Grammaires attribuées

Mirabelle Nebut

Bureau 332 - M3
mirabelle.nebut at lifl.fr

2011-2012

Jusqu'à présent, on a vu...

L'analyse lexicale :

- but : découper le flot de caractères en tokens
- basée sur : expressions régulières, AFD

L'analyse syntaxique :

- reconnaire la structure du flot de token
- construite une dérivation, un arbre syntaxique
- basée sur : grammaires algébriques

En termes de TP...

Analyse lexicale : TP2

- le planning d'entrée n'est pas lexicalement correct : levée de `ScannerException`
- il l'est : découpage en token

Analyse syntaxique : TP3

- le planning d'entrée n'est pas syntaxiquement correct : levée de `ParserException`
- il l'est : ... rien

Rien ?

Maintenant il faut produire le fichier Latex !

Quoi d'autre ?

Avant de produire le fichier Latex :

- vérifier que la date a un sens
- vérifier que le créneau horaire a un sens

Ces vérifications ne sont pas du ressort des analyses lexicale et syntaxique.

Plus généralement

Sur un programme syntaxiquement correct...

Une analyse sémantique qui vérifie par exemple :

- que le programme est correctement typé ;
- qu'il n'y a pas de double déclaration ;
- que toute variable est déclarée avant son utilisation ;
- etc.

Puis une ou plusieurs actions sémantiques :

- optimisation
- production de code
- calculs divers
- etc

Comment faire ça ?

Dans le TP1 : analyse dirigée par les délimiteurs =

- on force une structuration par ligne
- on lit une ligne, et on mélange reconnaissance et action
- l'action est associée à la ligne reconnue

Dans la version outillée : analyse dirigée par la syntaxe =

- basée sur une grammaire algébrique
- on associe les actions aux productions de la grammaire
- $\Rightarrow$ grammaire attribuée

Principe des grammaires attribuées

Les actions au sens large = des calculs sur des données

Ex : calcul = générer du Latex, données = nom master, soutenance, etc

Ex : calcul = vérifier le créneau, données = bornes de l'intervalle

$\Rightarrow$ on enrichit les grammaires algébriques par :

- des données : des attributs
- des calculs sur ces données : des actions

Exemple

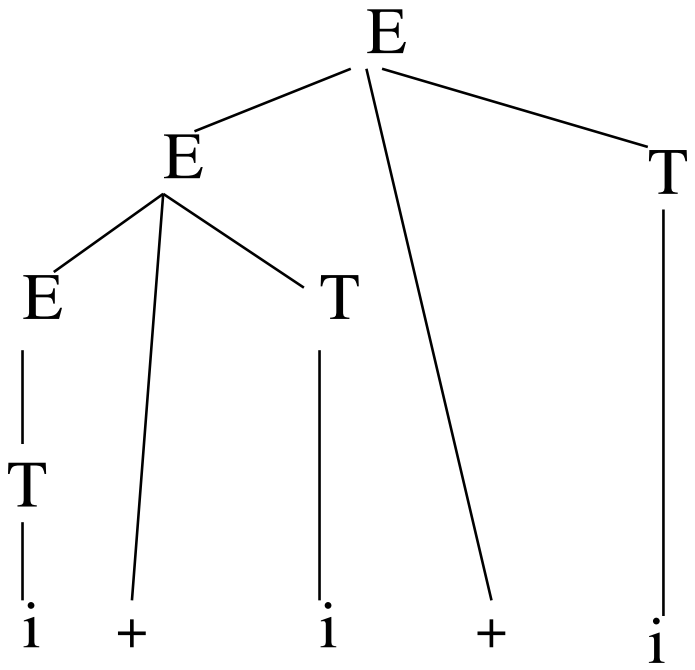
Calculer la valeur d'une expression arithmétique.

$$E \rightarrow E + T \mid T$$
$$T \rightarrow (E) \mid \texttt{i}$$

`i` vaut 3, valeur de `i + i + i` ?

Arbre syntaxique pour `i + i + i`

Très utile pour concevoir une attribution.



