Université Joseph Fourier **Master Informatique**
UFR IM^{2}AG

Sémantique des Langages de Programmation et Compilation

Un langage est défini par :

1. sa lexicographie, c'est-à-dire les mots (ou **lexèmes)** qui le composent,
2. sa syntaxe, ou l'organisation des mots dans une phrase,
3. sa sémantique qui en informatique est scindée en deux parties :
   — les règles de bonne construction d'un texte au delà de la syntaxe, comme par exemple le fait que les variables utilisées doivent être déclarées. On parle de sémantique statique.
   — le sens des phrases c'est-à-dire par exemple la fonction qu'elles définissent On parle alors de sémantique dynamique.

Nous donnons ici quelques définitions et résultat sur la description de la syntaxe des langages.

1 Préliminaires

Nous donnons la définition d'un système de transition, notion qui servira à la fois pour la syntaxe et la sémantique d'un langage de programmation. Un système de transition est un couple : $S = (C, \longrightarrow)$ tel que C est l'ensemble des configurations et $\longrightarrow \subseteq C \times C$ la relation de transition. Parfois, on utilise une configuration *initiale* C_0 et un ensemble des configurations *terminales*, sous-ensembles de C. Une *exécution* est une séquence de configurations $C_0, \cdots, C_n$ telle-que $C_i \longrightarrow C_{i+1}, i = 0, \ldots, n-1$.

2 Grammaires et langages

2.1 Définitions et notations

Alphabet, mots

— Un *alphabet* est un ensemble fini d'éléments appelés symboles ou lettres,
— un *mot* est une suite finie de lettres,
— le mot vide (de longueur 0) est noté ϵ,
— un *langage* est un ensemble de mots,
— soit Σ un alphabet, le langage de tous les mots est noté Σ^*. En fait, $\Sigma^* = \bigcup_{i \in \mathbb{N}} \Sigma^i$ où Σ^i est le langage sur Σ des mots de longueur i.

2.2 Exemples

— Soit $\Sigma = \{0, 1\}$. On peut définir le langage des entiers codés en binaire :$\{0, 1, 10, 11, 100, \cdots\}$.
— Soit $\Sigma = \{a, b\}$. $\{a, \epsilon, bb\}$ est un langage fini sur $\{a, b\}$.

Definition 2.1 *Une grammaire est la donnée d'un quadruplet $G = (V_T, V_N, Z, P)$ où*
— *V_T est un ensemble fini qui dénote l'ensemble des symboles terminaux,*
— *V_N, est un ensemble fini qui dénote l'ensemble des symboles non-terminaux,*
— *$Z \in V_N$ l'axiome et*
— *P est l'ensemble des productions.*

On note V = $V_T \cup V_N$, le vocabulaire de la grammaire. Par ailleurs, V_T et V_N sont disjoints ($V_T \cap V_N = \emptyset$). P l'ensemble des règles de production ; une règle est un couple (x, y) de mots sur V (avec $x, y \in V^*$). On note généralement $x \to y$ au lieu de $(x, y) \in P$.

Dérivation Une grammaire permet de dériver un mot u en un mot v en une étape si il existe un règle $x \to y$ telle que

— $u = \alpha x \beta$,
— $v = \alpha y \beta$,

Une dérivation en plusieurs étapes, notée $\to^*$, est la fermeture transitive de la relation $\to$.

Finalement, on peut définir le langage engendré par une grammaire $G = (V_T, V_N, Z, P)$ de la manière suivante :

$$L(G) = \{w \in V_T^* | Z \to^* w\}$$

Exemples

— L'ensemble des entiers codés en binaire est décrit par la grammaire suivante $G1 = (V_T, V_N, Z, P)$ où :
  — $V_T = \{0, 1\}$.
  — $V_N = \{Z, A\}$
  — Le règles de production sont décrites par :

$$\begin{array}{lcl} Z & \to & 0 \\ Z & \to & 1A \\ A & \to & 1A \\ A & \to & 0A \\ A & \to & \epsilon \end{array}$$

  Par exemple, $10 \in L(G_1)$ car $Z \to 1A \to 10A \to 10\epsilon = 10$. Par contre, $01 \notin L(G_1)$.

