Chapitre 5
Grammaires non contextuelles et arbres syntaxiques

Jean Privat

Université du Québec à Montréal

INF5000 — Théorie et construction des compilateurs
Automne 2011

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 humaine

Exemple

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

Questions

- La chaîne appartient-elle au langage ?
- Si oui, quelle est la structure (l'arbre) syntaxique ?

Grammaire non contextuelle

Jeton (terminal)

- Élément de l'alphabet du langage
- Seul le type du jeton est considéré et non le texte
- Distingués en bleu dans la suite

Production (non-terminal)

- Variable désignant un ensemble d'alternatives
- Une production spéciale, celle de départ (racine)

Alternative

- Une séquence de jetons et de productions

Le langage des formes

```
Grammar formes:
Lexer
   nombre = ('0'..'9')+;
   Token nombre;
   Ignored ' ';
Parser
   forme = {cercle:} 'centre' point 'rayon' long |
           {segment:} point '--' point ;
   point = '(' long ',' long ')' ;
   long = nombre unite ;
   unite = 'cm' | 'mm' | 'pt' | 'px' ;
```

Le langage des formes

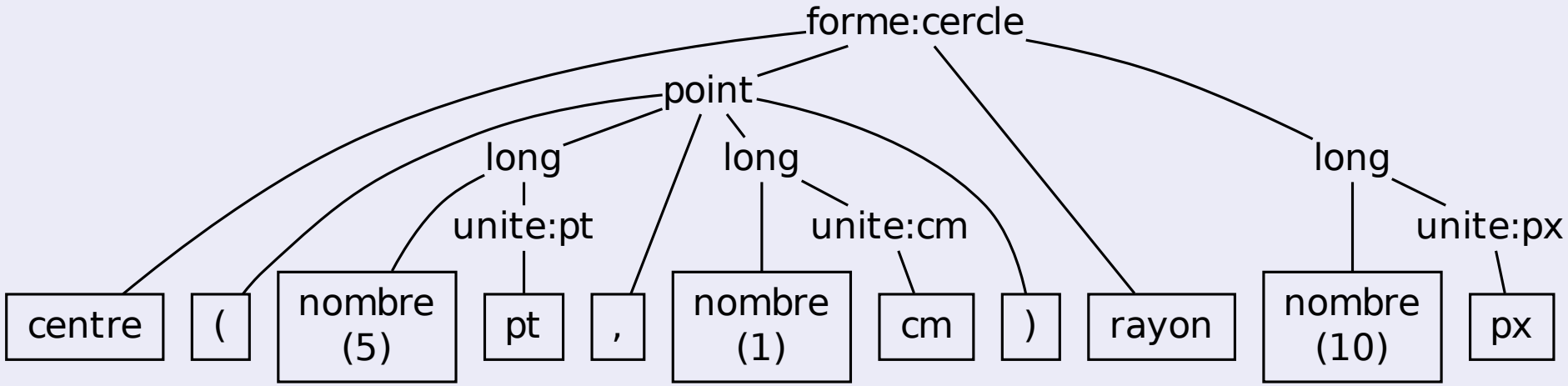
Exercice : Trouver l'arbre syntaxique

- *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

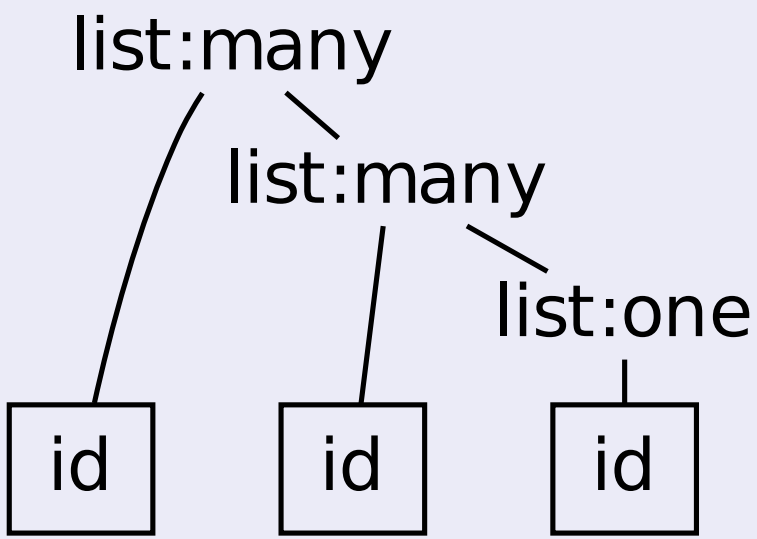
- *id id id*

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

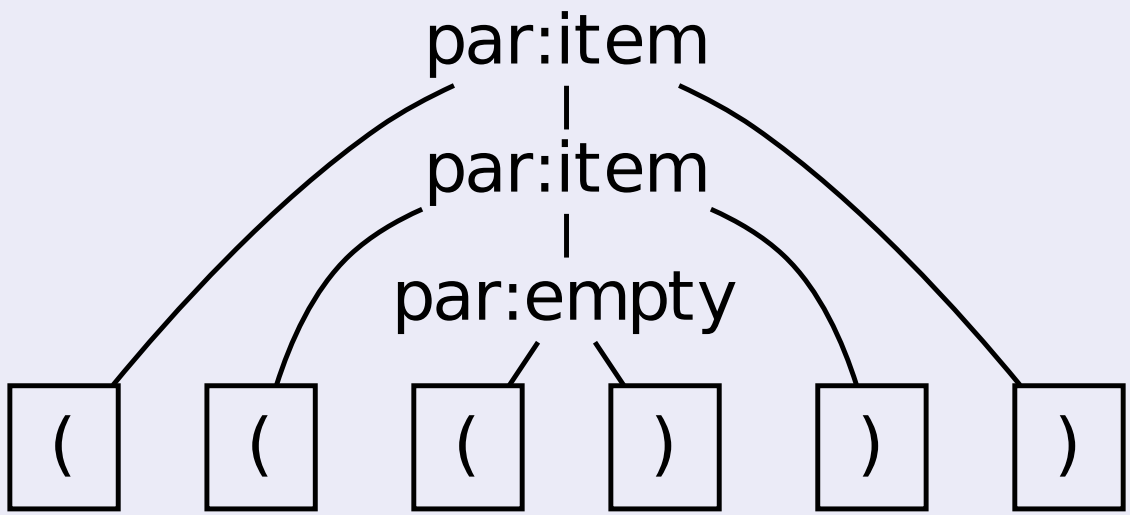
- *((()))*

Le langage des parenthèses

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

Exercice : Trouver l'arbre syntaxique

- *((()))*



Grammaires vs. expressions régulières, en théorie

Langage

- Expressions régulières (et automates) $\Rightarrow$ langage régulier
- Grammaires non contextuelles $\Rightarrow$ langage non contextuel

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) $\Rightarrow$ langage régulier
- Grammaires non contextuelles $\Rightarrow$ langage non contextuel

