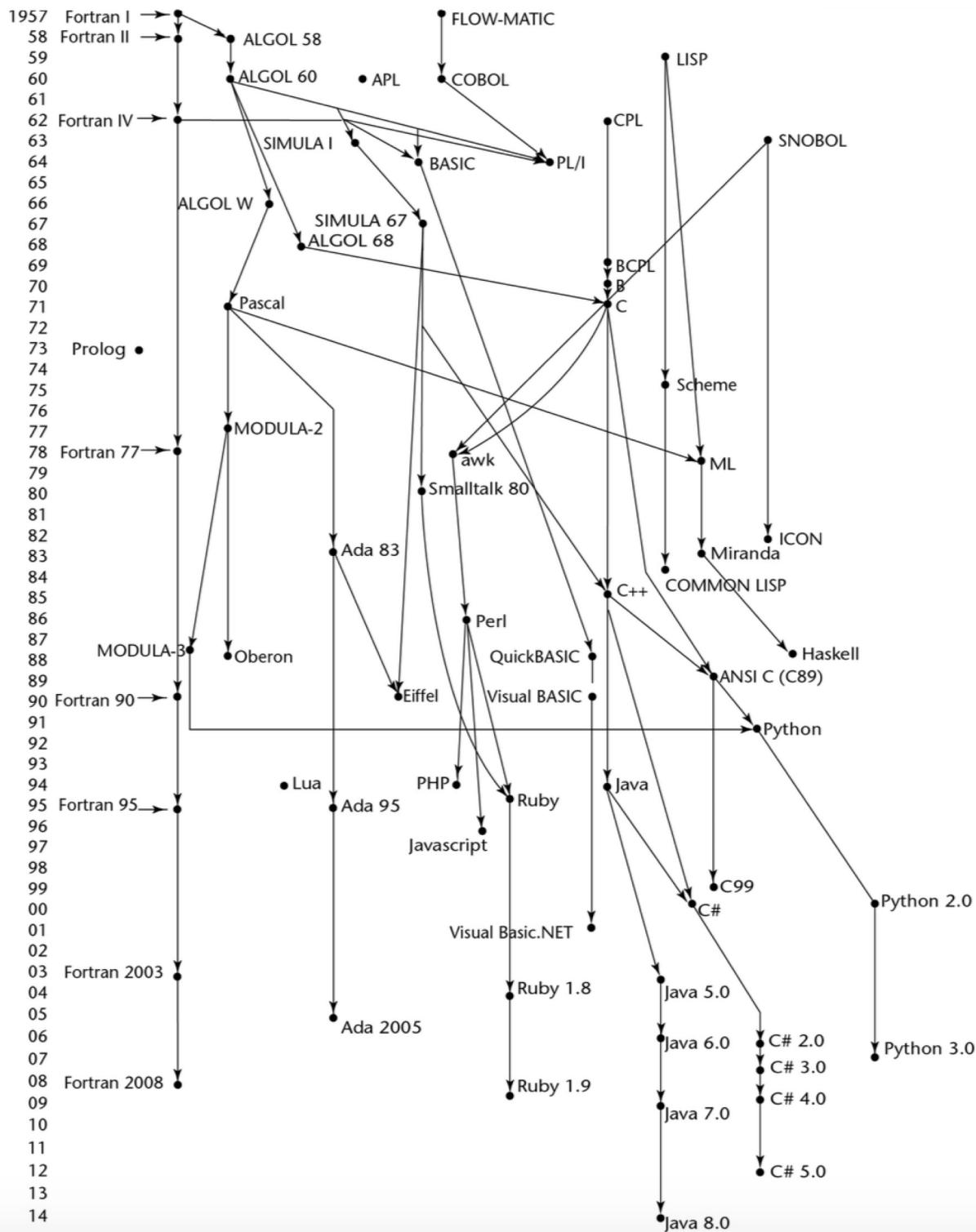
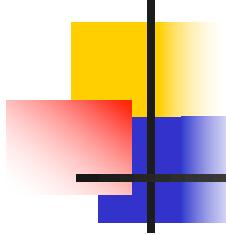


