Analyse Lexicale

Compilation

Chebieb

2.01

Décembre 2020

Table des matières

Objectifs

- Définir l'activité "analyse lexicale" dans le processus de compilation
- Énoncer les concepts de bases de l'analyse lexicale
- Expliquer le fonctionnement d'un analyseur lexical
- Appliquer les algorithmes nécessaires pour la création des analyseurs lexicaux
- Mettre en œuvre un analyseur lexical avec un générateur automatique

Introduction

Dans ce chapitre nous allons aborder la notion d'analyse lexicale et comment fabriquer un analyseur lexical :

- Le déroulement de l'analyse lexicale se base sur un ***a***utomate d'état ***f***ini (***AF***) qui permet de reconnaître ou rejeter les différents mots du langage source.
- La fabrication d'un analyseur lexical est une implémentation d'un ***a***utomate d'état ***f***ini ***d***éterministe (***AFD***).
- Cet AFD est construit d'une façon automatique à partir de la spécification de la forme (motif ou *pattern*) des mots du langage source sous forme d'un langage régulier.

Les principales notions abordée seront :

1. Définition et rôle de l'analyse lexicale
2. Notions de base pour l'analyse lexicale
3. Les outils pour le développement de l'analyseur lexical

I Définition et rôle de l'analyse lexicale

1. Comment se définissent les langages de programmation

C'est quoi un programme ?

Un texte regroupant une suite d'instructions avec des données combinées pour commander un processeur (une machine physique ou virtuelle dite aussi automate)

C'est quoi un langage de programmation?

Le programme possède une structure obéissant à des règles

Les règles fixent l'ensemble des signes et symboles à utiliser et la manière avec laquelle les utiliser

Ces règles définissent un langage de programmation

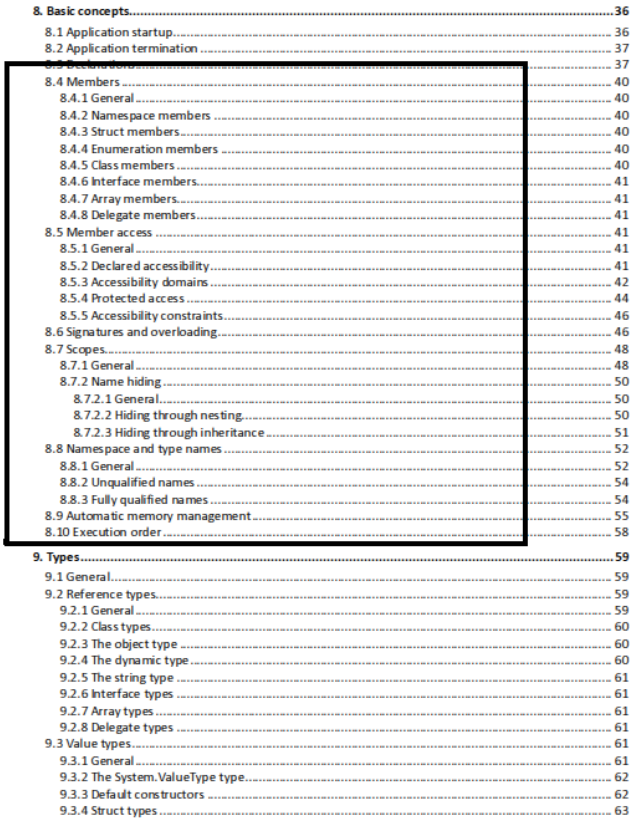
La description des langages de programmation

Les langages de programmations possèdent une description sur trois niveaux :

1. Le lexique : un ensemble de mots et de symboles à utiliser dite ***unités lexicales***
2. La syntaxe : un ensemble de règles de combinaison des mots et des symboles pour former les commandes ou ***les instructions***
3. La sémantique : les ***actions opérationnelles*** lancées par le processeur suite aux combinaisons former par l'auteur du programme

🔎 *Exemple : Description standardisée du langage MS C#*

C# est décrit par une spécification ECMA. Ci après quelques pages du sommaire de la spécification qui concerne le niveau lexical et syntaxique ainsi que sémantique



