Langages et paradigmes

Olivier Lecarme

Master d'Informatique, première année

2007–2008

Troisième partie III

Aspects lexico-syntaxiques

Expressions régulières et lexèmes

- Mécanisme de choix pour décrire des lexèmes compliqués.
- Servent aussi de donnée aux générateurs d'analyseurs lexicaux.
- Pas très faciles à construire, et jamais utilisées dans la description des langages, même dans les normes.

expressions régulières et lexèmes

- Servent à décrire les lexèmes, mais non pas tous les aspects lexicaux des langages :
  - majuscules et minuscules (Pascal, Ada)
  - passages à la ligne (anciennes versions de Fortran, point-virgule implicite en Icon)
  - commentaires ordinaires
  - commentaires actifs (pragmas d'Ada, commandes au compilateur)
  - décalages significatifs (Occam, Haskell, Python)

expressions régulières et lexèmes

- Ne fonctionnent bien que si le langage sépare proprement les aspects lexicaux des aspects syntaxiques :
  - en Fortran ou PL/I, les mots-clés sont reconnus par leur position et par conséquent non réservés
  - en Algol 60 ou 68, les mots-clés ont une écriture spécifique
  - dans la plupart des autres langages, les mots-clés sont des identificateurs réservés (y compris en Cobol qui en a plus de 400 !)

Grammaires algébriques

- Généralités
  - Nécessaires pour décrire des structures emboîtées.
  - Mécanisme de génération assez facile à utiliser pour la description.
  - Toutes les définitions de langages utilisent des variantes de la *Forme Normale de Backus*, utilisée initialement pour Algol 60.

généralités

- Les extensions simplifient l'écriture des constructions les plus courantes :
  - `[ ... ]` pour fragment facultatif
  - `{ ... }` pour fragment répété
  - en général rien d'autre, ce qui oblige à des répétitions
  - parfois on combine grammaires algébriques et expressions régulières en parties droites

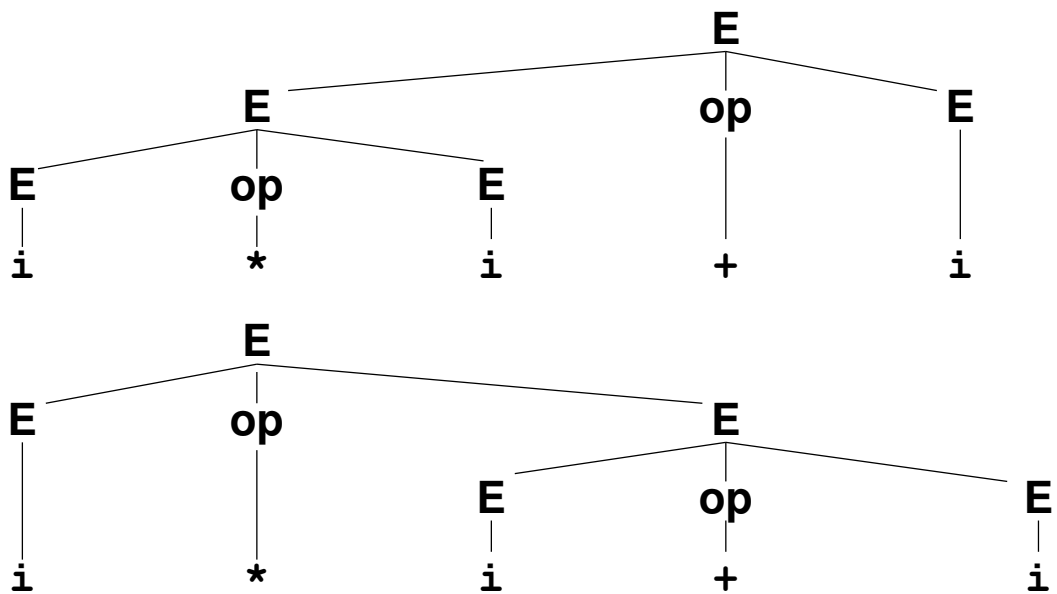
Arbres de dérivation

Soit la grammaire algébrique :

```
E = E op E
  = i
  = ( E )
op = + | - | * | /
```

