelles dans des contextes d'optimisation, que ce soit linéaire ou combinatoire. Ceux-ci sont essentiels pour résoudre le problème de reconnaissance d'une chaîne de caractères. Les réductions faites dans les sections 4 et 5, montre que le problème de reconnaissance d'une grammaire non contextuelle nécessite un algorithme de multiplication de matrices booléennes rapide et que le problème de multiplication de matrices booléennes nécessite un algorithme de reconnaissance rapide. Malheureusement, dans la littérature, il n'existe pas de façon de découler un algorithme de filtrage aussi performant que la réduction de Valiant que l'on puisse utiliser dans un solveur.

8 Remerciements

Je tiens à remercier sincèrement Claude-Guy Quimper qui a été mon professeur coordonnateur pour le projet et m'a donné de précieux conseils. Merci également à tous mes collègues qui m'ont aidé à la relecture de ce rapport.

A Modèle de l'algorithme de réduction

1.1 Fonction $TransitiveClosure(\mathcal{G}, [X_1, \ldots, X_m])$

Algorithm 3 $TransitiveClosure(\mathcal{G}, [X_1, \ldots, X_m])$

Ensure: $\mathcal{G}$ est une grammaire non contextuelle.
$\mathcal{C} \leftarrow \emptyset$
$n \leftarrow 2^{\lceil \log_2 m \rceil}$
$b[1..n, 1..n]$ où $dom(b[i,j]) = \mathcal{N}_\mathcal{G}$
for $i = 1$ **to** m **do**
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{b[i, i+1] \in \{A_k \mid (A_k \rightarrow x_i) \in \mathcal{P}_\mathcal{G} \text{ et } x_i \in dom(X_i)\}\}$
$(\mathcal{C}_1, V) \leftarrow P_1(\mathcal{G}, b, n)$
return $(\mathcal{C} \cup \mathcal{C}_1, V \cup b)$

1.2 Fonction $P_1(\mathcal{G}, b, m)$

Algorithm 4 $P_1(\mathcal{G}, b, m)$

Ensure: b est une matrice carrée de taille $m \times m$.
Ensure: $\mathcal{G}$ est une grammaire non contextuelle.
if $m > 1$ **then**
    $m_{1/2} \leftarrow \lceil m/2 \rceil$
    resize b **to** $2 \cdot m_{1/2}$

    $(\mathcal{C}, b_{P_1, m_{1/2}, 1}) \leftarrow P_1(\mathcal{G}, b[1 \leq i, j < m_{1/2}], m_{1/2})$
    for $i = 1$ **to** $m_{1/2}$ **do**
        for $j = 1$ **to** $m_{1/2}$ **do**
            $\mathcal{C} \leftarrow \mathcal{C} \cup \{b[i,j] = b_{P_1, m_{1/2}, 1}[i,j]\}$

    $(\mathcal{C}_{P_1}, b_{P_1, m_{1/2}, 2}) \leftarrow P_1(\mathcal{G}, b[m_{1/2} \leq i, j \leq m], m_{1/2})$
    $\mathcal{C} \leftarrow \mathcal{C}_{P_1}$
    for $i = m_{1/2}$ **to** m **do**
        for $j = m_{1/2}$ **to** m **do**
            $\mathcal{C} \leftarrow \mathcal{C} \cup \{b[i,j] = b_{P_1, m_{1/2}, 2}[i,j]\}$

    $(\mathcal{C}_{P_2}, b_{P_2, m}) \leftarrow P_2(\mathcal{G}, b, m)$
    $\mathcal{C} \leftarrow \mathcal{C}_{P_2}$
    for $i = 1$ **to** m **do**
        for $j = 1$ **to** m **do**
            $\mathcal{C} \leftarrow \mathcal{C} \cup \{b[i,j] = b_{P_2, m}[i,j]\}$
$V \leftarrow b_{P_1, m_{1/2}, 1} \cup b_{P_1, m_{1/2}, 2} \cup b_{P_2, m}$
return $(\mathcal{C}, V)$

1.3 Fonction $P_2(\mathcal{G}, b, m)$

Algorithm 5 $P_2(\mathcal{G}, b, m)$

Ensure: b est une matrice carrée de taille $m \times m$.
Ensure: $\mathcal{G}$ est une grammaire non contextuelle.
if $m > 1$ **then**
  $m_{1/4} \leftarrow \lceil m/4 \rceil$
  resize b **to** $4 \cdot m_{1/4}$

  $(\mathcal{C}, b_{P_2,m_{1/4}}) \leftarrow P_2(\mathcal{G}, b[m_{1/4} < i, j \leq 3 \cdot m_{1/4}], 2 \cdot m_{1/4})$
  for $i = m_{1/4}$ **to** $3 \cdot m_{1/4}$ **do**
    for $j = m_{1/4}$ **to** $3 \cdot m_{1/4}$ **do**
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{b[i,j] = b_{P_2,m_{1/4}}[i,j]\}$

