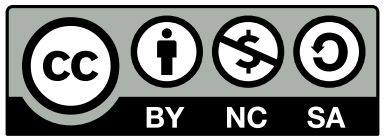
Chapitre 3 - Analyse syntaxique

INF7641 Compilation

Jean Privat

Université du Québec à Montréal

INF7641 Compilation
v251



Plan

Grammaires non contextuelles

Analyse syntaxique

Analyseur syntaxique

- Donnée : une séquence finie de jetons
- Résultat : une structure syntaxique (un arbre)

Générateur d'analyseur syntaxique

- Donnée : une description de syntaxe (une grammaire)
- Résultat : le code de l'analyseur syntaxique correspondant

Pour l'instant, c'est magique

- SableCC4 est aussi un générateur d'analyseur syntaxique
- Algorithmes pour plus tard (spoiler : on y parle d'automates)

Grammaire

Langages

- Décrire des langages par leur structure syntaxique
- Représentation intuitive
  ... plus que les expressions régulières ?

Exemple

- Une expression est la somme de deux expressions, ou le produit de deux expressions, ou un nombre

Questions

- Un mot (une « phrase ») appartient-il au langage ?
- Si oui, quelle est sa structure (son arbre) syntaxique ?

Grammaire non contextuelle

Grammaire hors-contexte, grammaire algébrique ou *context-free grammar*

Jeton (terminal)

- Élément de l'alphabet du langage
- Seul le **type** du jeton est considéré et non le texte (contenu)

Production (non-terminal)

- Désigne un ensemble d'**alternatives** (séparées par des |)
- Une des productions est spéciale, celle de **départ** (racine)
  C'est par convention la première production

Alternative

- Une séquence d'**éléments** (jetons ou productions)
- Alternative possiblement nommée (`{...:}` en SableCC4)
- Élément possiblement nommé (`[...:]` en SableCC4)

Exemple : le langage des formes

```
Language formes;
Lexer
  nombre = ('0'..'9')+;
  Ignored ' ';
Parser
  forme = {cercle:} 'centre' point 'rayon' long |
          {segment:} [src:]point '--' [dst:]point ;
  point = '(' [x:]long ',' [y:]long ')' ;
  long = nombre unite ;
  unite = 'cm' | 'mm' | 'pt' | 'px' ;
```

- Quels sont les jetons ? productions ? alternatives ? éléments ?

Arbre syntaxique

- Représentation structurelle d'un mot d'un langage
- Racine : production de départ
- Nœuds intermédiaires : productions
- Enfants : les éléments d'une alternative
- Feuilles : jetons

Le langage des formes

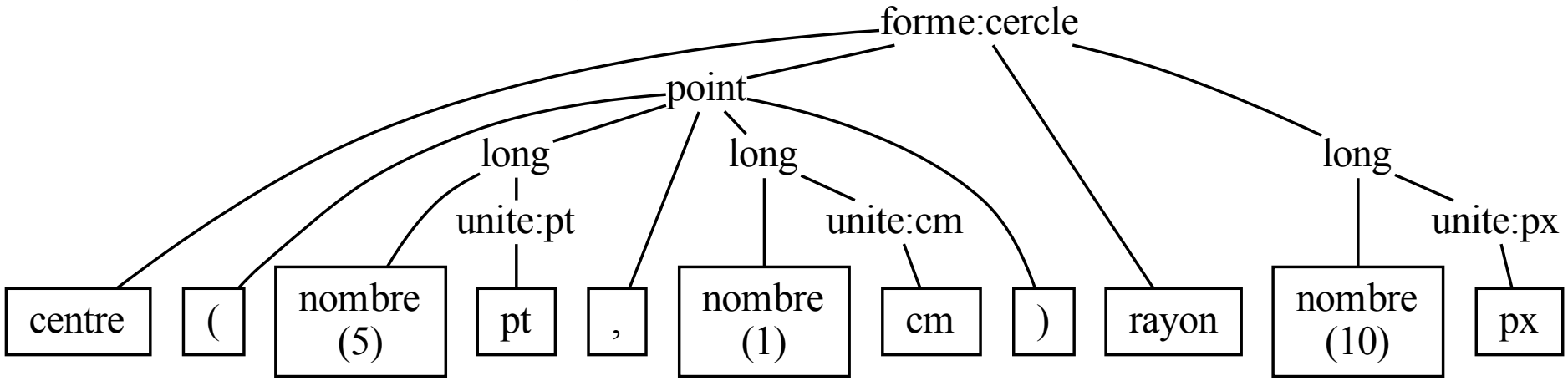
Exercice : Trouver l'arbre syntaxique

```
centre (5pt, 1cm) rayon 10 px
```

