Université de Montréal

Syntaxe et sémantique

Syntaxe = représentation du programme

Sémantique = sens attaché aux programmes

- ```
  if (x == 0) y = 5;
  ```
  ```
  if x = 0 then y := 5;
  ```
  ```
  (if (= x 0) (set! y 5))
  ```

- ```
  6 + 1 / 3
  ```
  $\Longrightarrow 6$

  ```
  6 + 1 / 3
  ```
  $\Longrightarrow 6.33\overline{3}$

  ```
  6 + 1 / 3
  ```
  $\Longrightarrow 2.33\overline{3}$

Université de Montréal

Ceci n'est pas un nombre

42

Syntaxe des expressions (1)

Préfixe	$+\ e_1\ e_2$	Fonctions en math
Infixe	$e_1 + e_2$	Opérateurs en math
Postfixe	$e_1\ e_2\ +$	Calculatrices HP

Infixe est familière mais plus ambiguë:

$$a + b * c \stackrel{?}{\equiv} a + (b * c) \stackrel{?}{\equiv} (a + b) * c$$
$$a - b - c \stackrel{?}{\equiv} a - (b - c) \stackrel{?}{\equiv} (a - b) - c$$
$$\text{not } x < 0 \stackrel{?}{\equiv} (\text{not } x) < 0 \stackrel{?}{\equiv} \text{not } (x < 0)$$
$$\sin a - b \stackrel{?}{\equiv} \sin (a - b) \stackrel{?}{\equiv} (\sin a) - b \stackrel{?}{\equiv} (\sin a)(-b)$$

Université de Montréal

Syntaxe des expressions (2)

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

C a 15 niveaux!

Ils s'associents à gauche. Exceptions: unaires, $x?y : z$ et affectations

1 seule associativité par niveau

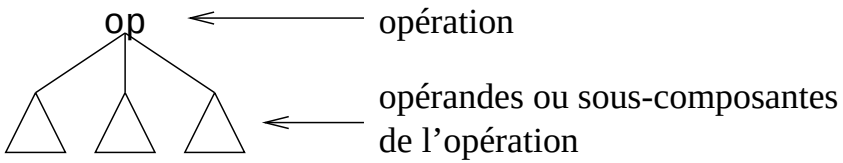
Il est difficile de mémorizer tous les niveaux

Université de Montréal

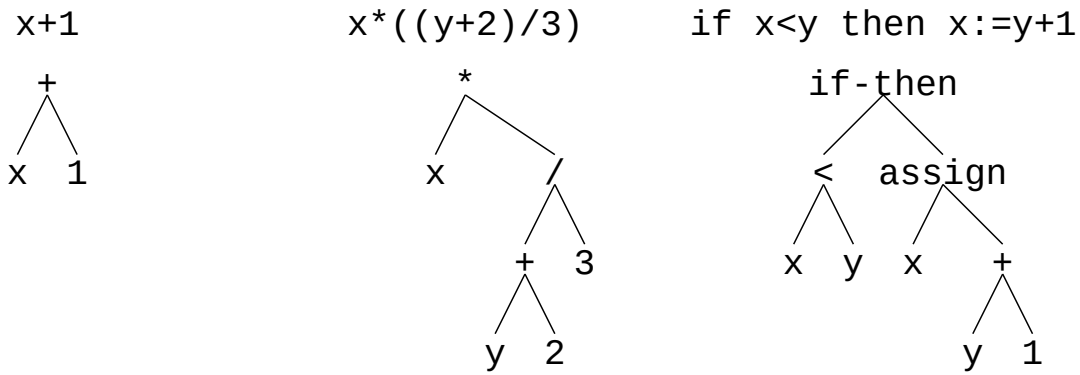
Syntaxe abstraite

Représentation structurelle, dépouillée des détails de la syntaxe

Forme générale:



Examples:



Pas de parenthèses et autre séparateurs sans importance

Point milieu entre syntaxe et sémantique

Définition formelle de la syntaxe

Tirée des concepts d'informatique théorique

Définir une *grammaire* qui spécifie exactement les programmes syntaxiquement corrects.