  $(\mathcal{C}_{P_3,1}, b_{P_3,m_{1/4},1}) \leftarrow P_3(\mathcal{G}, b[1 < i, j \leq 3 \cdot m_{1/4}], 3 \cdot m_{1/4})$
  $(\mathcal{C}_{P_3,2}, b_{P_3,m_{1/4},2}) \leftarrow P_3(\mathcal{G}, b[m_{1/4} < i, j \leq m], 3 \cdot m_{1/4})$
  $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}_{P_3,1} \cup \mathcal{C}_{P_3,2}$
  for $i = 1$ **to** $3 \cdot m_{1/4}$ **do**
    for $j = 1$ **to** $3 \cdot m_{1/4}$ **do**
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{b[i,j] = b_{P_3,m_{1/4},1}[i,j]\}$
  for $i = m_{1/4}$ **to** m **do**
    for $j = m_{1/4}$ **to** m **do**
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{b[i,j] = b_{P_3,m_{1/4},2}[i,j]\}$

  $(\mathcal{C}_{P_4}, b_{P_4,m_{1/4}}) \leftarrow P_4(\mathcal{G}, b, m)$
  $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}_{P_4}$
  for $i = 1$ **to** m **do**
    for $j = 1$ **to** m **do**
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{b[i,j] = b_{P_4,m_{1/4}}[i,j]\}$
$V \leftarrow b_{P_2,m_{1/4}} \cup b_{P_3,m_{1/4},1} \cup b_{P_3,m_{1/4},2} \cup b_{P_4,m_{1/4}}$
return $(\mathcal{C}, V)$

1.4 Fonction $P_3(\mathcal{G}, b, m)$

Algorithm 6 $P_3(\mathcal{G}, b, m)$

Ensure: b est une matrice carrée de taille $m \times m$.
Ensure: $\mathcal{G}$ est une grammaire non contextuelle.
$(\mathcal{C}, b^{(2)}) \leftarrow \text{multiply}(\mathcal{G}, b, b)$
$b'_m[1..m, 1..m]$
for $i = 1$ **to** m **do**
  for $j = 1$ **to** m **do**
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{b'_m[i,j] = b[i,j] \cup b^{(2)}[i,j]\}$

6: $m_{1/3} \leftarrow \lceil m/3 \rceil$
7: $s_{m_{1/3}}[1..2m_{1/3}, 1..2m_{1/3}]$
8: **for** $i = 1$ **to** $m_{1/3}$ **do**
9: **for** $j = 1$ **to** $m_{1/3}$ **do**
10: $\mathcal{C} \leftarrow \mathcal{C} \cup \{s_{m_{1/3}}[i,j] = b'_m[i,j]\}$
11: **for** $j = m_{1/3}$ **to** $m - 2m_{1/3}$ **do**
12: $\mathcal{C} \leftarrow \mathcal{C} \cup \{s_{m_{1/3}}[i,j] = b'_m[i, j + m_{1/3}]\}$
13: **for** $i = m_{1/3}$ **to** $m - 2m_{1/3}$ **do**
14: **for** $j = 1$ **to** $m_{1/3}$ **do**
15: $\mathcal{C} \leftarrow \mathcal{C} \cup \{s_{m_{1/3}}[i,j] = b'_m[i + m_{1/3}, j]\}$
16: **for** $j = m_{1/3}$ **to** $m - 2m_{1/3}$ **do**
17: $\mathcal{C} \leftarrow \mathcal{C} \cup \{s_{m_{1/3}}[i,j] = b'_m[i + m_{1/3}, j + m_{1/3}]\}$
18: $(\mathcal{C}_{P_2}, b_{P_2}) \leftarrow P_2(\mathcal{G}, s_{m_{1/3}}[1..2m_{1/3}, 1..2m_{1/3}], m_{1/3})$
19: **return** $(\mathcal{C} \cup \mathcal{C}_{P_2}, b'_m \cup s_{m_{1/3}} \cup b_{P_2})$