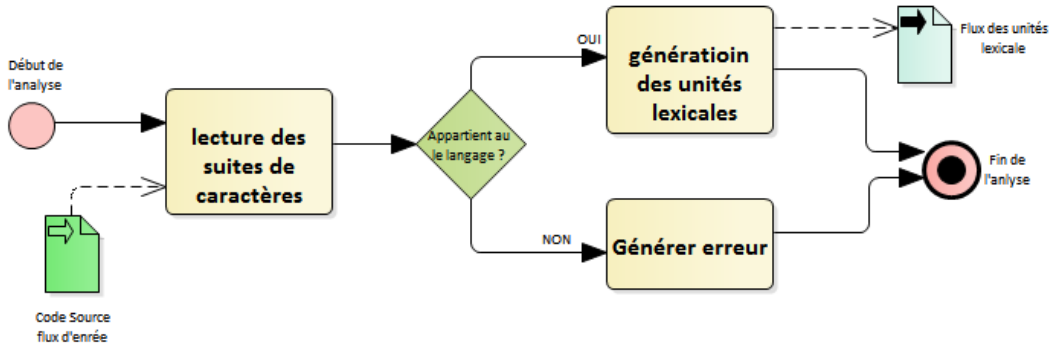
Fondamental : Langage et compilateur

A chaque langage ainsi défi sont associée un ou plusieurs compilateur pour pouvoir traduire les programmes vers des instructions du processeurs (machine) qui doit les exécuter

2. Définition de l'analyse lexicale

L'analyse lexicale est l'étape frontale de la chaîne de compilation. Elle consiste à vérifier le flux représentant le code source pour :

- Identifier les mots du langage utilisés dans le code source et les codifier dans un flux de sortie. Ces mots sont les ***UNITÉS LEXICALES*** (*TOKENS*) du code source.
- Éliminer les mots inutiles ou ***superflus*** qui ne sont pas destinés à être traduit dans le langage cible de la compilation : les blancs, les tabulations, les sauts de ligne et les commentaires .....
- Signaler les mots qui ne font pas partie du langage : ***erreurs lexicales***



Analyse lexicale

II Notions de base pour l'analyse lexicale

1. Concepts de base pour l'analyse lexicale

1.1. Rappel et définitions

Rappel : L'alphabet

Ensemble de symboles utilisés pour former des combinaisons servant dans une notation donnée dans un domaine donné. Formellement un alphabet est un ensemble fini non-vide de lettres ou symboles. Il est généralement noté ***Σ.***

Par exemple ***Σ = {0, 1}*** représente un alphabet pour une notation représentant les nombres binaires !

Rappel : Le mot

Nommé aussi "chaîne". C'est une succession de symboles formée en combinant les symboles d'un alphabet. Le mot est généralement noté ***ω***. Le nombre de symboles formant un mot ***ω*** est la longueur de ce mot notée ***|ω|***.

Si cette longueur est nul on parle alors du mot vide noté "***ε***" qui ne contient aucun symbole ***|ε|*** = 0

Méthode : La concaténation

C'est une loi de composition interne sur un alphabet représentant l'opération qui crée un mot sur l'alphabet formé des caractères du premier mot suivis de ceux du second. Elle est noté ***.***

La longueur du mot formé par concaténation est la somme des longueurs des de mots concaténés.

Exemple

Soit un alphabet Σ = {a, b, c} et soit deux mots sur cette alphabet tq ω_1=a et ω_2 = b.

La concaténation de ω_1 et ω_2 sera le mot ω_3 formé sur l'alphabet Σ tq

$\omega_3 = \omega_1 . \omega_2 = a.b$

Fondamental

Si ω est un mot sur l'alphabet Σ alors :

ε . ω = ω . ε = ω

ω^n signifie la concaténation n fois le mot ω. Si n=2 alors ω^2 = ω.ω et si n= 0 alors $\omega^0 = \varepsilon$

Σ* est le monoïde libre en/sur Σ : tout les mots construits sur l'alphabet Σ par concaténation

Exemple

Soit un alphabet $\Sigma = \{a\}$ le monoïde libre sur Σ est

$\Sigma^* = \bigcup_{n=0}^{\infty}\{\omega = a^n\} = \{\varepsilon\} \cup \{a\} \cup \{aa\} \cup \ldots \cup \{a^i\} \cup \ldots \cup \ldots$

Définition : Le langage

On désigne par "langage" un ensemble de mots formés sur un alphabet donnée.

