Deux types de problèmes

En informatique, il y a deux types de problèmes :

- Problèmes de calcul
  - On a des entrées $(x_1,x_2,\dots,x_n)$
  - On veut un résultat $y=f(x_1,x_2,\dots,x_n)$
  - On cherche un algorithme qui **calcule** la fonction f
- Problèmes de décision
  - On a des entrées $(x_1,x_2,\dots,x_n)$, y
  - On veut décider si $y\in f(x_1,x_2,\dots,x_n)$
  - On cherche un algorithme qui **décide** si $y\in f(x_1,x_2,\dots,x_n)$

Exemples

Problème de calcul : la somme

- **Données** : x et y deux entiers
- **Résultat** : z=x+y
- Le résultat est la valeur z, somme de x et y

Problème de décision : la primalité

- **Donnée** : n un entier
- **Question** : n est-il un nombre premier?
- La réponse à la question est oui/non

Pour nous

- On va s'intéresser à un problème de décision de théorie des langages utile en compilation,
- Le problème de l'appartenance :
  - **Données** :
    - G=(N,T,S,R) une grammaire algébrique
    - $m\in T^*$ un mot
  - **Question** :
    - Est-ce que $m\in L(G)$?

Résolution

Plusieurs manières pour résoudre le problème:

- au moyen des formes normales
- A l'aide de la transformation grammaire vers AP.

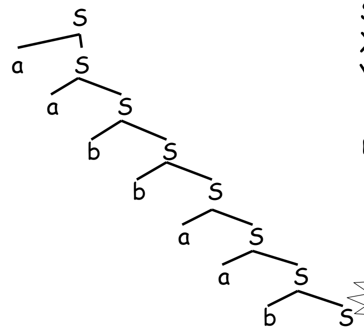
Avec les grammaires

- On met G sous FNG
- Toutes les règles sont de la forme
  $X\to a\gamma$ pour $\gamma\in(N\cup T)^*$
- Complexité de l'algorithme de décision:
  - Soit k le nombre maximal de règles associées aux variables
  - Chaque règle permet d'ajouter un terminal
  - Le mot est de longueur $|m|$
  - La complexité temporelle de cet algorithme est donc au plus $k^{|m|}$

Exemple

- m=aabbaab $\in$ L(G) pour G sous FNG de règles
  - $S\to aS|bS|bX$; $X\to aX|aY|a$; $Y\to a|b|aY|bY$
- On cherche les dérivations gauches qui permettent d 'engendrer m.

Exemple

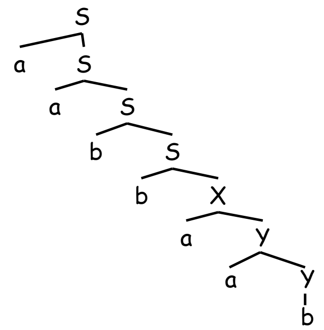


$S \to aS|bS|bX;$
$X \to aX|aY|a;$
$Y \to a|b|aY|bY$

m=aabbaab

raté

Exemple



$S \to aS|bS|bX;$
$X \to aX|aY|a;$
$Y \to a|b|aY|bY$

m=aabbaab

gagné

Conclusion

Il vaut mieux chercher un autre algorithme

Celui-ci est beaucoup trop lent!!!!

Avec les grammaires

- On met G sous FNC
- Toutes les règles sont de la forme
  $X \to AB$ ou $X \to a$ pour $X,A,B \in N$ et $a \in T$
- Complexité de l'algorithme de décision?
  - Soit k le nombre maximal de règles associées aux variables; Le mot est de longueur |m|
  - Dans le pire des cas, on a un arbre binaire à |m| feuilles et |m|-1 nœuds internes et k choix possibles par nœud
  - Le temps de cet algorithme est donc au plus $k^{|m|}$
- analogue au cas précédent; envisager une autre solution

Dernière tentative : automates à pile

On part de la grammaire G
- On construit l'AP correspondant
- On donne en entrée à l'AP le mot m
  - Si AP accepte m, $m \in L(G)$
  - Sinon, $m \notin L(G)$
- Complexité :
  - Comme on ne sait pas déterminiser les AP, la simulation déterministe d'un AP ND est a priori exponentielle.
  - Il faut envisager tous les arbres de calcul et en trouver un pour lequel la lecture a réussi.
- Il faut donc une autre méthode

Méthode Cocke Younger et Kasami (1965)

- Utilise :
  - Une grammaire G sous FNC
  - La **programmation dynamique**
- Combine les avantages des
  - Algorithmes **gloutons** qui effectuent le meilleur choix localement
  - Algorithmes de **recherche exhaustive** qui essayent toutes les possibilités et choisissent la meilleure
- Clairement, les solutions précédentes sont des algorithmes de recherche exhaustifs