1.5 Fonction $P_4(\mathcal{G}, b, m)$

Algorithm 7 $P_4(\mathcal{G}, b, m)$

Ensure: b est une matrice carrée de taille $m \times m$.
Ensure: $\mathcal{G}$ est une grammaire non contextuelle.
1: $(\mathcal{C}, b^{(2)}) \leftarrow \text{multiply}(\mathcal{G}, b, b)$
2: $b'_m[1..m, 1..m]$
3: **for** $i = 1$ **to** m **do**
4: **for** $j = 1$ **to** m **do**
5: $\mathcal{C} \leftarrow \mathcal{C} \cup \{b'_m[i,j] = b[i,j] \cup b^{(2)}[i,j]\}$
6:
7: $m_{1/4} \leftarrow \lceil m/4 \rceil$
8: $s_{m_{1/4}}[1..2m_{1/4}, 1..2m_{1/4}]$
9: **for** $i = 1$ **to** $m_{1/4}$ **do**
10: **for** $j = 1$ **to** $m_{1/4}$ **do**
11: $\mathcal{C} \leftarrow \mathcal{C} \cup \{s_{m_{1/4}}[i,j] = b'_m[i,j]\}$
12: **for** $j = m_{1/4}$ **to** $m - 3m_{1/4}$ **do**
13: $\mathcal{C} \leftarrow \mathcal{C} \cup \{s_{m_{1/4}}[i,j] = b'_m[i, j + 2m_{1/4}]\}$
14: **for** $i = m_{1/4}$ **to** $m - 3m_{1/4}$ **do**
15: **for** $j = 1$ **to** $m_{1/4}$ **do**
16: $\mathcal{C} \leftarrow \mathcal{C} \cup \{s_{m_{1/4}}[i,j] = b'_m[i + 2m_{1/4}, j]\}$
17: **for** $j = m_{1/4}$ **to** $m - 3m_{1/4}$ **do**
18: $\mathcal{C} \leftarrow \mathcal{C} \cup \{s_{m_{1/4}}[i,j] = b'_m[i + 2m_{1/4}, j + 2m_{1/4}]\}$
19: $(\mathcal{C}_{P_2}, b_{P_2}) \leftarrow P_2(\mathcal{G}, s_{m_{1/4}}[1..2m_{1/4}, 1..2m_{1/4}], 2m_{1/4})$
20: **return** $(\mathcal{C} \cup \mathcal{C}_{P_2}, b'_m \cup s_{m_{1/4}} \cup b_{P_2})$

B Modèle de l'algorithme de Strassen

2.1 Fonction $multiply\_strassen(\mathcal{G}, A, B)$

Algorithm 8 $multiply\_strassen(\mathcal{G}, A, B)$

Ensure: A et B sont des matrices carrées de taille $m \times m$.
Ensure: $\mathcal{G}$ est une grammaire non contextuelle.

if $n = 1$ **then**
  return $\{A_i \in \mathcal{N}_\mathcal{G} \mid \exists A_j \in A, A_k \in B \text{ tel que } (A_i \to A_j A_k) \in \mathcal{P}_\mathcal{G}\}$
else
  $m_{1/2} \leftarrow \lceil m/2 \rceil$
  $A^\star_m \leftarrow A$
  $B^\star_m \leftarrow B$
  resize $A^\star_m$ **to** $2 \cdot m_{1/2}$
  resize $B^\star_m$ **to** $2 \cdot m_{1/2}$

  $A_{m,1}[1..m_{1/2}, 1..m_{1/2}], B_{m,1}[1..m_{1/2}, 1..m_{1/2}]$
  $A_{m,2}[1..m_{1/2}, 1..m_{1/2}], B_{m,2}[1..m_{1/2}, 1..m_{1/2}]$
  $A_{m,3}[1..m_{1/2}, 1..m_{1/2}], B_{m,3}[1..m_{1/2}, 1..m_{1/2}]$
  $A_{m,4}[1..m_{1/2}, 1..m_{1/2}], B_{m,4}[1..m_{1/2}, 1..m_{1/2}]$
  $A_{m,5}[1..m_{1/2}, 1..m_{1/2}], B_{m,5}[1..m_{1/2}, 1..m_{1/2}]$
  $A_{m,6}[1..m_{1/2}, 1..m_{1/2}], B_{m,6}[1..m_{1/2}, 1..m_{1/2}]$
  $A_{m,7}[1..m_{1/2}, 1..m_{1/2}], B_{m,7}[1..m_{1/2}, 1..m_{1/2}]$
  for $i = 1$ **to** $m_{1/2}$ **do**
    for $j = 1$ **to** $m_{1/2}$ **do**
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{A_{m,1}[i,j] = A^\star_m[i,j] + A^\star_m[i + m_{1/2}, j + m_{1/2}]\}$
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{B_{m,1}[i,j] = B^\star_m[i,j] + B^\star_m[i + m_{1/2}, j + m_{1/2}]\}$
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{A_{m,2}[i,j] = A^\star_m[i + m_{1/2}, j] + A^\star_m[i + m_{1/2}, j + m_{1/2}]\}$
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{B_{m,2}[i,j] = B^\star_m[i,j]\}$
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{A_{m,3}[i,j] = A^\star_m[i,j]\}$
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{B_{m,3}[i,j] = B^\star_m[i, j + m_{1/2}] - B^\star_m[i + m_{1/2}, j + m_{1/2}]\}$
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{A_{m,4}[i,j] = A^\star_m[i + m_{1/2}, j + m_{1/2}]\}$
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{B_{m,4}[i,j] = B^\star_m[i + m_{1/2}, j] - B^\star_m[i,j]\}$
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{A_{m,5}[i,j] = A^\star_m[i,j] + A^\star_m[i, j + m_{1/2}]\}$
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{B_{m,5}[i,j] = B^\star_m[i + m_{1/2}, j + m_{1/2}]\}$
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{A_{m,6}[i,j] = A^\star_m[i + m_{1/2}, j] - A^\star_m[i,j]\}$
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{B_{m,6}[i,j] = B^\star_m[i,j] + B^\star_m[i, j + m_{1/2}]\}$
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{A_{m,7}[i,j] = A^\star_m[i, j + m_{1/2}] - A^\star_m[i + m_{1/2}, j + m_{1/2}]\}$
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{B_{m,7}[i,j] = B^\star_m[i + m_{1/2}, j] + B^\star_m[i + m_{1/2}, j + m_{1/2}]\}$

