Interface PM
IFT2905
Fuille de notes

Franz Girardin

13 mai 2024

Table des matières

Style de language de programmation

- ▶ **Impératif**
  - ▷ Procédural
  - ▷ Objet
- ▶ **Déclaratif**
  - ▷ Fonctionnel
  - ▷ Logique
- ▶ **Concurrent**
  - ▷ Mémoire partagée
  - ▷ Passage de message

Le **niveau d'abstraction** d'un language est la distance conceptuelle dudit language par rapport au language machine.

Programmation impérative procédurale

- ▶ `Fortran`, `Algol 60`, `Pascal`, `C`, `Ada`

Effectuent des opérations sur la mémoire ; les instructions sont regroupées en *procédures*. Peut facilement être traduit (compilé) en language machine.

Programmation impérative OO

- ▶ `Simula`, `Smalltalk`, `C++`, `Java`

Chaque objet de la mémoire est accompagné de code qui lui permet d'interagir avec les autres objets. Les *méthodes* remplacent les *procédures*. Le **flot de contrôle** passe d'un objet à l'autre par appel de méthode.

Programmation Fonctionnelle

- ▶ `Lisp`, `ML`, `Haskell`, `APL`

Une fonction est un calcul. Facilite l'analyse et le raisonnement ; limite les effets de bord. Généralement apprécié pour son élégence.

Section 1

Syntaxe des expressions

La notation **infixée** est plus familière et intuitive mais elle peut aussi être embigue.

- ▷ **Niveau de précédence** $a + b * c \equiv a + (b * c)$
- ▷ **Associativité** $a - b - c \equiv (a - b) - c$

Les niveaux de précédences établisssent la priorité des opérations et associent les termes des expressions de la gauche vers la droite. Chaque langage peut avoir une grande quantité de niveaux et ils peuvent s'avérer difficile à mémoriser.

Définition formelle de la syntaxe Un **langage** est un ensemble de *phrases* qui sont composées de *séquences de symboles* ; cest symboles représentent le vocabulaire du language. La **grammaire** est l'ensemble des règles précisant l'usage du langage.

Language et grammaire Il est possible de définir une langage $L(G)$ à partir d'une grammaire G. L'ensemble $L(G)$ est l'ensemble des *chaînes* et des *phrases*qui peuvent être généré en utilisant les règles de grammaire G. Les éléments $L(G)$ sont noté p et représentent les phrases ou chaînes de caractèresproduites par la grammaire. L'expression $\texttt{depart} \implies \cdots \implies p$ signifie qu'il est possible d'utiliser un **symbole** initial et appliquer n'importe quelle série de règle de production de la grammaire G pour arriver à la chaîne p.

$$L(G) = \left\{ p \mid \langle \texttt{départ} \rangle \right\}$$

Backus-Naur Form La BNF est un système de notation pour décrire la syntaxe d'un langage sous forme de **règles de production**. Chaque règle décrit une structure syntaxique spécifique en termes de séquences d'autres structures, qui peuvent être des symboles terminaux (c'est-à-dire les éléments de base du langage, tels que des mots-clés, des opérateurs ou des identificateurs) ou d'autres structures syntaxiques définies par des règles.

Exemple 1

Pour définir une **catégorie** en BNF, on peut utiliser la syntaxe suivant

$$\langle \texttt{cat} \rangle ::= x_1, x_2, \cdots x_n$$

Pour définir un binaire, on peut utiliser la syntaxe suivante

$$\langle \texttt{bin} \rangle ::= 0$$
$$\langle \texttt{bin} \rangle ::= 1$$
$$\langle \texttt{bin} \rangle ::= \langle \texttt{bin} \rangle \langle \texttt{bin} \rangle$$

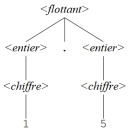
Définition 1 Dérivation directe

Il s'agit de l'application d'une **règle de production** définie en BNF Puisqu'il existe une règle définissant $\langle bin \rangle$, il est possible d'appliquer une dérivation directe de cette règle pour obtenir le binaire `01` :

$$\langle \texttt{bin} \rangle ::= \langle \texttt{bin} \rangle \langle \texttt{bin} \rangle \implies \langle \texttt{bin} \rangle \ \texttt{0} \implies \texttt{1 0}$$

Exemple 2 BNF pour type numérique

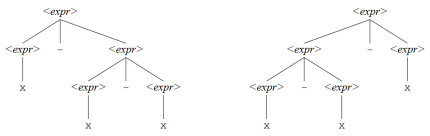
$$\langle \texttt{flottant} \rangle ::= \langle \texttt{entier} \rangle \ . \ \langle \texttt{entier} \rangle$$
$$\langle \texttt{entier} \rangle ::= \langle \texttt{chiffre} \rangle$$
$$\langle \texttt{entier} \rangle ::= \langle \texttt{chiffre} \rangle \langle \texttt{entier} \rangle$$
$$\langle \texttt{chiffre} \rangle ::= \langle \texttt{(0|1|2|3|4|5|6|7|8|9)} \rangle$$



On peut vérifier que l'expression `1.5` est un flottant selon la définission BNF en observant l'**arbre de dérivation** suivant dans lequel le départ est la racine et chaque phrase est une feuille

Grammaires ambigues Une grammaire est **ambigue** ssi il existe une phrase dans $L(G)$ qui a plusieurs arbres de dérivations.

> **Exemple 3** Phrase ambigue
>
> $\langle \texttt{expr} \rangle ::= x$
> $\langle \texttt{expr} \rangle ::= \langle \texttt{expr} \rangle - \langle \texttt{expr} \rangle$



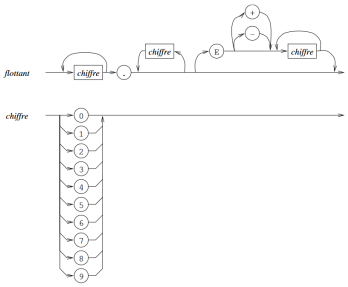
Autres motifs de syntaxes BNF

$x_1|x_2$: *peut être* x_1 **ou** x_2

(x) : *groupement*

$[x]$ **ou** $x?$ **ou** $\varepsilon|x$: x *peut apparaître* **0** *ou* **1** *fois*

$\{x\}$ **ou** : x *peut être répété* **0** *ou* **plusieurs** *fois*



Sucre syntaxique

- ▷ Exension syntaxique superficielle **équivalente** à une autre syntaxe
- ▷ Pas d'impact sur les propriétés internes du langage

Langage fonctionnel Les langages **fonctionnels** fournissent une environnment pour générer du code à un haut niveau d'abstraction. Le programmation fonctionnelle est un paradigme de programmation pour laquelle les expressions sont plus importantes que les affirmations. On compose ainsi les programmes en utilisant des **expressions** ; chacune d'elles génère une valeur et ces expressions peuvent être combinées pour engendrer une expression plus complexes. Cette approche est différente de la programmation impérative où les affirmations ont un effet sur l'état global et où les affirmations communiquent des valeurs via l'état global.