— Un ensemble d'identificateurs pour un langage : $G2 = (V_T, V_N, I, P)$ où :
  — $V_T = \{0, \cdots 9, a \cdots z\}$.
  — $V_N = \{I, A\}$
  — Le règles de production sont décrites par :

$$\begin{array}{lcl} I & \to & aA \\ & \cdots & \\ I & \to & zA \\ A & \to & 0A \\ & \cdots & \\ A & \to & 9A \end{array}$$

— Une grammaire d'expressions arithmétiques (On donne uniquement les règles de production)

$$\begin{array}{lcl} E & \to & E + T \\ E & \to & T \\ T & \to & T * F \\ T & \to & F \\ F & \to & \texttt{IDF} \\ F & \to & \texttt{NUM} \\ F & \to & (E) \end{array}$$

2.3 Classification des grammaires (Chomsky)

Pour chaque type de grammaire, on donne la forme des règles.

Type 0, Grammaires générales : $x \to y$, avec $x \in V^* \cdot V_N \cdot V^*, y \in V^*$

Type 1, Grammaires contextuelles : $\alpha \cdot A \cdot \beta \to \alpha \cdot \gamma \cdot \beta$, avec $\alpha, \beta, \gamma \in V^*, \gamma \neq \epsilon, A \in V_N$

Type 2, Grammaires non-contextuelles : (langages algébriques) $A \to \gamma, A \in V_N, \gamma \in V^*$

Type 3, Grammaires régulières : (langages rationnels)
- linéaires gauches, $A \to B \cdot a | A \to a, A, B \in V_N, a \in V_T$
- linéaires droites, $A \to a \cdot B | A \to a, A, B \in V_N, a \in V_T$

Dans le cas des langages de programmation, les grammaires régulières sont utilisées pour le lexique (analyse lexicale), les grammaires non-contextuelles pour décrire les programmes corrects.

3 Langages réguliers

3.1 Définitions

Soit V un **alphabet**. L'ensemble des **langages réguliers** sur V peut-être défini récursivement par les règles suivantes :
- le langage vide $\emptyset$ est un langage régulier ;
- le langage $\{\varepsilon\}$ est un langage régulier ;
- pour tout x de V, $\{x\}$ est un langage régulier ;
- si $R1$ et $R2$ sont des langages réguliers alors : $R1 \cup R2$, $R1.R2$ et $R1^*$ sont des langages réguliers.

Notons que certains langages ne sont pas réguliers, comme par exemple le langage $\{a^n.b^n \mid n > 0\}$ défini sur le vocabulaire $\{a, b\}$. En particulier la plupart des langages de programmation ne sont pas des langages réguliers.

3.2 Expressions régulières

Tout langage régulier sur V peut être décrit par une **expression régulière**, c'est-à-dire un terme construit sur l'alphabet $V \cup \{+, ., *, \underline{\varepsilon}, \underline{\emptyset}\}$ de la manière suivante :
- $\underline{\emptyset}$ décrit le langage vide $\emptyset$
- $\underline{\varepsilon}$ décrit le langage $\{\varepsilon\}$
- pour tout x appartenant à V, x décrit le langage $\{x\}$
- si $r1$ et $r2$ sont des expressions régulières sur V décrivant respectivement les langages $R1$ et $R2$ alors : $r1 + r2$ décrit le langage $R1 \cup R2$, $r1.r2$ décrit le langage $R1.R2$, et $r1^*$ décrit le langage $R1^*$.

On considèrera dans la suite que l'opérateur unaire $*$ est plus prioritaire que l'opérateur ., lui-même plus prioritaire que l'opérateur $+$.

On donne ci-dessous quelques exemples d'expressions régulières définies sur le vocabulaire $\{a, b, c\}$, ainsi que les langages (réguliers) correspondants :

Expression régulière	Eléments du langage
$a + b.c$	$\{a, bc\}$
$(ab)^*$	$\{\varepsilon, ab, abab, ababab, \ldots\}$
$ab + c*$	$\{\varepsilon, ab, c, cc, ccc, \ldots\}$

3.3 Automate d'état fini

Un intérêt des langages réguliers est que, pour tout langage régulier R défini sur V, il existe un algorithme efficace pour décider si un mot quelconque de V^* appartient ou non à R. Cet algorithme repose sur une caractérisation de R à l'aide d'un automate d'état fini déterministe.