arbres de dérivation

La phrase `i * i + i` admet deux arbres de dérivation, elle est donc *ambiguë* :



arbres de dérivation

- L'une des dérivations possibles est :
  `E` $\Rightarrow$ `E op E` $\Rightarrow$ `E op i` $\Rightarrow$ `E + i` $\Rightarrow$ `E op i + i` $\Rightarrow$ `E * i + i` $\Rightarrow$ `i * i + i`
- Une autre est :
  `E` $\Rightarrow$ `E op E` $\Rightarrow$ `E op E op E` $\Rightarrow$ `i op E op E` $\Rightarrow$ `i * E op E` $\Rightarrow$ `i * i op E` $\Rightarrow$ `i * i + E` $\Rightarrow$ `i * i + i`

arbres de dérivation

L'ambiguïté peut être levée par des règles de priorité ou d'associativité. Également par récriture de la grammaire :

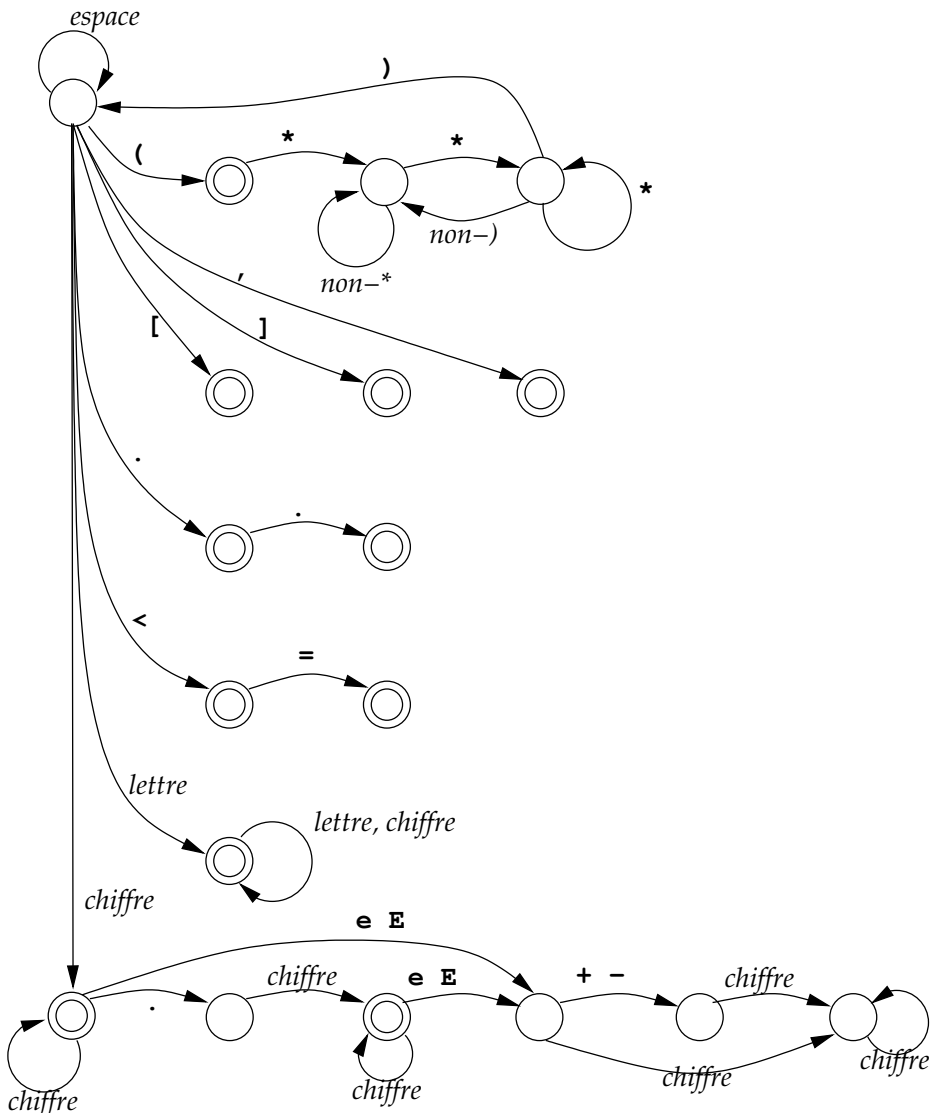
```
E = E opadd T
  = T
T = T opmul F
  = F
F = i
  = ( E )
opadd = + | -
opmul = * | /
```

