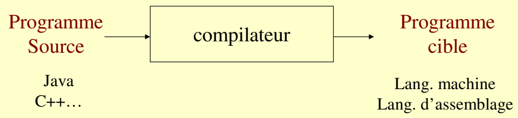
Théorie des langages et Compilation

claire.lefevre@univ-angers.fr

Compilation

- Traduction d'un programme écrit dans un premier langage (langage source) en un programme équivalent écrit dans un autre langage (langage cible)



quelques tâches d'un compilateur

- Analyse lexicale : lecture du programme source pour regrouper les caractères en unités lexicales (« catégories de mots »)
- Analyse syntaxique (ou grammaticale) : on regroupe les unités lexicales (« mots ») en structures grammaticales (« phrases ») qui seront généralement représentées par un arbre
- Production de code intermédiaire : construction d'un programme écrit dans un langage de + bas niveau

=> En compilation, on étudie des méthodes, des techniques, des algorithmes efficaces pour réaliser ce type de tâches

Les langages : ils sont partout

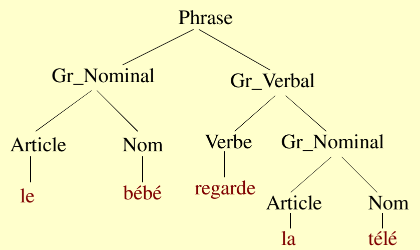
- Informatique :
  - Langages de programmation « évolués »
  - Langages machine
  - Protocoles de communication
  - Adresses IP, adresses web…
- « naturels » :
  - Français, anglais, chinois…
- Musique :
  - Do ré ré mi do do ré
- Génétique :
  - Codes ADN : ATCTACGTAAG

que fait-on avec ?

- Les décrire : caractériser les expressions « bien formées » d'un langage
  - INSTR : BLOC
    - | if EXPR_PAR INSTR [else INSTR]
    - | do INSTR while EXPR_PAR
    - …
  - Phrase → Gr_Nominal Gr_Verbal
    Gr_Nominal → Article Nom
    Gr_Nominal → Article Adjectif Nom
    Gr_Verbal → Verbe Gr_Nominal
    Gr_Verbal → Verbe
    …

- Vérifier qu'une expression est bien formée, et construire sa structure
  - Le bébé regarde la télé — bien formé
  - Bébé le télé regarde la — mal formé

  Structure exprimée par un arbre :



- « Transformer » une expression
  - Traduire dans un autre langage (compilation par exemple)
  - « calculer » quelque chose (calculatrice, …)
- Rechercher des motifs
  - egrep, awk, lex…

```
$ cat fich
If (x <= y){
    y = x;
}
else {
    x = y-x;
}
$ egrep -n '= y' fich
1:  If (x <= y){
5:    x = y-x;
$
```

7

Théorie des langages
Quoi ? Pourquoi ?

- Étude de « machines » (outils de calcul) abstraites
- Pour décrire, analyser, travailler efficacement sur les langages
- Pour modéliser la notion de calcul fini afin d'étudier
  - quels problèmes on est capable de résoudre
  - avec quelle efficacité

8

Historique

- Années 1930 : A. Turing => existence de pbs pour lesquels il n'existe pas de calcul fini pouvant fournir un résultat dans tous les cas
- Années 1940-50 : automates finis
- Fin années 1950 : N. Chomsky, grammaires formelles
- Fin années 1960 : S. Cook étend les machines de Turing => étude de la « complexité » de pbs

9

Deux grands types d'utilisation

- Les automates et les grammaires sont utilisés pour la conception et le développement de logiciels (en particulier en compilation)
- Les machines de Turing nous aident à comprendre quels problèmes on est capable de résoudre en temps fini et, parmi ceux-ci, quels problèmes ne sont pas résolubles en temps « raisonnable »

10

Langages
Concepts de base

11

Alphabets et chaînes

- Un alphabet Σ est un ensemble fini, non vide, de symboles
  - Ex : $\Sigma = \{a, b, c\}$
- Un mot ou une chaîne ω formé(e) sur un alphabet est une suite finie $s_1 s_2 \ldots s_n$ de symboles de cet alphabet
  - Ex : $\omega = abaa$
- La chaîne vide, notée ε, est une chaîne ne contenant aucun symbole
- La longueur d'une chaîne ω, notée $|\omega|$, est le nombre de symboles composant la chaîne ω
  - $|abaa| = 4$ $|\varepsilon| = 0$

12

Opérations sur les chaînes

- La concaténation de 2 chaînes u et v, notée u.v ou uv, est la chaîne obtenue en juxtaposant u et v

  si $u = a_1a_2 \ldots a_n$ *et* $v = b_1b_2 \ldots b_p$

  alors $uv = a_1a_2 \ldots a_nb_1b_2 \ldots b_p$

- Puissances d'une chaîne ω
  - ω^k est la chaîne formée par la concaténation de k occurrences de ω

    $$\omega^k = \underbrace{\omega\ \omega\ \omega\ \ldots\ \omega\ \omega}_{\text{k fois}}$$