Algorithme CYK

- **Notation** : $x_{i,j}$ facteur de x contenant les lettres $x(i)x(i+1)…x(i+j-1)$ i.e. le facteur de lonueur j qui commence en position i
- **Exemple :** Pour x=abracadabra, on a $x_{3,3}$=abracadabra=rac
- **Principe :** On calcule l'ensemble des variables $V_{i,j}$

$$V_{i,j} =\{A : A\in N : A\rightarrow^* x_{i,j}\}$$

et ceci pour tout i et pour tout j
- Le problème de l'appartenance se formule :

$$x\in L(G) \Leftrightarrow S\in V_{1,|x|}$$

Algorithme CYK

Pour i:=1 **à** n **faire**
  $V_{i,1}$:={A | A∈N, A→x(i) ∈R}
Pour j:=2 **à** n **faire**
  Pour i:=1 **à** n-j+1 **faire**
    $V_{i,j}$:=∅
    Pour k:=1 **à** j-1 **faire**
      $V_{i,j}$:= $V_{i,j}$∪{A | A →BC ∈R, B∈$V_{i,k}$, C∈$V_{i+k,j-k}$}

Exemple

j \ i	a	a	b	b	a	b
1	A	A	B,C	B,C	A	B,C

S→AB|BB
A →CC|AB|a
B →BB|CA|b
C →BA|AA|b

Exemple

j \ i	a	a	b	b	a	b
1	A	A	B,C	B,C	A	B,C
2	C	S,A	S,A,B	B,C	S,A	

S→AB|BB
A →CC|AB|a
B →BB|CA|b
C →BA|AA|b

Exemple

j \ i	a	a	b	b	a	b
1	A	A	B,C	B,C	A	B,C
2	C	S,A	S,A,B	B,C	S,A	
3	A,C	S,A,C	S,A,B,C	S,A,B,C		

S→AB|BB
A →CC|AB|a
B →BB|CA|b
C →BA|AA|b

Exemple

j \ i	a	a	b	b	a	b
1	A	A	B,C	B,C	A	B,C
2	C	S,A	S,A,B	B,C	S,A	
3	A,C	S,A,C	S,A,B,C	S,A,B,C		
4	S,A,B,C	S,A,B,C	S,A,B,C			

S→AB|BB
A →CC|AB|a
B →BB|CA|b
C →BA|AA|b

Exemple

a	a	b	b	a	b
A	A	B,C	B,C	A	B,C
C	S,A	S,A,B	B,C	S,A	
A,C	S,A,C	S,A,B,C	S,A,B,C		
S,A,B,C	S,A,B,C	S,A,B,C			
S,A,B,C	S,A,B,C				

i, j, 5

$S \to AB|BB$
$A \to CC|AB|a$
$B \to BB|CA|b$
$C \to BA|AA|b$

Exemple

a	a	b	b	a	b
A	A	B,C	B,C	A	B,C
C	S,A	S,A,B	B,C	S,A	
A,C	S,A,C	S,A,B,C	S,A,B,C		
S,A,B,C	S,A,B,C	S,A,B,C			
S,A,B,C	S,A,B,C				
S,A,B,C					

i, j

$S \to AB|BB$
$A \to CC|AB|a$
$B \to BB|CA|b$
$C \to BA|AA|b$

Complexité

n: **Pour** i:=1 **à** n **faire**
  $V_{i,1}$:={A:A∈N, A→x(i) ∈R} — Temps constant

Pour j:=2 **à** n **faire**
  $\sum_{i=1}^{n-j+1}\left(\sum_{k=1}^{j-1} c\right)$: **Pour** i:=1 **à** n-j+1 **faire**
    $V_{i,j}$:=∅
    $\sum_{k=1}^{j-1} c$: **Pour** k:=1 **à** j-1 **faire**
      $V_{i,j}$:= $V_{i,j}$∪{A:A →BC ∈R, B ∈ $V_{i,k}$, C ∈$V_{i+k,j-k}$} — Temps constant c

$$\sum_{j=2}^{n}\left(\sum_{i=1}^{n-j+1}\left(\sum_{k=1}^{j-1} c\right)\right)$$

$$n+\sum_{j=2}^{n}\left(\sum_{i=1}^{n-j+1}\left(\sum_{k=1}^{j-1} c\right)\right)$$

Complexité

$$n+\sum_{j=2}^{n}\left(\sum_{i=1}^{n-j+1}\left(\sum_{k=1}^{j-1} c\right)\right) = n + c\sum_{j=2}^{n}\left(\sum_{i=1}^{n-j+1}(j-1)\right) =$$
$$n + c\sum_{j=2}^{n}((n-j+1)(j-1)) = n + c\sum_{j=2}^{n}(-j^2 + j(n+2) - (n+1)) =$$
$$n - c\left(\frac{n(n+1)(2n+1)}{6} - 1\right) + c(n+2)\left(\frac{n(n+1)}{2} - 1\right) - c(n+1)(n-1) =$$
$$O(n^3)$$

Explication

- Si on a la règle
  - A→BC avec
    - B ∈ $V_{i,k}$
    - C ∈ $V_{i+k,j-k}$
  - B→* $x_{i,k}$ et C→* $x_{i+k,j-k}$
  - Donc, A →* $x_{i,j}$ et A ∈ $V_{i,j}$
- Et vice-versa

La programmation dynamique

- L'apport de la programmation dynamique est dans la construction de la « pyramide »
- Celle-ci aurait tout aussi bien pu être remplacée par des appels récursifs
- Dans ce cas, on retombe sur les idées du début, car cela revient de faire un grand nombre de fois le même appel.
- Par contre, une implémentation avec les appels récursifs qu'on fait uniquement une fois, en gardant le résultat est équivalent à CYK.