Analyse lexicale

- Généralités
  - L'analyseur syntaxique appelle l'analyseur lexical, qui lui renvoie un *lexème*.
  - À chaque appel, l'analyseur lexical doit reprendre son algorithme au début.
  - Il accepte normalement le lexème le plus long possible.
  - Le fonctionnement général est celui d'un automate d'états finis très simple, dans les cas simples.

généralités

- La programmation peut se faire par un grand énoncé **cas**, ou par un automate général utilisant une matrice de transition.



Aspects pratiques

La théorie donne une vision trop simpliste :

- Les mots-clés doivent être cherchés dans une table, pour éviter de construire un automate beaucoup trop grand.
- La recherche du lexème le plus long peut conduire à avancer trop loin dans le texte :
  - `1..10` ou `1.10` en Pascal
  - `DO 5 I = 1.25` ou `DO 5 I = 1,25` en Fortran
  - certains opérateurs composites très compliqués en Algol 68

aspects pratiques

- Les commentaires doivent être ignorés par l'analyseur lexical.
- Les « commentaires significatifs » doivent être traités spécialement :
  - mettre les vérifications en fonction
  - mettre certaines améliorations en fonction
  - proposer de mettre une variable dans un registre
  - cette procédure n'est jamais récursive

Commentaires

Très différents suivant les langages :

- `C` en colonne 6 en Fortran IV
- **comment** `...;` en Algol 60
- **procedure** `Un(a) deux : (b)` en Algol 60
- **end** `blabla;` en Algol 60
- **com** `...` **com** en Algol 68
- `{ ... }` ou `(* ... *)` en Pascal
- `/* ... */` en PL/I et C
- `-- ...` en Ada (`//` en C^{++}, `;` en Lisp, etc.)
- etc.

Analyse syntaxique

- Généralités
  - Tous les analyseurs courants procèdent de gauche à droite de manière déterministe.
  - L'analyse descendante ou prédictive prédit la prochaine étape d'une dérivation gauche au vu du prochain lexème.

généralités

- L'analyse ascendante ou déductive rassemble tous les composants d'une partie droite avant d'effectuer une réduction, suivant donc de manière rétrograde les étapes d'une dérivation droite.
- L'analyse ascendante n'essaie pas de prédire, ce qui lui permet d'accepter des grammaires plus générales.