  - $\omega^0 = \varepsilon$

- Un préfixe d'une chaîne ω est une suite, éventuellement vide, de symboles débutant ω
- Un suffixe de ω est une suite de symboles terminant ω
- Une sous-chaîne (ou facteur) d'une chaîne ω est une suite de symboles apparaissant consécutivement dans ω

  si $\omega = x.u.y$

  alors x est un préfixe de ω
  y est un suffixe de ω
  u est une sous-chaîne

- Notation : $|\omega|_x$ est le nombre d'occurrences de la chaîne x dans la chaîne ω

Langages

- Un langage est un ensemble de chaînes sur un alphabet Σ
- Le langage vide, noté $\varnothing$, ne contient aucune chaîne
- Attention : $\varnothing \neq \{\varepsilon\}$
- Le langage « plein », noté Σ^*, contient toutes les chaînes que l'on peut former sur l'alphabet Σ
- Σ^+ contient toutes les chaînes *non vides* sur Σ

Opérations sur les langages

- L'union de A et B est composée de toutes les chaînes qui apparaissent dans l'un au moins des langages A ou B : $A \cup B = \{\omega \mid \omega \in A \text{ ou } \omega \in B\}$
- L'intersection de A et B est composée des chaînes apparaissant à la fois dans A et dans B :

  $$A \cap B = \{\omega \mid \omega \in A \text{ et } \omega \in B\}$$
- La différence de A et B est le langage composé des chaînes de A n'apparaissant pas dans B :

  $$A \setminus B = \{\omega \mid \omega \in A \text{ et } \omega \notin B\}$$
- Le complémentaire de A sur un alphabet Σ comprend toutes les chaînes de Σ^* n'apparaissant pas dans A :

  $$\overline{A} = \Sigma^* \setminus A$$

- La concaténation de 2 langages A et B est le langage, noté A.B ou AB, composé de toutes les chaînes formées par une chaîne de A concaténée à une chaîne de B :

  $$A.B = \{u.v \mid u \in A, v \in B\}$$

  - Ex : A = {ab, a} B = {bc, b}
    A.B = {abbc, abb, abc, ab}
- Puissances d'un langage A :

  A^k est le langage formé par la concaténation de k occurrences de A
  - $A^0 = \{\varepsilon\}$
  - $A^{n+1} = A^n.A$ (ou $A.A^n$)

  A^k : « *mots formés par la concaténation de k mots de A* »

- Étoile de Kleene (fermeture ou clôture par .)
  - La fermeture de Kleene d'un langage A est le langage, noté A^*, défini par : $A^* = \bigcup_{n\geq 0} A^n = A^0 \cup A^1 \cup A^2 \cup A^3 \cup \ldots$

    « *mots formés par la concaténation d'un nbre qcq de mots de A* »
  - La fermeture positive de A est le langage, noté A^+, défini par : $A^+ = \bigcup_{n\geq 1} A^n = A^1 \cup A^2 \cup A^3 \cup \ldots$

    « *mots formés par la concaténation de 1 ou plusieurs mots de A* »

Propriété : $A^+ = A.A^* = A^*.A$
$A^* = A^+ \cup \{\varepsilon\}$

Langage ou problème ?

- Un problème de décision : auquel on répond par *oui* / *non*
  Ex 1 : soit x un nombre décimal, décider s'il est premier
  Ex 2 : soit un prg écrit en C, est-il syntaxiquemt correct ?

- On peut modéliser ces problèmes par la notion de langage
  L = ensemble de données pour lesquelles la réponse est *oui*
  Ex 1 : Lp = ens. des nbres premiers (notation décimale)
  Ex 2 : Lc = ens. des prgs C syntaxiquement corrects

Ce que la théorie des langages permet (parfois) :

- Déterminer si une chaîne donnée appartient à un langage
=> reconnaissance d'un langage
- Définir exactement quelles chaînes constituent un langage
=> spécification d'un langage

- Différents modèles permettent de formaliser ces pb
- Ces modèles n'ont pas tous la même « puissance »
- On classifie les langages selon le type de modèles qui permettent de les formaliser

Classification de Chomsky

Classes de langages	**Types de machines**	**Types de grammaires**
Réguliers	Automates finis	Type 3 : régulières
Non contextuels	Automates à pile	Type 2 : non contextuelles
Contextuels		Type 1 : contextuelles
Récursivement énumérables	Machines de Turing	Type 0 : sans restriction