  $(\mathcal{C}_1, V_1, M_{m,1}) \leftarrow multiply\_strassen(\mathcal{G}, A_1, B_1)$
  $(\mathcal{C}_2, V_2, M_{m,2}) \leftarrow multiply\_strassen(\mathcal{G}, A_2, B_2)$
  $(\mathcal{C}_3, V_3, M_{m,3}) \leftarrow multiply\_strassen(\mathcal{G}, A_3, B_3)$
  $(\mathcal{C}_4, V_4, M_{m,4}) \leftarrow multiply\_strassen(\mathcal{G}, A_4, B_4)$
  $(\mathcal{C}_5, V_5, M_{m,5}) \leftarrow multiply\_strassen(\mathcal{G}, A_5, B_5)$
  $(\mathcal{C}_6, V_6, M_{m,6}) \leftarrow multiply\_strassen(\mathcal{G}, A_6, B_6)$
  $(\mathcal{C}_7, V_7, M_{m,7}) \leftarrow multiply\_strassen(\mathcal{G}, A_7, B_7)$
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{\bigcup_{i=1}^{7} \mathcal{C}_i\}$

43: $C_m[1..m, 1..m]$
44: **for** $i = 1$ **to** $m_{1/2}$ **do**
45: **for** $j = 1$ **to** $m_{1/2}$ **do**
46: $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_m[i,j] = M_{m,1}[i,j] + M_{m,4}[i,j] - M_{m,5}[i,j] + M_{m,7}[i,j]\}$
47: **for** $j = 1$ **to** $m - m_{1/2}$ **do**
48: $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_m[i, j + m_{1/2}] = M_{m,3}[i,j] + M_{m,5}[i,j]\}$
49: **for** $i = 1$ **to** $m - m_{1/2}$ **do**
50: **for** $j = 1$ **to** $m_{1/2}$ **do**
51: $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_m[i + m_{1/2}, j] = M_{m,2}[i,j] + M_{m,4}[i,j]\}$
52: **for** $j = 1$ **to** $m - m_{1/2}$ **do**
53: $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_m[i+m_{1/2}, j+m_{1/2}] = M_{m,1}[i,j] - M_{m,4}[i,j] + M_{m,3}[i,j] + M_{m,6}[i,j]\}$
54:
55: $V \leftarrow \bigcup_{i=1}^{7}(V_i \cup A_{m,i} \cup B_{m,i}) \cup A_m^\star \cup B_m^\star \cup C_m$
56: **return** $(\mathcal{C}, V, C_m)$

Références

Côté, M.-C., B. Gendron, C.-G. Quimper et L.-M. Rousseau. 2011, «Formal languages for integer programming modeling of shift scheduling problems», *Constraints Journal*, p. 54–76.

Ebert, F. 2007, «CFG parsing and boolean matrix multiplication», cahier de recherche.

Lee, L. 2002, «Fast context-free parsing requires fast boolean matrix multiplication», *Journal of the ACM*, vol. 49, n° 1, p. 1–15.

Pesant, G., C.-G. Quimper, L.-M. Rousseau et M. Sellmann. 2009, «The polytope of context-free grammar constraints», dans *Proceedings of the 6th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR 09)*, p. 223–232.

Quimper, C.-G. et T. Walsh. 2006, «Global grammar constraints», cahier de recherche, COMIC-2006-005. Short version appeared at CP-06.

Quimper, C.-G. et T. Walsh. 2007, «Decomposing global grammar constraints», dans *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP 07)*, p. 590–604.

Valiant, L. 1974, «General context-free recognition in less than cubic time», *Journal of Computer and System Sciences*, vol. 10, n° 2, p. 308–315.