Les questions à se poser pour démarrer

- Quelles données veut-on calculer $\Rightarrow$ quels attributs ?
- Comment les calculer $\Rightarrow$ quelles actions ?

Données - attributs

Les attributs sont associés aux symboles de la grammaire

Symboles = terminaux et non-terminaux

Exemple : Données = valeur

- de l'expression elle-même : E ;
- de ses sous-expressions : T, `i`.

Attributs dans l'exemple

attribut *val* de type entier associé au non-terminal E, noté $E.val$

attribut *val* de type entier associé au non-terminal T, noté $T.val$

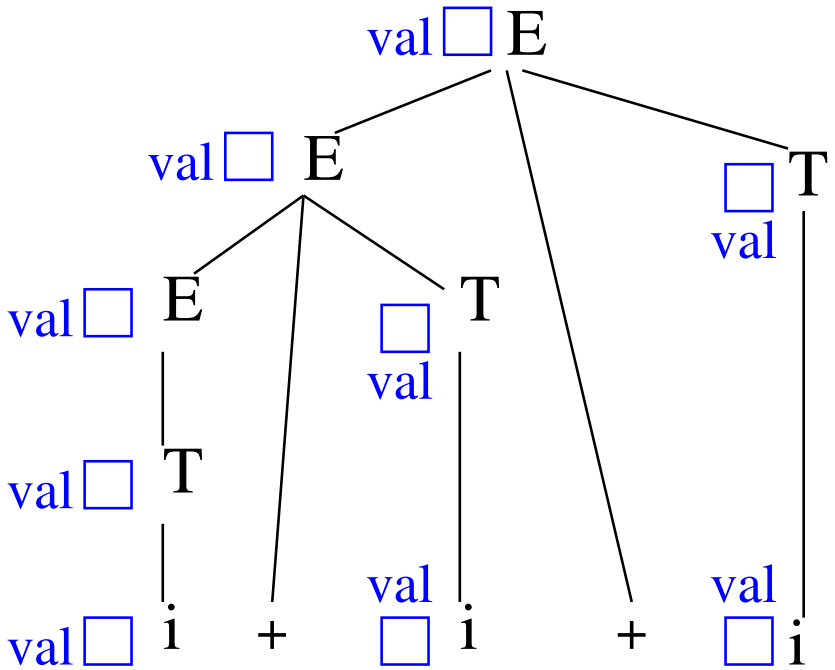
attribut *val* de type entier associé au terminal `i`, noté `i`.*val*

NB : similitude avec les attributs d'un objet en POO

Arbre décoré

Nœuds de l'arbre syntaxique décorés avec la valeur des attributs.

Autant de ≪ cases ≫ que les actions devront remplir.



Calculs - actions

Additions, par ex pour $E \rightarrow E + T$:

$$E.val = E.val + T.val$$

On associe une action à la production en distinguant les différentes occurrences de E :

$$E \rightarrow E + T \qquad \{ E_0.val = E_1.val + T.val \}$$

≪Passage de valeurs≫ :

$$E \rightarrow T \qquad \{ E.val = T.val \}$$

Remarque

Qui fixe la valeur de l'attribut *val* du terminal `i` ?

$\Rightarrow$ elle est initialisée par l'analyseur lexical.

Grammaire attribuée finale

$$
\begin{array}{ll}
E \rightarrow E + T & \{\ E_0.val = E_1.val + T.val\ \} \\
E \rightarrow T & \{\ E.val = T.val\ \} \\
T \rightarrow (\ E\ ) & \{\ T.val = E.val\ \} \\
T \rightarrow \mathtt{i} & \{\ T.val = \mathtt{i}.val\ \}
\end{array}
$$

Et si la grammaire était ambiguë ?

On obtiendrait la même valeur pour les 2 arbres admis par

`i + i + i`...

... mais considérons la grammaire :

$$E \rightarrow E + E \mid E * E \mid \texttt{i}$$

Le mot `i + i * i` a deux significations, E.`val` a 2 valeurs possibles.

$\Rightarrow$ attribuer une grammaire ambiguë n'a aucun sens pratique.

Comment $\neq$ quand

Cette grammaire spécifie comment calculer des valeurs associées à ses symboles.