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

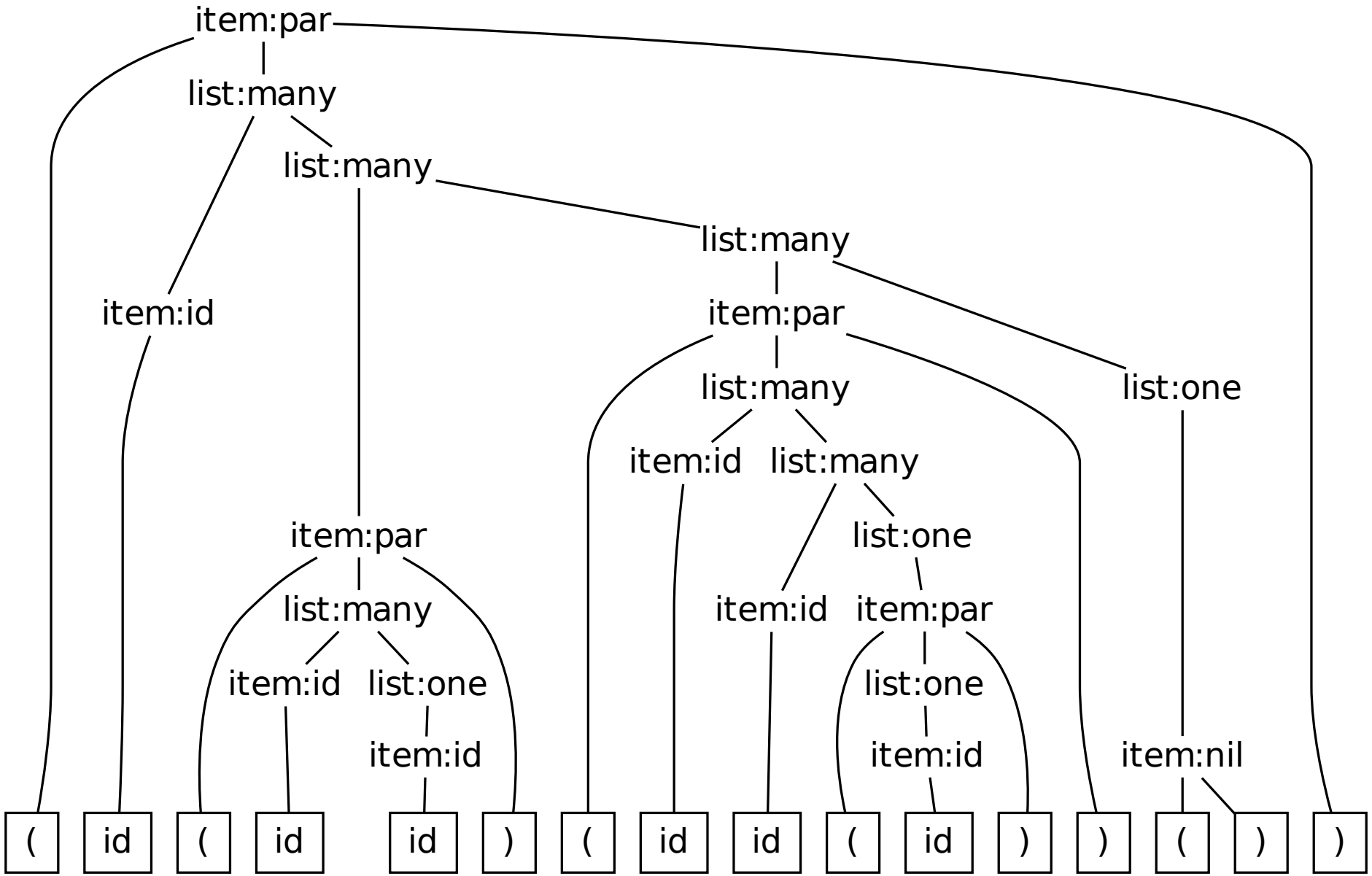
Le langage Lisp

```
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



Le langage des expressions arithmétiques

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

Exercice : Trouver l'arbre syntaxique

- $5 + 4 * 2$
- $5 - 4 - 2$
- $5 + 4 + 2$

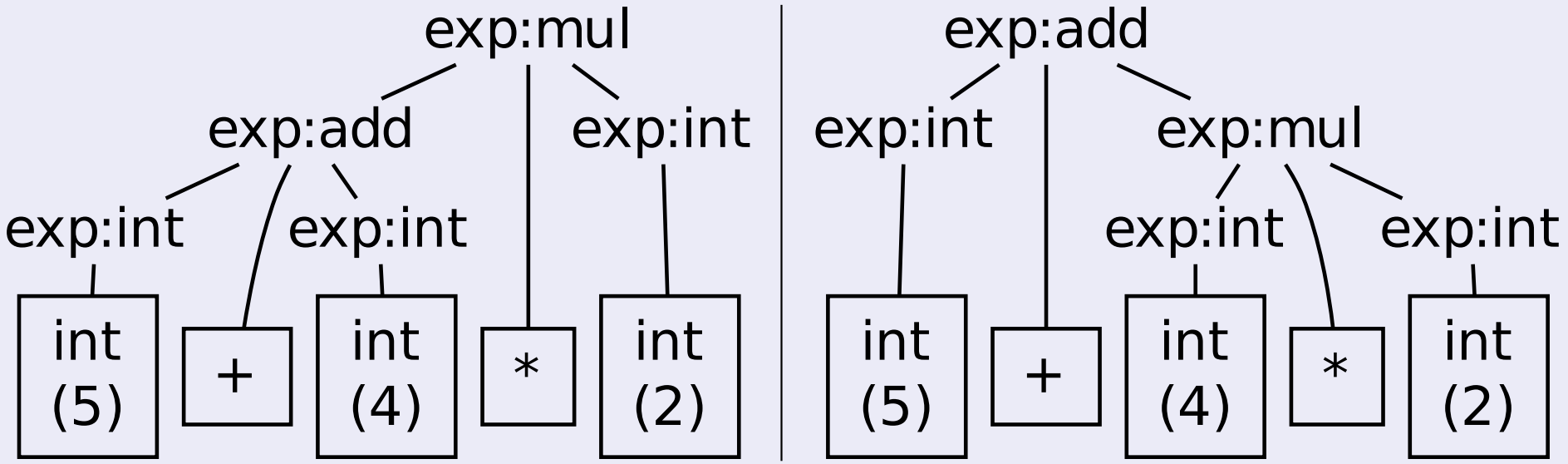
Ambiguïté

Grammaire ambiguë

- Plusieurs arbres syntaxiques pour une même phrase

Exemple

- *5 + 4 * 2*



Ambiguïté

Problème d'arbre

- La question n'est pas sur l'appartenance au langage
- Mais sur obtenir un arbre unique

Détection d'ambiguïté

- Problème non décidable (mais on se débrouille)

Solutions

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

Ambiguïté : Récrire

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

Attention

- Priorités explicites ne marchent que pour les cas simples

Exercice

Question : Concevoir les grammaires

- Soit l'alphabet { '0', '1' }
- Langage où chaque '1' est forcément suivi d'un '0'
  Exemples : *001001010* OK ; *1101* pas OK

- Langage des palindromes
  Exemples : *011010110* OK ; *1101* pas OK

Exercice

Question : Concevoir les 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 | ;

- Langage des palindromes
  Exemples : *011010110* OK ; *1101* pas OK
  S = '0' S '0' | '1' S '1' | '1' | '0' | ;