On appelle automate d'état fini un quintuplet $A = (Q, V, \delta, q_0, F)$ dans lequel :
- Q est un ensemble fini d'états ;
- V est un ensemble fini de symboles (le vocabulaire d'entrée) ;

— $\delta \subseteq Q \times V \cup \{\varepsilon\} \times Q$ est la relation de transition ;
— $q_0 \in Q$ est l'état initial ;
— $F \subseteq Q$ est l'ensemble des états terminaux.

Un automate d'état fini A est dit **déterministe** si et seulement si la relation de transition est une fonction (partielle).

Etant donné un automate d'état fini déterministe, une **configuration** est un couple (q, u) où $q \in Q$ et $u \in V^*$. La relation de transition est alors définie de la manière suivante :

$(q, a.w) \longrightarrow (q', w)$ ssi $\delta(q, a) = q'$. Un mot fini $w = x_0.x_1.x_2.\ldots.x_n$ de V^* est **accepté** par un automate A si et seulement si il existe une exécution $C_0, \cdots, C_n$ telle que $C_0 = (q_0, w)$ et $C_n = (q, \epsilon) \wedge q \in F$.

Le **langage accepté** (ou reconnu) par un automate A, noté $L(A)$ est constitué de l'ensemble des mots acceptés par A.

On a les propriétés suivantes :
— le langage accepté par un automate d'état fini est un langage régulier ;
— à partir de tout automate d'état fini acceptant un langage (régulier) R il est possible de construire un automate d'état fini **déterministe** acceptant le même langage ;
— à partir de toute expression régulière r décrivant un langage (régulier) R il est possible de construire un automate d'état fini déterministe acceptant ce même langage.
— le complément d'un langage régulier est un langage régulier, l'intersection de deux langages réguliers est un langage régulier.

3.4 Application à l'analyse lexicale

L'intérêt des langages réguliers est qu'il existe un algorithme efficace permettant de décider si un mot fini w appartient ou non à un langage régulier L : il suffit en effet de construire un automate A **déterministe** acceptant L et de vérifier que cet automate accepte bien le mot w. L'automate A étant déterministe, cette vérification peut se faire au fur et à mesure d'un parcours unique de w (elle est donc linéaire en fonction de la taille de ce mot).

Par conséquent il est très important pour l'efficacité de l'analyse lexicale que l'**ensemble des lexèmes** d'un langage de programmation soit un **langage régulier** (même si ce langage lui-même n'est pas régulier).

Pour définir cet ensemble de lexèmes, plutôt que d'utiliser un automate d'état fini ou une expression régulière (dont les tailles pourraient être rédhibitoires), il existe des formalismes plus compact en pratique comme par exemple la notation BNF (*Backus-Naur Form*). Cette notation correspond en fait à une séquence d'expressions régulières de la forme

$$a_i ::= r_i \quad (\text{pour } i \in 1 \ldots n)$$

dans laquelle les a_i sont des identificateurs et les r_i sont des expressions régulières définies sur $V \cup \{a_1, a_2, \ldots a_{i-1}\}$. On note r'_i l'expression régulière définie sur V et associée à chaque identificateur a_i. r'_i peut être obtenue en substituant dans r_i chaque occurrence de $a_j \in \{a_1, a_2, \ldots a_{i-1}\}$ par l'expression régulière r'_j correspondante.

4 Langages hors-contexte (algébrique)

4.1 Dérivations

Soit $G = (V_T, V_N, Z, P)$ une grammaire hors-contexte, w et w' des mots sur $V_T \cup V_N$ $(w, w' \in (V_T \cup V_N)^*)$.

On dit alors que :

— w' est un *dérivé direct* par G de w, noté $w \Rightarrow_G w'$, si et seulement si, il existe $X \in V_N$, il existe $u, t, v \in (V_T \cup V_N)^*$ tels que $w = uXv$ et $w' = utv$ et $X \to t \in P$.
— w' est un *dérivé* par G de w, noté $w \Rightarrow_G^* w'$, si et seulement si il existe $w_0, w_1, \ldots, w_n \in (V_T \cup V_N)^*$ tels que $w_0 = w$, $w_n = w'$ et pour tout i, w_{i+1} est un dérivé direct de w_i.

Remarque : $\Rightarrow_G$ et $\Rightarrow_G^*$ sont deux relations définies sur $(V_T \cup V_N)^*$, $\Rightarrow_G^*$ étant la fermeture transitive de $\Rightarrow_G$.