Grammaires

CSI 3520
Lab 7

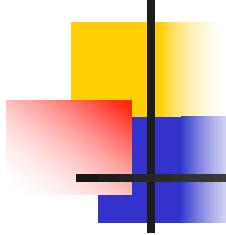
Langages de programmation à haut-niveau





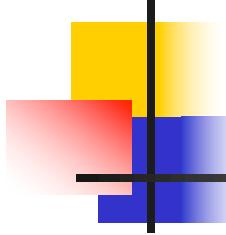
Introduction

- **Syntaxe:** la forme ou la structure des expressions, des instructions et des unités de programme
- **Sémantique:** la signification des expressions, des instructions et des unités de programme
- La syntaxe et la sémantique fournissent la définition d'un langage



Description d'une syntaxe

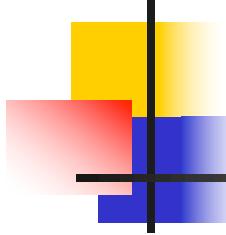
- Une **phrase** est une chaîne de caractères sur un alphabet
- Une **langue** est un ensemble de phrases
- Un **lexème** est l'unité syntaxique de niveau le plus bas d'un langage (par exemple, *, sum, begin)
- Un **jeton** (token) est une catégorie de lexèmes (par exemple, un identifiant)



Définition formelle d'une langues

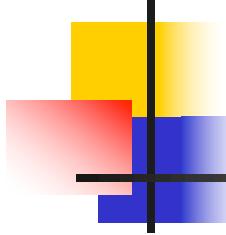
■ Reconnaissance:

- Un dispositif qui lit les chaînes d'entrée sur l'alphabet de la langue et décide si les chaînes d'entrée appartiennent à la langue
- Ex: partie analyse syntaxique d'un compilateur



Suite

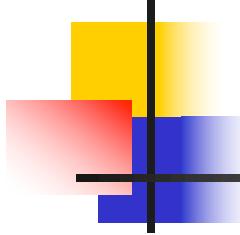
- Générateurs:
 - Un appareil qui génère des phrases d'une langue
 - On peut déterminer si la syntaxe d'une phrase particulière est syntaxiquement correcte en la comparant à la structure du générateur



CFG et BNF

■ **Context-Free Grammars:**

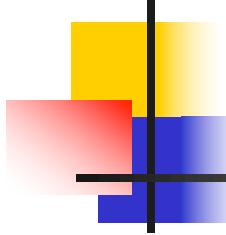
- Développé par Noam Chomsky au milieu des années 1950
- Destinés à décrire la syntaxe des langues naturelles
- Définir une classe de langages appelés *context-free languages*



Suite

- **Backus-Naur Form:**

- Inventé par John Backus pour décrire Algol 58
- La notation pour CFG est souvent appelée Backus-Naur Form (BNF)



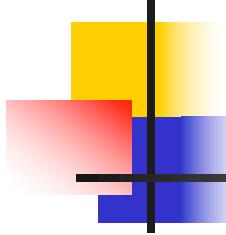
Grammaires sans contexte

- Un CFG se compose de
 - Un ensemble de terminaux **T**
 - Un ensemble de non-terminaux **N**
 - Un symbole de début **S** (non terminal)
 - Un ensemble de productions / règles

$\langle \text{assign} \rangle \rightarrow \langle \text{var} \rangle = \langle \text{expression} \rangle$

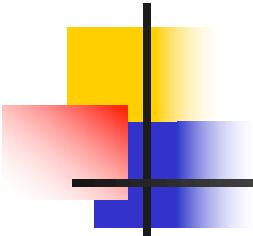
$\langle \text{var} \rangle \rightarrow A \mid B \mid C$

$\langle \text{expression} \rangle \rightarrow \langle \text{var} \rangle + \langle \text{var} \rangle$



Expressions régulières

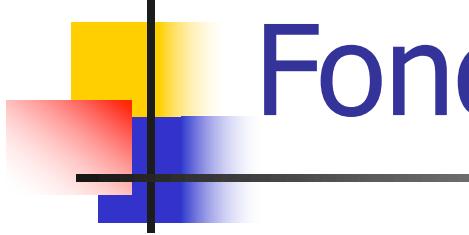
- Une expression régulière est l'une des suivantes:
 - Caractère
 - Chaîne vide
 - Deux expressions régulières concaténées
 - Deux expressions régulières séparées par |
 - Une expression régulière suivie de l'étoile Kleene * (concaténation de zéro ou plus de chaînes)