Objectifs du cours

- Théorique : étude des langages réguliers et des langages non contextuels

- Applications à la compilation :
  - Les automates finis et les expressions régulières sont à la base des outils utilisés pour l'analyse lexicale
  - Les grammaires non contextuelles sont à la base des outils utilisés pour l'analyse syntaxique

- Notions de calculabilité et de décidabilité

Expressions régulières
et
Automates finis

2 modèles pour les langages réguliers

- Expressions régulières (ER) : manière « algébrique » de décrire un langage régulier
  => notation simple et précise

- Automates finis (AF) : manière quasi opérationnelle de décrire un langage régulier
  => peut facilement être implémenté

Dans de nombreux systèmes de recherche de motifs
- les ER servent de langage d'interface avec l'utilisateur
- les AF sont à la base de l'implémentation

Expressions régulières (ER) et langages

- Les ER sur un alphabet Σ et les langages correspondants sont définis récursivement par :

base :

- $\emptyset$ est une ER qui représente le langage $\emptyset$
- ε est une ER qui représente le langage $\{\varepsilon\}$
- a est une ER (pour tout $a \in \Sigma$) qui représente $\{a\}$

récur : si r et s sont des ER qui représentent les langages R et S, alors

- r | s (ou r + s) qui représente le langage $R \cup S$
- r.s (ou rs) qui représente le langage R.S
- r^* qui représente le langage R^*

sont des ER

Exercice

- Quel est le langage représenté par :
  - a*b*
  - aa*b*b
  - (a*b*)*aa(a|b)*
- Donner une expression régulière pour :
  - les chaînes sur {a,b,c} comprenant au moins un a et au moins un b
  - les chaînes sur {a,b,c} qui contiennent au plus une fois deux a consécutifs

- Notation : on note r^+ pour $r.r^*$ (ou $r^*.r$)
- Priorité des opérateurs (par ordre décroissant) :

  * puis . puis |
- Deux ER r et s sont équivalentes, noté $r \equiv s$ ou $r = s$, si elles représentent le même langage

Rem : de nombreuses ER peuvent représenter le même langage

Ex :

- $\emptyset^* = \varepsilon$
- $a^* = a^+ \mid \varepsilon$
- $(a \mid \varepsilon)^* = a^*$
- $(a \mid b)^* = (a^*b^*)^*$

Automates finis

- Un modèle pour la reconnaissance d'un langage
- Automate fini pour un langage L : « machine » qui permet de répondre à la question $\omega \in L$?

Automates finis déterministes (AFD)

- Un AFD est un quintuplet $(\Sigma, Q, q_0, F, \delta)$ où :
  - Σ est un alphabet fini
  - Q est un ensemble fini, non vide, d'états
  - $q_0 \in Q$ est l'état de départ (initial)
  - $F \subseteq Q$ est l'ensemble des états acceptants (ou finals)
  - δ est une fonction de transition de $Q \times \Sigma$ dans Q
- $\delta(p, a) = q$ est parfois noté $p \xrightarrow{a} q$ et signifie :

  « *si l'on est en l'état p et que l'on lit le symbole a alors on passe en l'état q* »

Chaînes et langage acceptés par un AFD

Soit un AFD $M = (\Sigma, Q, q_0, F, \delta)$

- M accepte une chaîne $\omega = a_1a_2\ldots a_n$ si il y a un chemin dans le diagramme de transitions qui
  - débute en l'état initial q_0
  - est étiqueté par $a_1a_2\ldots a_n$

  et se termine en un état acceptant de F
- Le langage accepté par M est

  $L(M) = \{\omega \mid \omega \text{ est accepté par } M\}$

Automate complet

- Un AFD est dit complet si la fonction de transition δ est totale : $\delta(p,a)$ est partout définie (pour tous p et tous a)
- Un état puits P est un état
  qui n'est **pas** acceptant
  et tel que toutes les transitions issues de P mènent à P
- On peut toujours compléter un AFD pour le rendre complet :
  - On ajoute un état puits P
  - On fait en sorte que toutes les transitions non définies mènent à P
- Cela ne change pas le langage accepté par l'AFD

31

Formellement

- On définit une fonction Δ qui étend la fonction de transition δ aux chaînes
- $\Delta(q, \omega) = q'$ signifie :
  « *si l'on est en l'état q et que l'on lit la chaîne ω alors on arrive en l'état q'* »
- Définition par récurrence sur la longueur de ω

  $\Delta : Q \times \Sigma^* \rightarrow Q$

  base : $\Delta(q, \varepsilon) = q$

  $\Delta(q, a) = \delta(q, a)$ si $a \in \Sigma$

  récur : $\Delta(q, \omega a) = \delta(\Delta(q, \omega), a)$ si $\omega \in \Sigma^*$ et $a \in \Sigma$

32

Soit un AFD $M = (\Sigma, Q, q_0, F, \delta)$, on a alors :

- Une chaîne ω est acceptée par M ssi $\Delta(q_0, \omega) \in F$
  « *en partant de l'état initial q_0 et en lisant ω, on arrive à un état acceptant* »
- Le langage accepté par M est $L(M) = \{\omega \mid \Delta(q_0, \omega) \in F\}$
  « *l'ensemble de toutes les chaînes qui, partant de q_0, mènent à un état acceptant* »
- Les langages pour lesquels il existe un AFD sont appelés les langages réguliers

33

Exemple 1

Un homme (H), un loup (L), une bique (B) et un chou (C) sont sur la rive gauche d'une rivière.