4.2 Langages hors-contexte

Soit $G = (V_T, V_N, Z, P)$ une grammaire hors-contexte. Le *langage défini* par G (on dit aussi langage *reconnu* par G) est l'ensemble $L(G)$ tel que :

$$L(G) = \{w \in V_T^* \mid Z \Rightarrow_G^* w\}$$

Un langage L est dit *hors-contexte* s'il existe une grammaire hors-contexte qui le reconnaît.

Remarque : Tout langage régulier est un langage hors-contexte, il existe des langages hors-contexte qui ne sont pas réguliers.

4.3 Arbres de dérivation

Soit $G = (V_T, V_N, Z, P)$ une grammaire et w un mot de $L(G)$. On appelle *arbre de dérivation* (ou *arbre syntaxique*) de w dans G tout arbre n-aire A dont les nœuds sont des éléments de $(V_T \cup V_N)$ et tel que :
— la racine de A est Z (l'axiome de G) ;
— les feuilles de A sont des éléments de $V_T \cup \{\varepsilon\}$ et la séquence gauche-droite des feuilles de A est égale à w ;
— les nœuds non feuilles de A sont des éléments de V_N et si un nœud non feuille X a pour fils u_1, u_2, $\ldots u_n$ dans A alors la règle $X \to u_1.u_2 \ldots u_n$ doit appartenir à P.

A tout mot de $L(G)$ on peut associer un arbre de dérivation. Si de plus cet arbre est *unique* alors la grammaire est dite *non ambigüe*.

4.4 Automates à Piles

Un *automate à pile* est un 7-uplet $M = (Q, V, \Gamma, \Delta, Z_0, q_0, F)$ tel que :
— Q est un ensemble fini d'états ;
— V est l'alphabet d'entrée ;
— Γ est l'alphabet de pile ;
— $Z_0 \in \Gamma$ est le symbole initial de pile ;
— $q_0 \in Q$ est l'état initial ;
— $F \subseteq Q$ est l'ensemble des états accepteurs
— $\Delta \subseteq ((Q \times V^* \times \Gamma) \times (Q \times \Gamma^*))$ est la relation de transition.

Un automate à pile permet de reconnaître un ensemble de mots de V^* (le langage accepté, ou reconnu par l'automate) : à chaque transition $((q, u, \beta), (q', \gamma)) \in \Delta$ l'automate passe de l'état q à 'état q' en lisant le préfixe u sur la chaîne d'entrée et en remplaçant le sommet de pile β par γ.

Etant donné un automate à pile, une **configuration** est un élément $(q, w, \alpha) \in Q \times V^* \times \Gamma^*$ dans laquelle q dsigne l'état courant de l'automate, w la chaîne d'entrée et α le contenu de la pile.

La relation de transition est alors définie de la manière suivante :

$(q, w, \alpha) \longrightarrow (q', w', \alpha')$ ssi $((q, u, \beta), (q', \gamma)) \in \Delta$ avec :

— $w = uw'$ (le préfixe u a été lu sur la chaîne d'entrée) ;
— $\alpha = \beta.\delta$ et $\alpha' = \gamma.\delta$ (le sommet de pile β a été remplacé par γ).

Un automate à pile est dit *déterministe* si et seulement si pour chaque configuration C il existe *au plus* une configuration C' dérivable à partir de C.

Un mot w de V^* est *reconnu* (ou *accepté*) par M si et seulement si il existe une exécution $C_0, C_1, \ldots C_n$ telle que :
— $C_0 = (q_0, w, Z_0)$ (l'automate est dans son état initial, la pile ne contient que le symbol initial, la chaîne d'entrée contient w) ;
— $C_n = (q, \varepsilon, \gamma)$ avec $q \in F$ (l'automate a atteint un état final, la chaîne d'entrée est vide)

Le *langage accepté ou reconnu par* M, $L(M)$, est alors défini par : $L(M) = \{w \in V^* \mid w \text{ est reconnu par } M\}$

4.5 Automates à pile et langages hors-contexte

On a les propriétés suivantes :
— le langage reconnu par un automate à pile est un langage hors-contexte ;
— pour tout langage hors-contexte il existe un automate à pile qui le reconnaît ;
— il n'y a pas équivalence entre automates à pile déterministes et non-dèterministes : il existe des langages hors-contexte non reconnaissables par un automate à pile déterministe.