Le langage des formes

Exercice : Trouver l'arbre syntaxique

```
centre (5pt, 1cm) rayon 10 px
```



Le langage des listes

```
list = {many:} id list |
       {one:} id ;
```

Exercice : Trouver l'arbre syntaxique

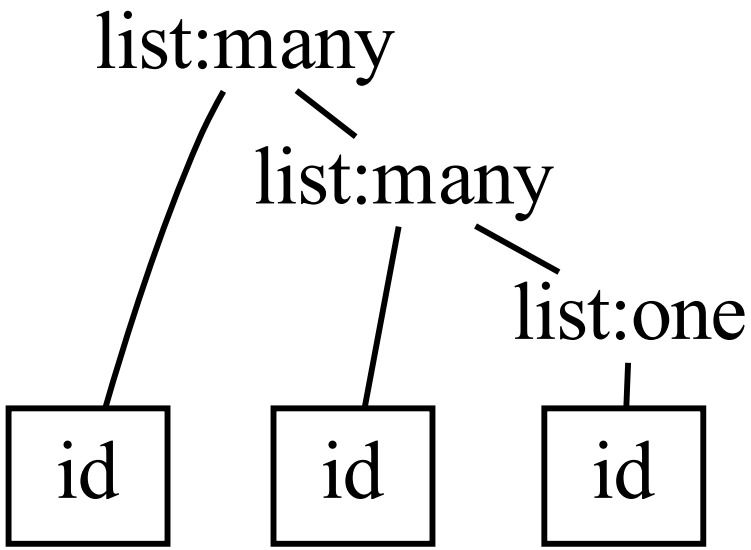
```
id id id
```

Le langage des listes

```
list = {many:} id list |
       {one:} id ;
```

Exercice : Trouver l'arbre syntaxique

```
id id id
```



Le langage des parenthèses

```
par = {item:} '(' par ')' |
      {empty:} '(' ')' ;
```

Exercice : Trouver l'arbre syntaxique

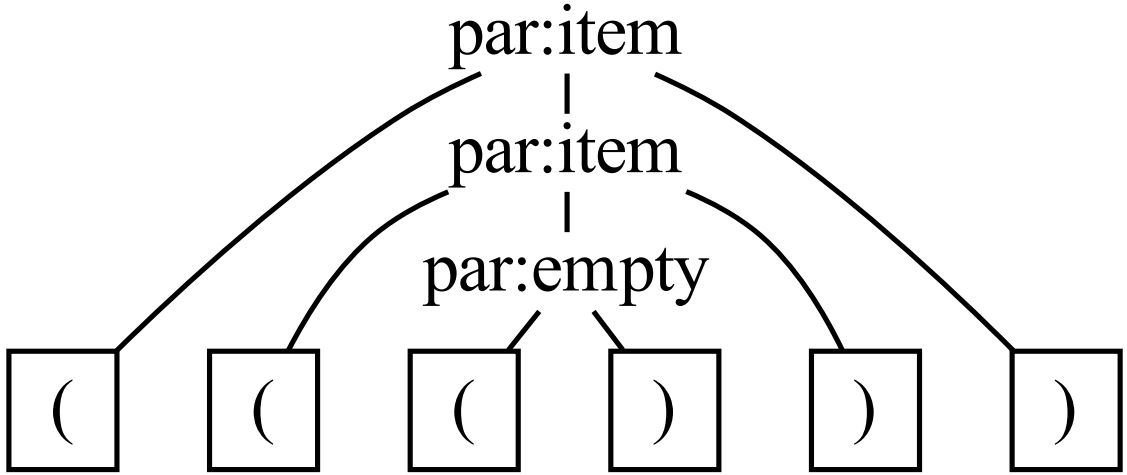
```
((()))
```

Le langage des parenthèses

```
par = {item:} '(' par ')' |
      {empty:} '(' ')' ;
```