- 
- Les littéraux numériques en Pascal peuvent être générés par ce qui suit:

digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

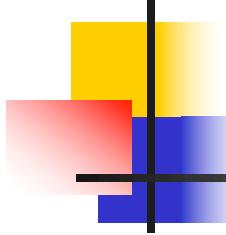
unsigned_integer → *digit digit* *

unsigned_number → *unsigned_integer* ((. *unsigned_integer*) | ϵ)
 ((((e | E) (+ | - | ϵ) *unsigned_integer*) | ϵ)



Fondamentaux de BNF

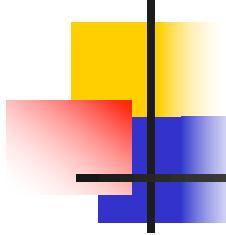
- Les abstractions sont utilisées pour représenter des classes de structures syntaxiques
 - Comme les variables syntaxiques (aussi appelées symboles non terminaux, ou tout simplement non-terminaux)
- Les terminaux sont des lexèmes ou des jetons
- Une règle a
 - LHS - qui est non-terminal,
 - RHS - qui est une chaîne de terminaux et/ou non-terminaux



Suite

<i>digit</i>	→	0		1		2		3		4		5		6		7		8		9
--------------	---	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---

- Les non-terminaux sont souvent entourés de crochets
- Une règle simple
<if_stmt> → if <logic_expr> then <stmt>
- Grammaire: un ensemble de règles non-vide fini
- Un symbole de début est un élément spécial des non-terminaux d'une grammaire

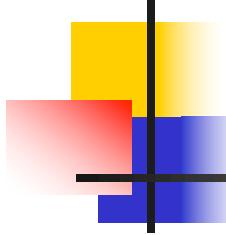


Règles BNF

- Une abstraction (ou symbole non terminal) peut avoir plus d'une RHS

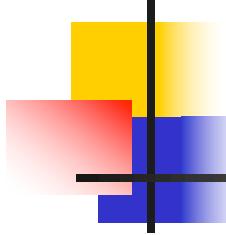
$$\begin{aligned} <\text{stmt}> \rightarrow & <\text{single_stmt}> \\ & | \text{ begin } <\text{stmt_list}> \text{ end } \end{aligned}$$

$digit$	\longrightarrow	0		1		2		3		4		5		6		7		8		9
---------	-------------------	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---



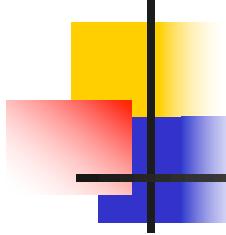
Les règles peuvent être récursives

- Les listes syntaxiques sont décrites comme suit:
$$\begin{aligned} <\text{ident_list}> \rightarrow & \text{identifier} \mid \text{identifier}, \\ & <\text{ident_list}> \end{aligned}$$
- Une **dérivation** est une application répétée de règles, commençant par le symbole de début et se termine par une phrase



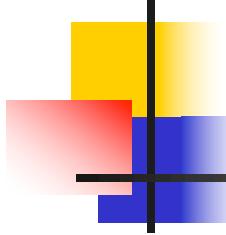
Exemple

- **<assign>** -> **<var>** = **<expression>**
- Total = subtotal1 + subtotal2
 - LHS: l'abstraction étant définie
 - RHS: mélange de jetons, de lexèmes et de références à d'autres abstractions
- les abstractions **<var>** et **<expression>** doivent évidemment être définies pour que la définition **<assign>** soit utile



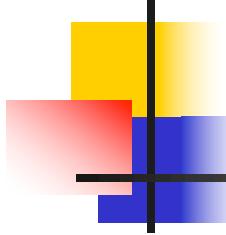
Suite

- **<assign> -> <var> = <expression>**
- Cette règle particulière spécifie que l'abstraction <assign> est définie comme une instance de l'abstraction <var>, suivie par le lexème =, suivie d'une instance d'abstraction <expression>