Formellement un langage noté L sur un alphabet Σ est un sous ensemble de Σ^* : $L \subset \Sigma^*$

Un langage peut être infini ($|L| = \infty$), fini ($\exists\ n \in N\ ; |L| = n$) ou vide ($L = \Phi$).

Méthode : Les opérations sur les langages

Soit deux langages L_1 et L_2 sur un alphabet donné Σ. L_1 est L_2 sont des sous ensembles de Σ^* alors les opérations suivantes sont possibles sur ces langages :

- La concaténation : $L_1.L_2 = \{\omega \in \Sigma^* / \omega = \omega_1\ .\ \omega_2 \wedge \omega_1 \in L_1 \wedge \omega_2 \in L_2\}$
- Union : $L_1 \cup L_2 = \{\omega \in \Sigma^* / \omega \in L_1 \vee \omega \in L_2\}$
- Intersection: $L_1 \cap L_2 = \{\omega \in \Sigma^* / \omega \in L_1 \wedge \omega \in L_2\}$
- Itération: ou ***fermeture*** d'un langage : $L^* = \bigcup_{n=0}^{\infty} L^n = \{\varepsilon\} \cup L^n / n \geq 1$
- Itération positive ou ***fermeture positive*** d'un langage : $L^+ = \bigcup_{n=1}^{\infty} L^n = L^n / n \geq 1 \quad (\varepsilon \notin L^+)$

Cette itération est appelée aussi la fermeture ou l'étoile de Kleene du nom de celui qui l'a introduit*

Définition : Le lexique et le lexème

Le lexique d'un langage est la manière de former les mots sur un alphabet donné.

Le lexème est la succession de caractères appartenant à une chaîne donnée qui peut former un mot d'un langage.

Exemple

Soit le langage L défini sur l'alphabet Σ tel que : $\Sigma = \{a, 0\}, \quad L = \{\omega \in \Sigma^* / \omega = a^n 0^m \wedge n > m\}$

- Le lexique de ce langage est cette forme $a^n 0^m$ qui signifie plus de "***a***" que de "***0"*** dans les mots de L
- La chaîne $\phi = aaa0000000$ n'est pas un mot du langage L mais il contient un lexème $\psi = aaa00$

1.2. Les langages réguliers et les expressions régulières

Rappel : Langages réguliers ou rationnels

Un langage ***régulier*** sur un alphabet est un sous ensemble d'un monoïde sur un alphabet, qui peut-être construit avec les opérations : union, itération et concaténation.

Rappel : Propriétés des langages réguliers

Un langage régulier L sur un alphabet Σ est défini récursivement :

- $\Phi, \{\varepsilon\}$ sont des langages réguliers ;
- $\forall a \in \Sigma, L = \{a\}$ est un langage régulier ;
- Si deux langages $L_1\ et\ L_2$ sont réguliers sur Σ alors :
  $L_1 \cup L_2$, $L_1.L_2\ et\ L^*$ sont aussi des langages réguliers sur Σ;

– Il n'y a pas d'autre langages réguliers sur Σ

Rappel : Langage régulier et grammaires

Dans la classification de Chomsky, ces langages sont engendrée par des grammaires régulières droites ou une grammaires régulières gauches (il sont de type 3)

Fondamental : Expression régulière et lexique d'un langage

Une expression régulière est une façon concise de décrire le lexique d'un langage régulier. C'est à dire la forme générale des mots d'un langage régulière. Elles ont été définies par Kleene* récursivement sur l'alphabet du langage qu'elles décrivent :

Soit un alphabet Σ et un langage L, une expression régulière (R) qui décrit L est R(L) :

- $R = \Phi$ est l'expression qui décrit le langage vide Φ ;
- $R = \varepsilon$ l'expression qui décrit le langage $\{\varepsilon\}$
- $\forall a \in \Sigma \ : \ R = a$ est une expression qui décrit le langage $\{a\}$
- Pour deux expressions régulières $R_1 \ et \ R_2$ décrivant deux langages $L_1 \ et \ L_2 \ sur \ \Sigma$:
  - $R_1 | R_2 \ correspond \ à \ L_1 \cup L_2$
  - $R_1 R_2 \ correspond \ à \ L_1 . L_2$
  - $R_1^* \ correspond \ à \ L_1^*$ (fermeture de Kleene).