Exercice : Trouver l'arbre syntaxique

```
((()))
```



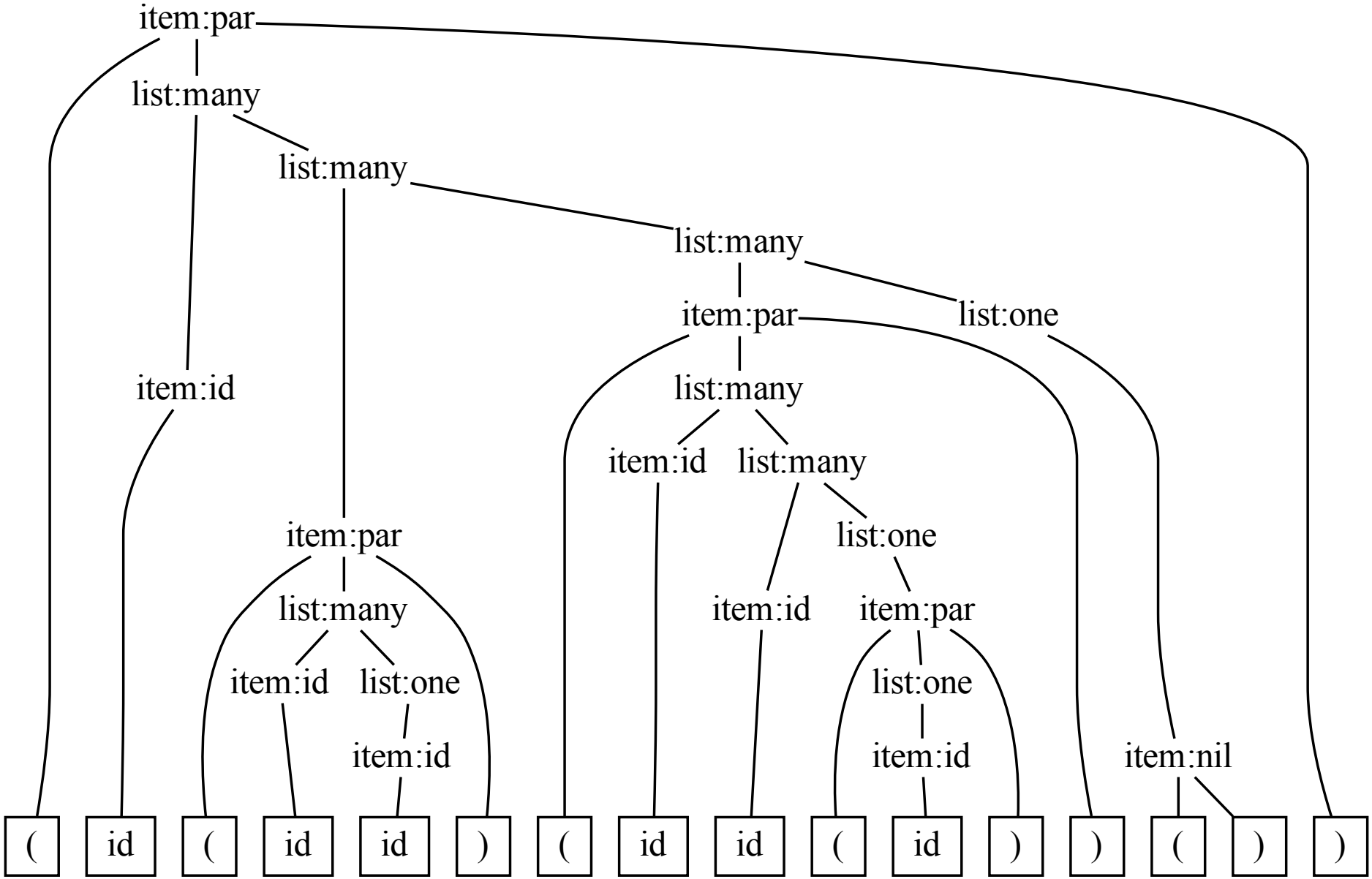
Le langage Lisp (S-expressions)

```
item = {par:} '(' list ')' |
       {nil:} '(' ')' |
       {id:} id ;
list = {many:} item list |
       {one:} item ;
```

Exercice : Trouver l'arbre syntaxique

- `(id(id id)(id id(id))())`

Le langage Lisp



Backus–Naur form (BNF)

- Métalangage pour les grammaires non contextuelles
- Utilisé pour décrire la syntaxe de ALGOL 60
- Variations possibles (étendu, augmenté, ad hoc)
- Inspire les syntaxes des générateurs d'analyseurs syntaxiques

Exemple

```
<item> ::= "(" <list> ")" | "(" ")" | <id>
<list> ::= <item> <list> | <item>
<id> ::= <alpha> | <id> <alpha>
<alpha> ::= les lettres de "a" à "z"
```

Entorses

- Gestion approximative de jetons
- Gestion approximative des blancs (et autres caractères ignorés)

Grammaire formelle

- Un ensemble fini de symboles terminaux: a, b, etc.
- Un ensemble fini de symboles non-terminaux: A, B, etc.
- Dont un est appelé **axiome** (celui de départ), souvent noté S
- Un ensemble fini de règles de production: $P \rightarrow \alpha$

Exemple

$$I \rightarrow o\ L\ c$$
$$I \rightarrow o\ c$$
$$I \rightarrow i$$
$$L \rightarrow I\ L$$
$$L \rightarrow I$$