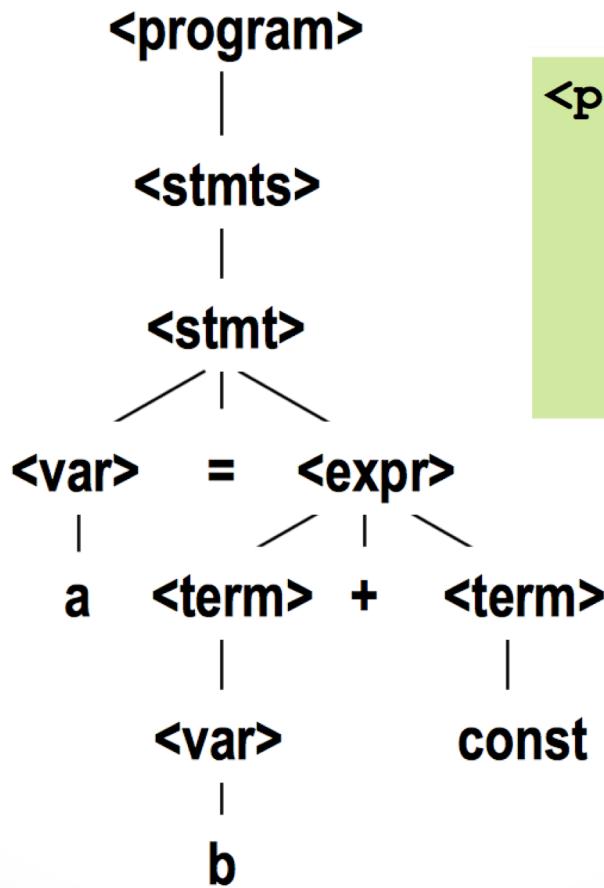
Dérivations

- Une dérivation est une application répétée de règles, commençant par le symbole de début et se terminant par une phrase (tous les symboles de terminal)
- BNF est un générateur
 - Utiliser la grammaire pour générer des phrases appartenant à la langue décrite par cette grammaire

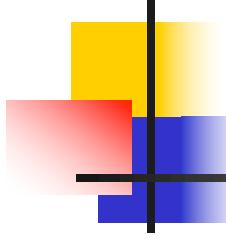


Arbre d'analyse

- Une représentation hiérarchique d'une dérivation



```
<program> → <stmts>
<stmts> → <stmt> | <stmt> ; <stmts>
<stmt> → <var> = <expr>
<var> → a | b | c | d
<expr> → <term> + <term> | <term> - <term>
<term> → <var> | const
```



Example 1

- $\text{Index} = 2 * \text{count} + 17;$

Lexemes	Tokens
index	identifier
=	Equal_sign
2	Int_literal
*	Mult_op
count	Identifier
+	Plus_op
17	Int_literal
;	semicolon