Exemple

Soit l'alphabet $\Sigma = \{0, 1, a, b\}$. Les expressions suivantes peuvent être formées :

$0, 1, a, b$
$0a \,|\, 0b$
$a^* b^*$
a^+ qui simplifie : $a.a^*$

Attention : Priorités des opérations sur les expressions régulières

La priorité des opérations lors de la construction des expressions régulières est dans l'ordre décroissant suivant:

- l'itération $(*)$,
- la concaténation $(.)$
- l'union $(\cup)$.

Les ( ) peuvent être donc utilisée pour les sous expressions comme $(a|b)^* b^* = R^* b^* où \, R = a|b$

Rappel : Théorème de Kleene

Les langages réguliers sur un alphabet (décrit par des expressions régulières) sont reconnaissables par automates d'états finis (voir les détails dans :Kleen* Eilenberg*). En d'autres termes

- Soit un alphabet σ et un ensemble de langages réguliers R(L) (décrits en expressions régulières).
- Soit AF(L) ensemble de langages reconnues par des Automates d'états finis (AF).
  $R(L) = AF(L) \ sur \ \Sigma^*$

1.3. Les automates d'états fini

Définition : Automate d'état fini (formalisme de Moore) (AF)

Un automate d'état fini (théorème de Moore) est une structure $\langle Q, \Sigma, \gamma, i, F \rangle$:

Σ: un alphabet

Q : ensemble fini d'états

$\gamma : Q \times \Sigma \longrightarrow Q$:la fonction de transition

$i\ (i \in Q)$: l'état initial de l'automate

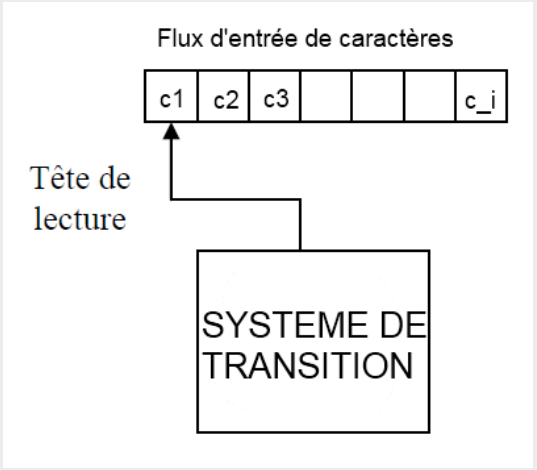
$F\ (F \subset Q)$: ensemble des états finaux

Fondamental : Reconnaissance d'un langage

Un langage L sur un alphabet Σ reconnu par un automate AF est l'ensemble $\{\omega \in \Sigma / \gamma\,(i, \omega) \in F\}$.

Cette ensemble est noté L(AF).

Un AF représente un machine à états ayant une tête de lecture qui avance sur un flux d'entrée caractère par caractères

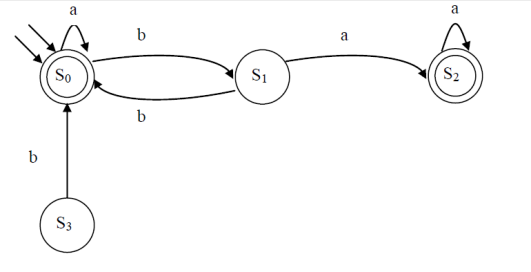


Automate d'États Finis

Définition : Automate d'états finis déterministe (AFD)

C'est un automate déterministe est caractérisé par :

- L'absence de transition vide dite \varepsilon-\text{transition} $\forall e \in Q\ \gamma\,(e, \varepsilon) \notin Q$
- Il ne peut exister qu'une transition au plus à partir d'un état ***e*** avec un symbole ***s*** $\forall \langle e, s \rangle \in Q \times \Sigma\ \ \exists\, T = \{\gamma\,(e,\ s) \in Q\}$ avec $|T| \leq 1$



$AFD = \langle S, \Sigma, \gamma, s_0, F \rangle$ avec
$\Sigma = \{a, b\}$
$S = \{s_0, s_1, s_2, s_3\}$
$F = \{s_0, s_2\}$
Représentation graphique de $\gamma : S \times \Sigma \longrightarrow S$