Dérivation

- Alternative « mathématique » à un arbre syntaxique
- Séquence d'applications de règles de production, jusqu'à faire disparaître tous les non-terminaux
- Un **mot** appartient au langage si une dérivation de l'axiome au mot existe
- Dérivation **gauche** (resp. droite), on applique les règles sur le non-terminal le plus à gauche (resp. droite)

Exemple

$\underline{I} \rightarrow o\underline{L}c \rightarrow o\underline{I}Lc \rightarrow oi\underline{L}c \rightarrow oi\underline{I}c \rightarrow oiocc$

Hiérarchie des langages

Grammaires vs. expressions régulières, en théorie

Langage

- Expressions régulières (et automates) $\Rightarrow$ langages réguliers
- Grammaires non contextuelles $\Rightarrow$ langage non contextuels

Qui est le plus fort ?

- Peut-on définir tout langage régulier avec une grammaire non contextuelle ?
- Peut-on définir tout langage non contextuel avec une expression régulière ?

Grammaires vs. expressions régulières, en théorie

Langage

- Expressions régulières (et automates) $\Rightarrow$ langages réguliers
- Grammaires non contextuelles $\Rightarrow$ langage non contextuels

Qui est le plus fort ?

- Peut-on définir tout langage régulier avec une grammaire non contextuelle ? $\rightarrow$ oui
- Peut-on définir tout langage non contextuel avec une expression régulière ? $\rightarrow$ non

$\Rightarrow$ Les langages non contextuels incluent les langages réguliers

Grammaires vs. expressions régulières, en pratique

Les expressions régulières sont suffisantes pour l'analyse lexicale

- Recherche de sous-chaînes
- Séquences de jetons au fur et à mesure

Les grammaires sont nécessaires pour l'analyse syntaxique

- Fabrication d'arbres syntaxiques
- Un seul arbre complet d'un coup

Hiérarchie de Chomsky

Par Noam Chomsky (linguiste et anarchiste), 1956

- Langage régulier (type 3)
  - Productions de formes $P \to tQ$ et $P \to t$
- Langage non contextuel (type 2)
  - Productions de forme $P \to \gamma$
- Langage contextuel (type 1)
  - Productions de forme $\alpha P \beta \to \alpha \gamma \beta$
- Langage général (type 0)
  - Productions de forme $\alpha \to \beta$

où P et Q sont des productions (non-terminaux), t un jeton (terminal), α, β, γ sont des séquences (possiblement vides) de productions et de jetons

Lemme de la pompe (expressions régulières)

Lemme (de la pompe, ou de l'étoile)

Soit L un langage régulier. Il existe un entier $n > 0$ tel que tout mot w de L de longueur $|w| \geq n$ possède une factorisation $w = xyz$ telle que

$$|y| > 0$$
$$|xy| \leq n$$
$$xy^*z \subseteq L$$

Informellement

Quand un mot est suffisamment grand, il y a forcément un morceau qui peut être répété ou être supprimé.

Preuve

- Pour un langage régulier L, il existe un DFA à n états.
- Soit w un mot de L de longueur $|w| \geq n$
- Il existe un chemin $q_0 \xrightarrow{w} f$ de l'état initial q_0 à un état final f
- Un des $n+1$ premiers états de ce chemin est forcément répété (principe des tiroirs, *pigeonhole principle*)
  On note q cet état
- Le chemin $q_0 \xrightarrow{w} f$ se factorise donc en $q_0 \xrightarrow{x} q \xrightarrow{y} q \xrightarrow{z} f$
  Avec $|y| > 0$ et $|xy| \leq n$
- Donc $xy^*z \subseteq L$:
  on peut « pomper » y autant de fois qu'on veut
- CQFD

Résultats

Ne sont pas régulier

- Le langage $a^n b^n$ (avec $n \geq 0$) sur l'alphabet $\{a, b\}$
- Le langage des palindromes sur l'alphabet $\{a, b\}$
- HTML, XML, JSON
- Et plein d'autres...

Informellement

- Un automate fini ne sait pas compter

Preuve $a^n b^n$ non régulier

- Soit L le langage $a^n b^n$ (avec $n \geq 0$) sur l'alphabet $\{a, b\}$
- Supposons par l'**absurde** que L soit régulier
- Il existe n qui satisfait le lemme de la pompe
- Soit le mot $w = a^n b^n$, on observe que $|w| \geq n$
- Par le lemme de la pompe, on peut décomposer w en xyz avec $|xy| \leq n$, $|y| > 0$, et $xy^*z \subseteq L$
- Parce que $|xy| \leq n$, x et y ne sont composés que de a
- Parce que $|y| > 0$, y possède au moins un a
- Parce que $xy^*z \subseteq L$, $xz \in L$
- Or xz possède moins de a que de b, c'est une **contradiction**
- CQFD

Conception de grammaires

Exercice : Conception de grammaires

Soit l'alphabet { '0', '1' }

Langage où chaque '1' est forcément suivi d'un '0'

- Exemples : `001001010` OK ; `1101` pas OK
- Le langage est-il régulier ?