- Déf: *Vocabulaire* (Σ) = ensemble de symboles
- Déf: *Phrase* = séquence de symboles tirés de Σ.
- Déf: *Langage* = ensemble de phrases
- Déf: *Grammaire* = régles décrivant précisément le langage

Chapitre 2 de Sethi

Backus-Naur Form

Grammaire en format BNF: ensemble de *catégories* et de *productions* avec une catégorie désignée *catégorie de départ*

- *Catégorie* = nom d'un type de fragment de phrase
  Ex: $\langle$expression$\rangle$, $\langle$entier$\rangle$, $\langle$type$\rangle$
- *Production* = règle de la forme

$$\langle \text{cat} \rangle ::= x_1 \; x_2 \; \ldots \; x_n$$

où $\langle$cat$\rangle$ est une catégorie et x_i est une catégorie ou un symbole

Exemple:
$$\langle \text{bin} \rangle ::= 0$$
$$\langle \text{bin} \rangle ::= 1$$
$$\langle \text{bin} \rangle ::= \langle \text{bin} \rangle \; \langle \text{bin} \rangle$$

Dérivation BNF

Déf: *Dérivation directe* = application d'une règle de production

Ex: $y_1 \ldots \langle \text{cat} \rangle \ldots y_m \Rightarrow y_1 \ldots x_1 \ldots x_n \ldots y_m$

il existe une régle $\langle \text{cat} \rangle ::= x_1 \ldots x_n$

Déf: *Dérivation* = séquence de dérivations directes

commence par la catégorie de départ et se termine par une phrase

Ex: $\langle \text{bin} \rangle \Rightarrow \langle \text{bin} \rangle \ \langle \text{bin} \rangle \Rightarrow \langle \text{bin} \rangle \ 0 \Rightarrow 1 \ 0$

Déf: L(G) = langage défini par la grammaire G

L(G) = $\{ \ p \mid \langle \text{départ} \rangle \Rightarrow \cdots \Rightarrow p \ \}$

Exemple de BNF

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

1.5 est-il un flottant? $\langle \text{flottant} \rangle \Rightarrow \langle \text{entier} \rangle \ . \ \langle \text{entier} \rangle \Rightarrow$
$\langle \text{chiffre} \rangle \ . \ \langle \text{entier} \rangle \Rightarrow \langle \text{chiffre} \rangle \ . \ \langle \text{chiffre} \rangle \Rightarrow 1 \ . \ \langle \text{chiffre} \rangle \Rightarrow 1 \ . \ 5$

Université de Montréal

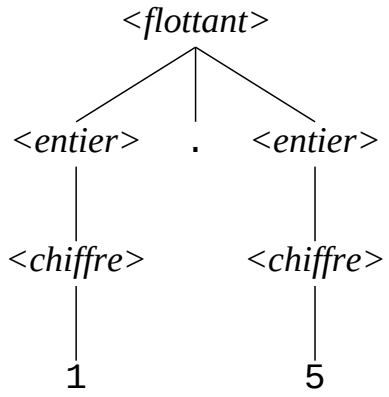
Arbre de dérivation

Arbre capturant la structure syntaxique d'une phrase

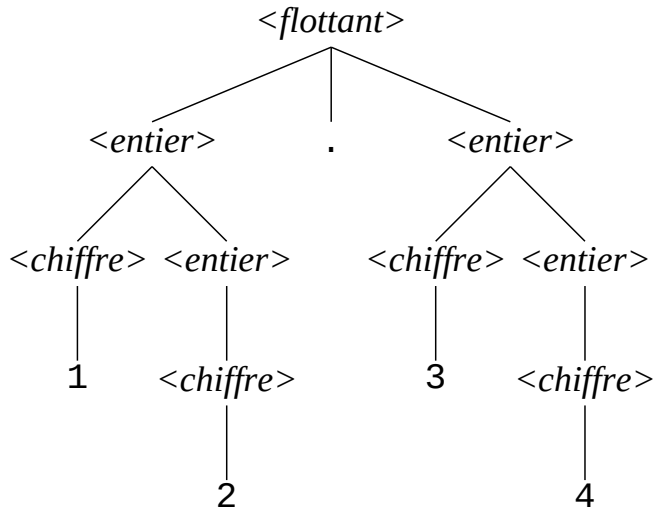
Représentation compacte d'une dérivation