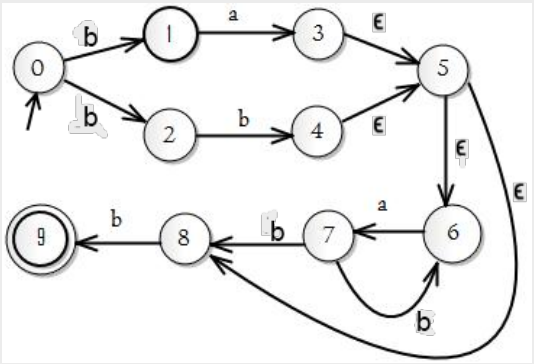
Définition : Automate d'états finis non déterministe (AFND ou AFN))

C'est un automate non déterministe est reconnu par :

- Présence de transition vide dite $\varepsilon - \text{transition} \exists\, e \in Q \;\exists\, \gamma\,(e, \varepsilon) \in Q$
- Présence possible de plus d'une transition d'un état ***e*** avec un symbole ***s*** $\forall \langle e, s \rangle \in Q \times \Sigma \;\; \exists\, T = \{\gamma(e,\ s) \in Q\}$ avec $|T| \geq 0$

La représentation graphique de :

$AFD = \langle S, \Sigma, \gamma, 0, F \rangle$ avec
$\Sigma = \{a, b, \varepsilon\}$
$S = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
$F = \{9\}$
$\gamma : S \times \Sigma \longrightarrow S$
$\exists \varepsilon - transition$
$\exists\, T = \{\gamma(0, b), \gamma(7, b) \in Q\}$ avec $|T| = 2$



Exemple AFN

Méthode : Définition et Implémentation d'un AFD

Les automates déterministes "***AFD***" sont :

- Non intuitifs à définir mais relativement simple à implémenter par programme. On aura besoin :
  - Table de transition TransTable : structure de donnée représentant l'AFN $\langle Q, \Sigma, \gamma, i, F \rangle$ précisément l'ensemble des états possibles de : $\gamma : Q \times \Sigma \longrightarrow Q$
    - Une fonction de transition $\text{Transist}\,(e, c)$: utilise la TransTable pour déterminer le prochain état à atteindre avec l'entrée ***c***
    - Une fonction de lecture du flux $\text{GetNextChar}\,()$ qui retourne le caractère lu sur le flux d'entrée
    - Si la $\text{Transist}\,(e, c)$ retrouve l'état final alors le flux est reconnu par l'automate ***"ACCEPTATION"***
    - Si elle ne retourne aucun état dans la table alors le flux est rejeté et la chaîne est ***non reconnue***

Méthode : Définition et Implémentation d'un AFN

Les automates non déterministes "***AFN***" sont :

- Plus intuitifs à définir mais pas simple à implémenter par programme. On aura besoin :
    - D'une Table de transition TransTable : structure de donnée représentant l'AFN $\langle Q, \Sigma, \gamma, i, F \rangle$ précisément l'ensemble des états possibles de : $\gamma : Q \times \Sigma \longrightarrow Q$
    - D'une méthode de transformation en AFD pour pouvoir les implémenter

Bibliographie

Janusz A. Brzozowski *Canonical regular expressions and minimal state graphs for definite events* Proceedings of the Symposium on Mathematical Theory of Automata, Polytechnic Institute of Brooklyn, New York, Wiley April 1962-1963

Samuel Eilenberg *Automata, Languages and Machines* Vol. A, Academic Press 1974

Victor M. Glushkov *The abstract theory of automata* Russian Mathematical Surveys, vol. 16 1961

John Hopcroft *An n log n algorithm for minimizing states in a finite automaton* Proc. Internat. Sympos., Technion, Haifa, New York, Academic Press 1971

Edward F. Moore *Gedanken-experiments on sequential machines in Automata studies* Princeton, N. J. Princeton University Press coll. Annals of mathematics studies 1956

Stephen Cole Kleene *Representation of events in nerve nets and finite automata* C.E. Shannon and J. McCarthy, Automata Studies, vol. 34, Princeton, N. J., Princeton University Press, coll. « Annals of Mathematics Studies 1956

Webographie

www.gnu.org/software/flex

JavaCC

www.cs.princeton.edu/~appel/modern/java/JLex

http://www.bumblebeesoftware.com

https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expressions

https://fr.wikipedia.org/wiki/Ken_Thompson