Langage des palindromes

- Exemples : `011010110` OK ; `1101` pas OK
- Le langage est-il régulier ?

Exercice : Conception de grammaires

Soit l'alphabet { '0', '1' }

Langage où chaque '1' est forcément suivi d'un '0'

- Exemples : `001001010` OK ; `1101` pas OK

```
s = '0' s | '1' '0' s | ;
```

- Le langage est-il régulier ?

Langage des palindromes

- Exemples : `011010110` OK ; `1101` pas OK

```
s = '0' s '0' | '1' s '1' | '1' | '0' | ;
```

- Le langage est-il régulier ?

Langage des grammaires

Question

- Écrire la grammaire des grammaires
- Alphabet (jetons) : `id`, `altid`, `'='`, `'|'`, `';'` et `str`

Langage des grammaires

Question

- Écrire la grammaire des grammaires
- Alphabet (jetons) : `id`, `altid`, `'='`, `'|'`, `';'` et `str`

Réponse

```
prods = {many:} prods prod  | {one:} prod ;
prod = id '=' alts ';' ;
alts = {many:} alts '|' alt | {one:} alt ;
alt = altid atoms | atoms ;
atoms = {many:} atoms atom | {none:} ;
atom = {id:} id | {str:} str ;
```

Langage des grammaires (suite)

Question

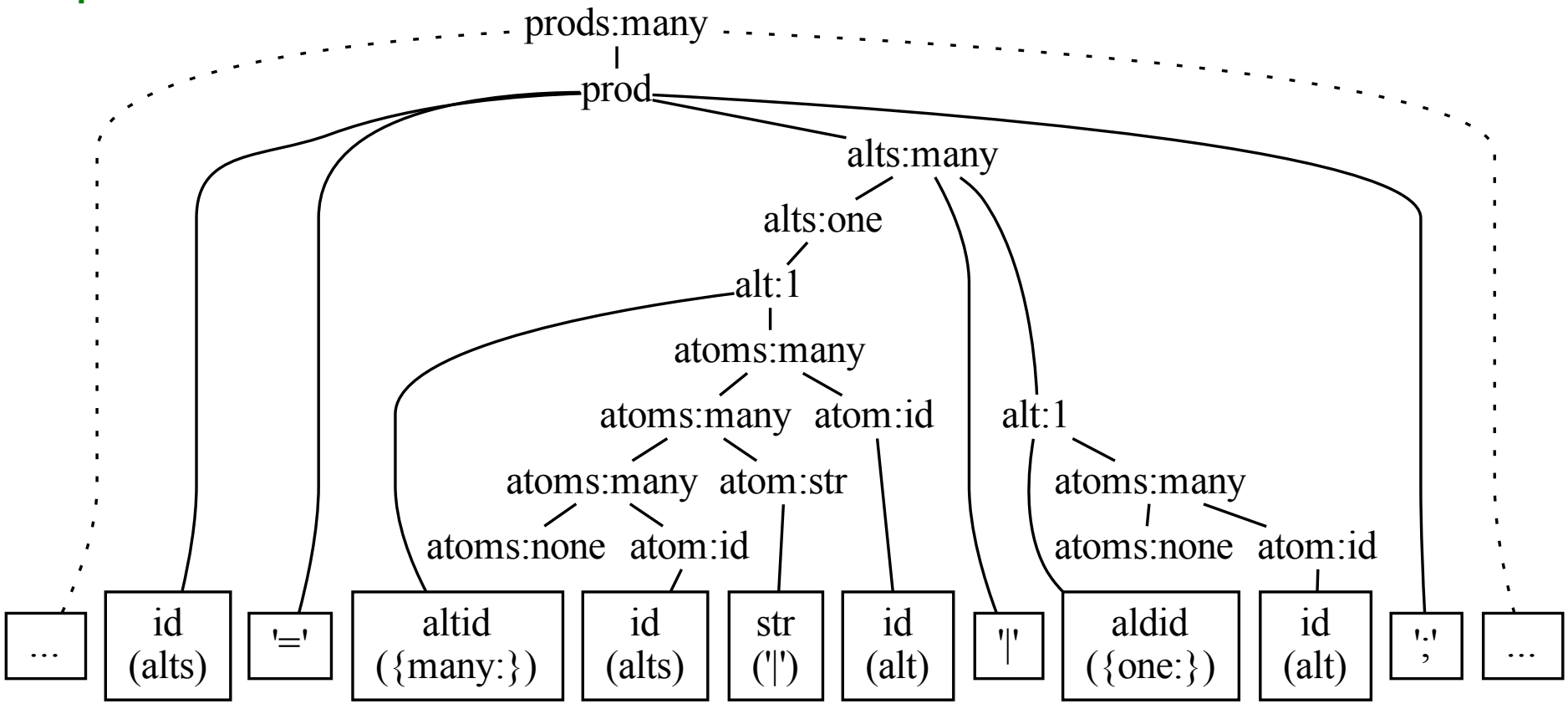
- Écrire l'arbre syntaxique de la grammaire