Exemples:

`1.5`



`12.34`



Racine = départ, feuilles = phrase. Presque un ASA

Université de Montréal

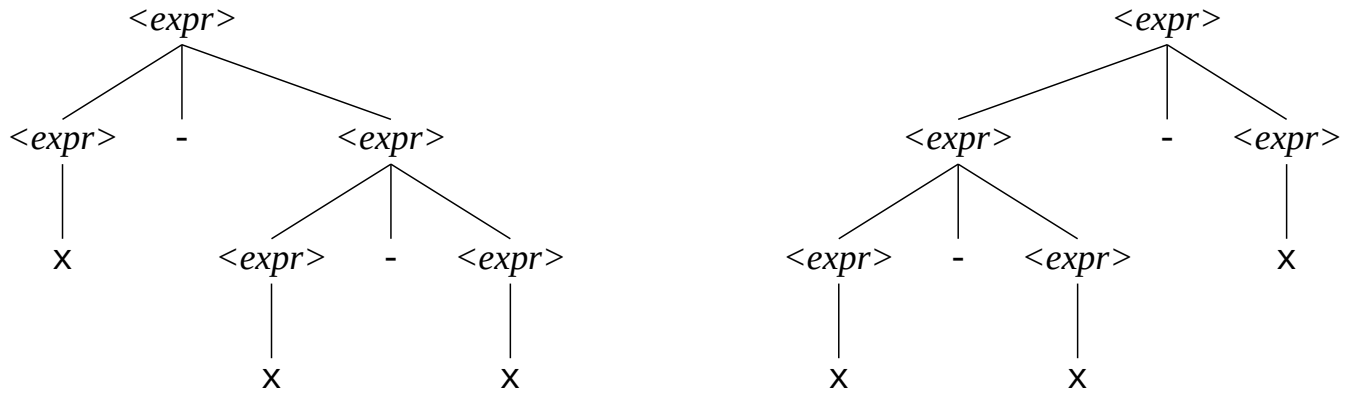
Grammaires ambiguës

Déf: une grammaire G est *ambiguë* ssi il existe une phrase dans L(G) qui a plusieurs arbres de dérivation (pas juste plusieurs dérivations)

Exemple: $\langle \text{expr} \rangle ::= x$

$\langle \text{expr} \rangle ::= \langle \text{expr} \rangle - \langle \text{expr} \rangle$

$x - x - x$:



Résoudre ou éliminer les ambiguïtés en BNF pour l'utiliser dans l'ASA

Extended BNF

Extension de la syntaxe BNF avec des expressions régulières:

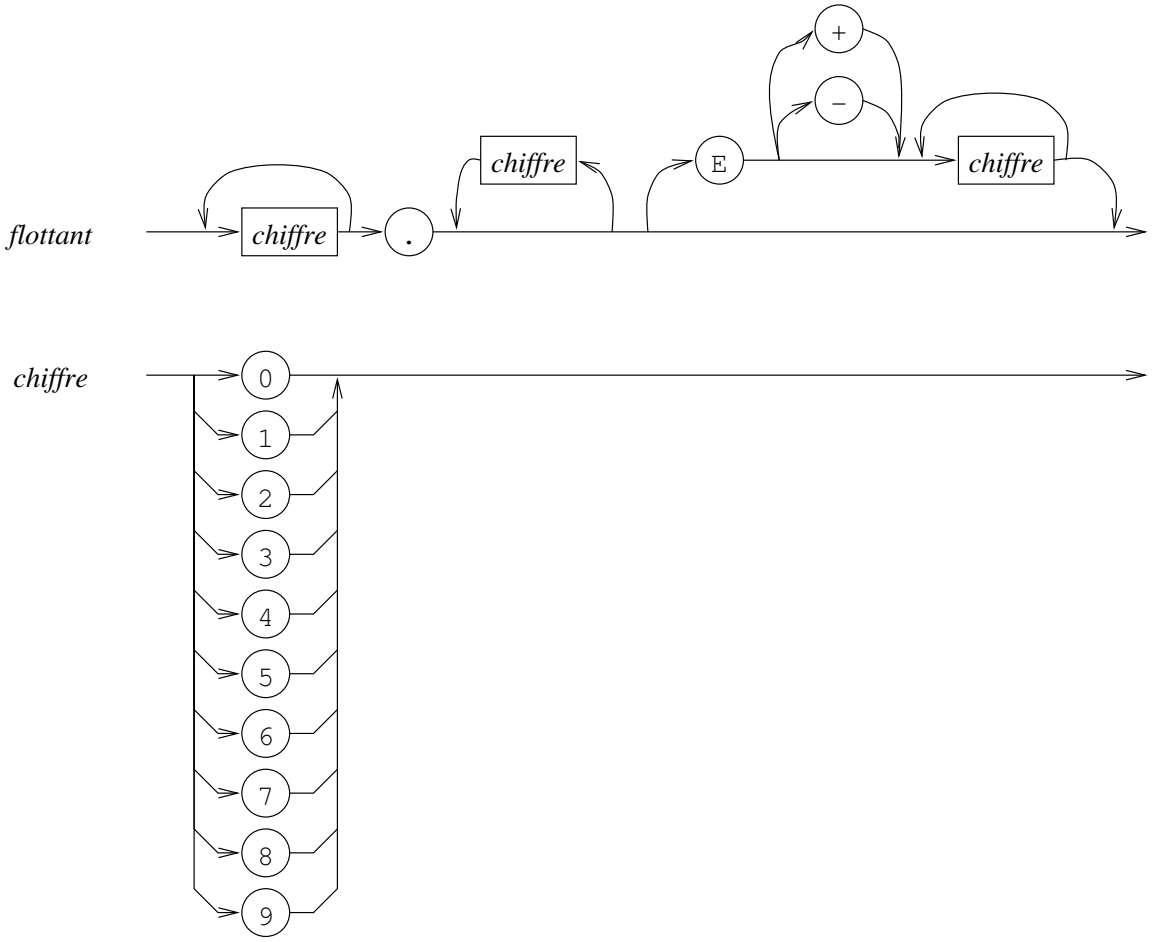
$x_1 \vert x_2$	peut-être soit x_1 soit x_2
(x)	groupement
$[x]$	parfois noté $x?$ équivalent à $\varepsilon \vert x$
$\{x\}$	parfois noté $x*$

Exemple: $\langle \text{entier} \rangle ::= \langle \text{chiffre} \rangle \{ \langle \text{chiffre} \rangle \}$

Université de Montréal

Diagramme syntaxique

Représentation graphique d'une catégorie d'une grammaire



Spécification et implantation

Spécification $\Rightarrow$ syntaxe et sémantique sans implantation

Pas de spécification $\Rightarrow$ pas de bug de l'implantation

Une implantation fidèle à la spécification est dite *conforme*

L'implantation a les mains libres quant à la manière de procéder

Les spécifications permettent d'offrir des *extensions*

Université de Montréal

Définition d'un langage

- Syntaxe
  - Règles lexicales
  - Grammaire
- Sémantique
  - Règles de typage (sémantique statique)
  - Règles d'évaluation (sémantique dynamique)

$$[Char] \Rightarrow [Lexeme] \Rightarrow ASA \Rightarrow AnnotatedASA \Rightarrow Value$$

Université de Montréal

Sucre syntaxique

Extension syntaxique superficielle

Absolument équivalente à une autre syntaxe

Gérable lors de la conversion vers l'ASA

Pas d'impact sur les propriétés internes du langage

Langages utilisés

Dans ce cours apparaîtront probablement les langages suivants:

- Haskell: Langage fonctionnel pur “Avoid success at all costs”
- Scheme: Roi de la méta-programmation
- Prolog: Grand-père de la programmation logique
- C: Programmation impérative de bas niveau
- Rust: Même chose, mais avec filet de sécurité
- Curry: Mélange de programmation fonctionnelle et logique