Catégories de grammaires

- LR(0) non LL :

```
A = B b
B = B a | a
```

- LALR(1) non SLR :

```
A = b B b | B c | a b
B = a
```

catégories de grammaires

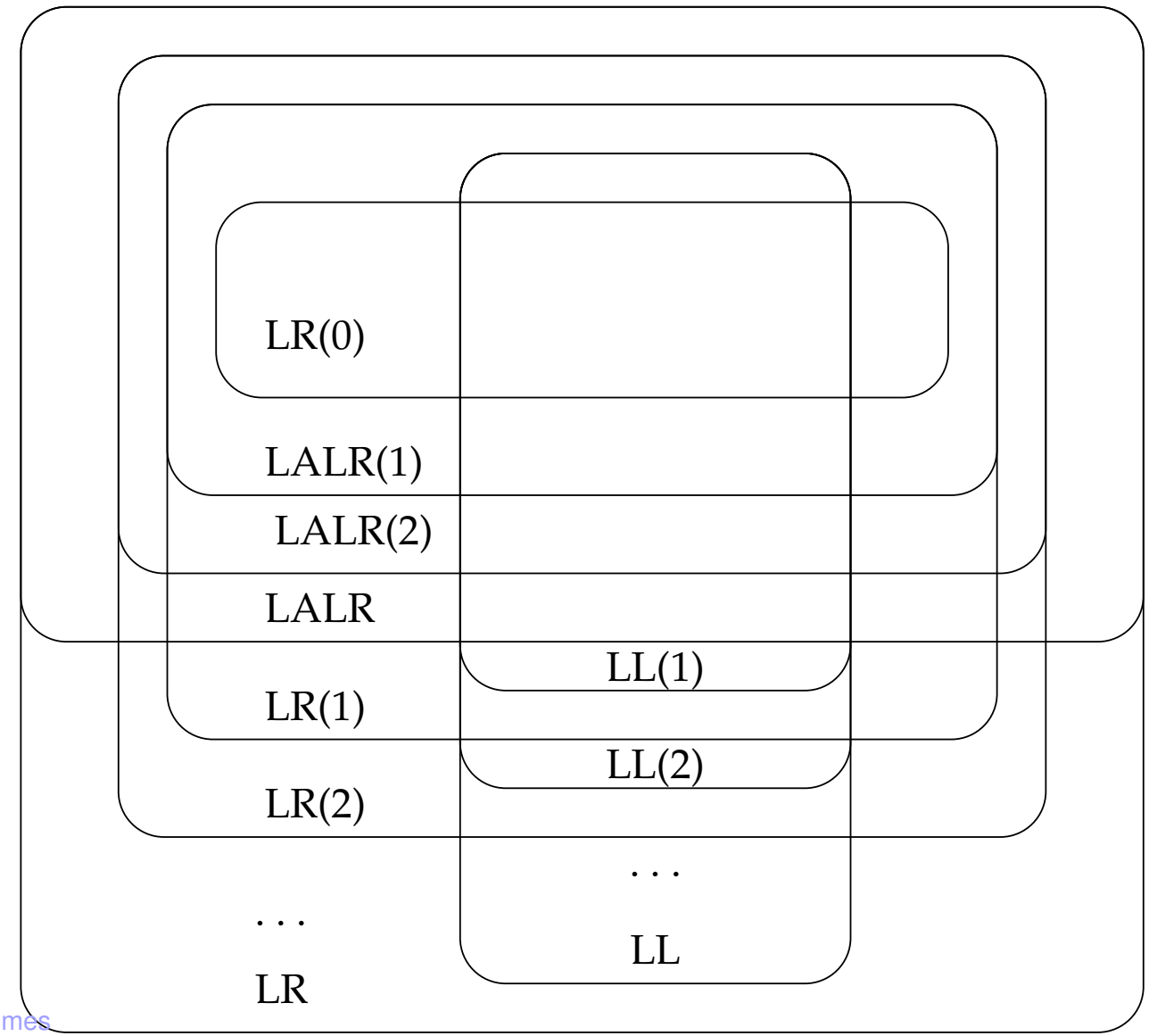
- LR(1) non LALR :

```
A = a C a | b C b | a D b | b D a
C = c
D = c
```

- non ambiguë mais non LR :

```
A = a A a | ν
```

catégories de grammaires



Influence sur les langages

- Certaines définitions de langages sont prévues pour un type particulier d'analyse :
  - PL/360 (Wirth) est un langage de précédence simple.
  - Certaines constructions de beaucoup de langages sont prévues pour faciliter l'analyse LL(1), en particulier le mot-clé en début de chaque type d'énoncé.

influence sur les langages

- En revanche, certaines constructions nécessitent des entorses :
  - appel de procédure et affectation en Pascal
  - déclarations de fonctions en C
  - modèles des types en Algol 68
  - point-virgule facultatif en Icon