Langage des grammaires (suite)

Question

- Écrire l'arbre syntaxique de la grammaire

Réponse



Ambiguités

Langage des expressions arithmétiques (calc)

```
exp = {add:} exp '+' exp |
      {sub:} exp '-' exp |
      {mul:} exp '*' exp |
      {int:} int |
      {par:} '(' exp ')' ;
```

Exercice : Trouver l'arbre syntaxique

- `5 + 4 * 2`
- `5 - 4 - 2`
- `5 + 4 + 2`

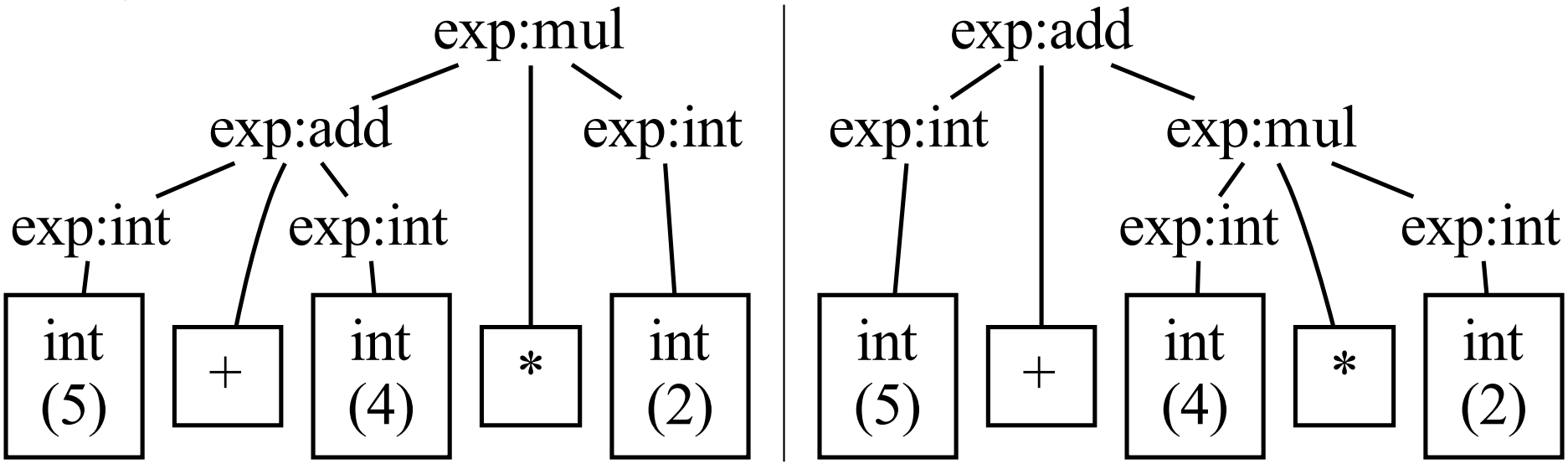
Ambiguïté

Grammaire ambiguë

- Plusieurs arbres syntaxiques pour une même mot

Exemple

- `5 + 4 * 2`



Ambiguïté

Problème d'arbre

- La question n'est pas sur l'appartenance au langage
- Mais sur obtenir un arbre unique
  Ou une dérivation gauche (resp. droite) unique

Détection d'ambiguïté

- Problème non décidable (mais on se débrouille)

Solutions

- Récrire la grammaire
- Priorités explicites (grammaire augmentée)
- Changer le langage

Fait amusant

- Les grammaires de la plupart des spécifications de langages sont ambiguës

Ambiguïté : Récrire

```
exp = {add:} exp '+' factor |
      {sub:} exp '-' factor |
      {factor:} factor ;
factor = {mul:} factor '*' term |
         {term:} term ;
term = {int:} int |
       {par:} '(' exp ')' ;
```