Exemple 2

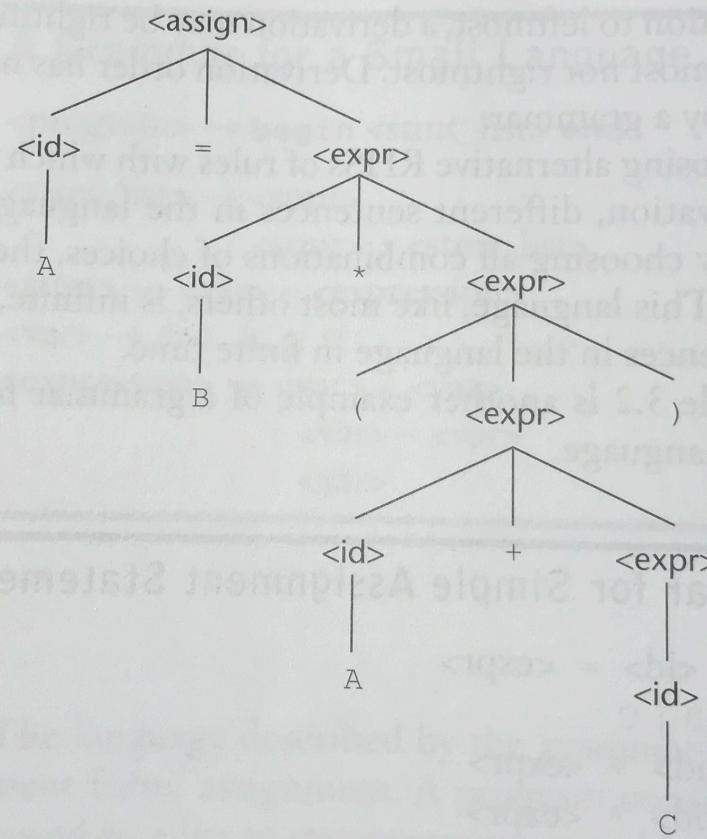
118

Chapter 3 Describing Syntax and Semantics

Figure 3.1

A parse tree for the simple statement

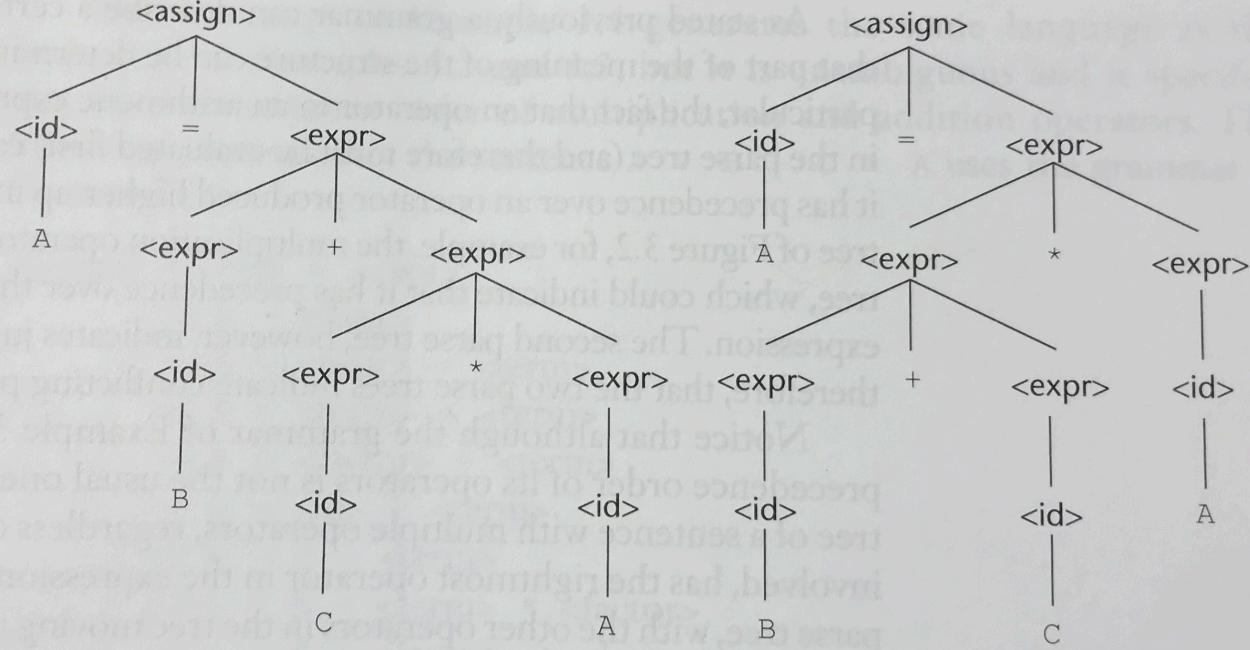
$A = B * (A + C)$

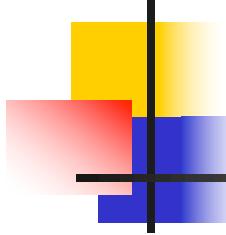


Exemple 3

Figure 3.2

Two distinct parse trees
for the same sentence,
 $A = B + C * A$





Ressources

- SWI-Prolog
 - <http://www.swi-prolog.org/>
- Visual Prolog
 - <http://www.visual-prolog.com/>
- GNU Prolog
 - <http://gnu-prolog.inria.fr/>
- InterProlog (Java)
 - <http://www.declarativa.com/interprolog/>
- P# (.NET)
 - <http://www.dcs.ed.ac.uk/home/jjc/psharp/psharp-1.1.4/dlpsharp.html>