Il y a une barque pouvant transporter l'homme et *l'un seulement* des 3 autres.

- But : faire traverser la rivière à tout le monde
- Contraintes :
  - si le loup et la bique sont ensemble sans surveillance, le loup mange la bique
  - De même pour la bique et le chou

Est-il possible de faire traverser la rivière à tout le monde sans perte ? Si oui, comment ?

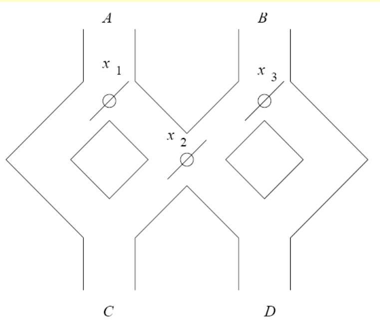
Modéliser à l'aide d'un AFD les situations et transitions possibles

⇒ États : situations (qui est sur chaque rive)

⇒ Transitions : passages possibles d'une situation à une autre

35

Exemple 2 : jeu de bille



36

Équivalence entre AF et ER

On sait que les AF permettent de modéliser les langages réguliers

Qu'en est-il de l'expressivité des ER ?

- Il existe des méthodes pour transformer toute ER en AF (admis, cf cours L1)
  => les ER ne sont pas plus puissantes que les AF
- Mais sont-elles aussi puissantes que les AF ?
  => Oui, car on peut montrer que, pour tout AF, il existe une ER qui représente le même langage

=> les ER et les AF ont exactement la même capacité de représentation

ils reconnaissent les mêmes langages :

les langages réguliers

37

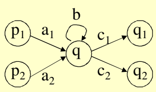
Construction d'une ER correspondant à un AFD par élimination d'états

Étant donné un AFD,

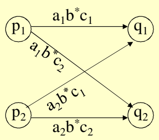
on va éliminer un à un les états q qui ne sont pas l'état initial ni un état final

en contrepartie, on ajoute des arcs entre prédécesseurs et successeurs de q, étiquetés par des ER

Exemple

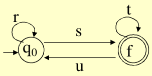


devient



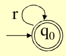
méthode

1. Pour chaque état final f,
   éliminer tous les états sauf q_0 et f
2. Si $q_0 \neq$ f alors on obtient :



et l'ER correspondante est (p.ex.) : r*s (t | ur*s)*

Si q_0 = f alors on obtient :



et l'ER correspondante est : r*

3. L'ensemble des chaînes acceptées par l'AF est l'union des chaînes acceptées par chaque état final

Lemme de l'étoile

(ou lemme de la pompe)

- On a vu 2 caractérisations des langages réguliers
  - ER
  - AFD

⇒ Pour montrer qu'un langage est régulier,
il « suffit » de trouver une ER ou un AF qui le décrit

⇒ Pour montrer qu'un langage n'est pas régulier,
il faudrait être sûr qu'il n'existe pas d'ER ni d'AF

On va voir, via le lemme de l'étoile, une façon de faire

Lemme de l'étoile (pumping lemma)

But : montrer qu'un langage L n'est pas régulier

Exemple :

Supposons que $L = \{a^n b^n \mid n \geq 0\}$ est régulier

- Alors il existe un AFD à N états qui reconnaît L
- Si on lit a^N, alors on passe au moins 2 fois par le même état de l'automate

  Autrement dit, il y a une boucle de lg k (p. ex) sur les a
- Donc, si $a^N b^N$ est accepté par l'AFD, $a^{N+k} b^N$ l'est aussi, pourtant, il n'appartient pas à L

Donc L'AFD ne reconnaît pas L, et L n'est pas régulier

Lemme de l'étoile (informel)

Si L est un langage régulier, alors :
- Il existe un AFD à N états qui reconnaît L
- Et tous les mots de L de longueur ≥ N sont tels que
  - il existe une boucle dans les N premiers symboles
  - et donc, si on passe 0 ou plusieurs fois dans la boucle, le mot est toujours accepté par l'AFD (et donc ∈ L)

- Utilisation :
  - pour montrer qu'un langage n'est pas régulier
  - en utilisant une démarche par l'absurde (voir ex. précédent)