Exercice : Trouver l'arbre syntaxique

- `5 + 4 * 2`
- `(5 + 4) * 2`
- `5 + (4 * 2)`

Ambiguïté : Priorités explicites

```
exp = {add:} exp '+' exp |
      {sub:} exp '-' exp |
      {mul:} exp '*' exp |
      {int:} int |
      {par:} '(' exp ')' ;
   Priority
      Left mul;
      Left add, sub;
```

Attention

- Priorités explicites ne marchent que pour les cas simples et dépendent de l'outil

Le langage des conditions (if)

```
s = {if:} 'if' '(' e ')' s |
    {nop:} 'nop' ';' ;
e = 'true' | 'false' ;
```

Exercice : ajouter `else` (optionnel)

Le langage des conditions (if)

```
s = {if:} 'if' '(' e ')' s |
    {nop:} 'nop' ';' ;
e = 'true' | 'false' ;
```

Exercice : ajouter `else` (optionnel)

```
s = ... |
    {ifelse:} 'if' '(' e ')' s 'else' s ;
```

Exercice : trouver l'arbre syntaxique

- `if (true) if (true) nop; else nop;`

Dangling else

- `if (true) if (true) nop; else nop;`

Problème classique d'ambiguïté

À quel `if` se rattache le `else` pendant (*dangling*) ?

- `if (true) { if (true) nop; else nop; }`
- `if (true) { if (true) nop; } else nop;`

Solutions

- Changer le langage
  - Forcer les accolades (ou équivalent)
  - Forcer le `else` (ou équivalent)
- Ajouter une règle grammaticale supplémentaire
  - Le `else` se rattache au `if` le plus proche
  - Mais il faut pouvoir la spécifier dans l'outil...

Ambiguïté : Réécrire

Réécrire la grammaire pour enlever l'ambiguïté

Ambiguïté : Réécrire

Réécrire la grammaire pour enlever l'ambiguïté

```
s = {if:} 'if' '(' e ')' s |
    {ifelse:} 'if' '(' e ')' s_else 'else' s |
    {nop:} 'nop' ';' ;
s_else =
    {ifelse:} 'if' '(' e ')' s_else 'else' s_else |
    {nop:} 'nop' ';' ;
e = 'true' | 'false' ;
```

Problème

- C'est horrible !

Priorités explicites et implicites

En SableCC4

```
s = {if:} 'if' '(' e ')' s |
    {ifelse:} 'if' '(' e ')' s 'else' s |
    {nop:} 'nop' ';' ;
        Priority
        Right if, ifelse;
e = 'true' | 'false' ;
```

Autres outils

- Appliquent une priorité implicitement, parfois avec un warning
- Ça peut causer de mauvaises surprises :(

Générateur d'analyseur syntaxique

Utilisation d'un générateur d'analyse syntaxique

Entrée

- Une grammaire

Fichiers générées

- L'analyseur syntaxique (parser)
- Les classes de la structure syntaxique du langage
- Des classes de manipulation de la structure (visiteurs)

Classes de la structure syntaxique

Hiérarchie de classe

- Une classe par production
- Une sous-classe par alternative si plusieurs

Accesseurs (navigation)

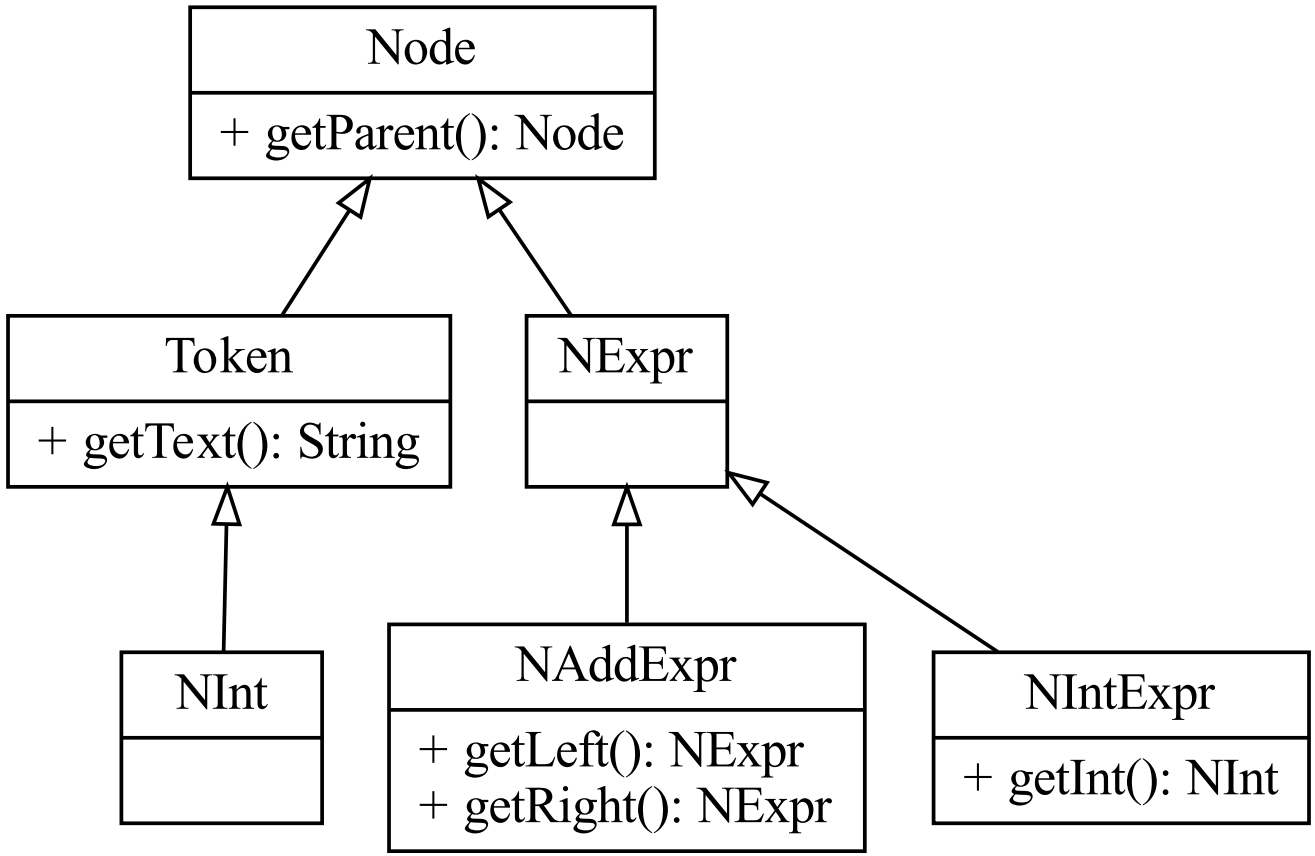
- Nœud parent
- Nœud fils : chacun des éléments (production ou jeton)

Étiquetage

- Alternative « `{bla:}` » $\rightarrow$ nom des classes
- Terme « `[bla:]` » $\rightarrow$ nom des accesseurs

Classes de la structure syntaxique

```
expr = ... |
       {add:} [left:]expr '+' [right:]expr |
       {int:} int;
```

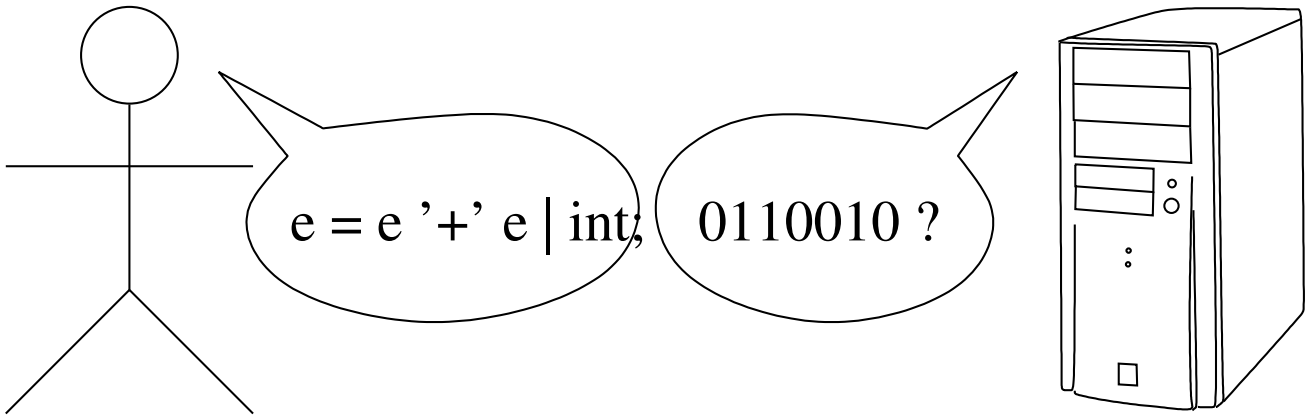


Comment utiliser l'arbre syntaxique ?

La prochaine fois !

- Venez avec votre portable (chargé)
- Installez SableCC4beta2 (si c'est pas déjà fait)
- Maîtrisez un éditeur de texte pour Java

Algorithmes



Plusieurs familles d'algorithmes

- LL, LR, PEG, descendent récursif, etc.
- Sur des classes de grammaires parfois disjointes
- Pour les meilleurs algos, il y a des automates :)
- Une prochaine fois...