Mais elle ne dit pas quand effectuer ces calculs. . .

. . . ni dans quel ordre effectuer les actions.

Grammaire attribuée = formalisme de spécification, pas d'exécution.

Au sens strict, actions = équations, pas affectations.

Pour rajouter une sémantique opérationnelle, on introduit 2 types d'attributs : les synthétisés et les hérités.

Sur l'exemple

$$
\begin{array}{ll}
E \rightarrow E + T & \{ E_0.val = E_1.val + T.val \} \\
E \rightarrow T & \{ E.val = T.val \} \\
T \rightarrow (E) & \{ T.val = E.val \} \\
T \rightarrow \texttt{i} & \{ T.val = \texttt{i}.val \}
\end{array}
$$

Ces actions respectent le schéma :

```
val_att_gauche_prod = f (val_att_droite_prod)
```

Pour les symboles non-terminaux :

- en partie gauche de production : occurrences de définition des attributs
- en partie droite de production : occurrences d'utilisation des attributs

Définition

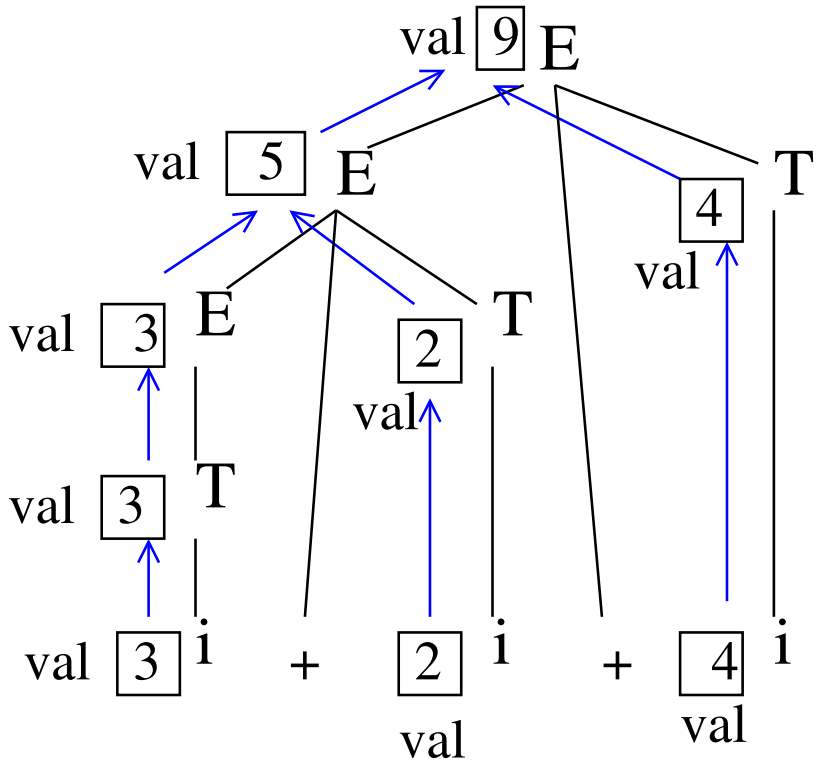
Le schéma : `val_att_gauche_prod = f (val_att_droite_prod)` correspond à un attribut synthétisé :

- défini quand associé à un non-terminal en partie gauche de production ;
- utilisé quand associé à un non-terminal ou terminal en partie droite de production.

Exception : attribut du terminal `i`.*val* :

- valeur fixée par analyseur lexical ;
- n'apparaît qu'en partie droite (utilisation seule) ;
- dit synthétisé par convention.

Sur l'arbre syntaxique



Les attributs synthétisés « remontent » dans l'arbre décoré.

Formellement

L'attribut $X.a$ est synthétisé si X apparaît en partie gauche de production et si la valeur de $X.a$ est calculée en fonction de la valeur d'attributs associés à des symboles apparaissant en partie droite de production.

$X \to X_1 X_2 \dots X_n \{ X.a = f(X_1.x_1, \dots, X_n.x_n) \} \qquad X_i \in V_T \cup V_N$

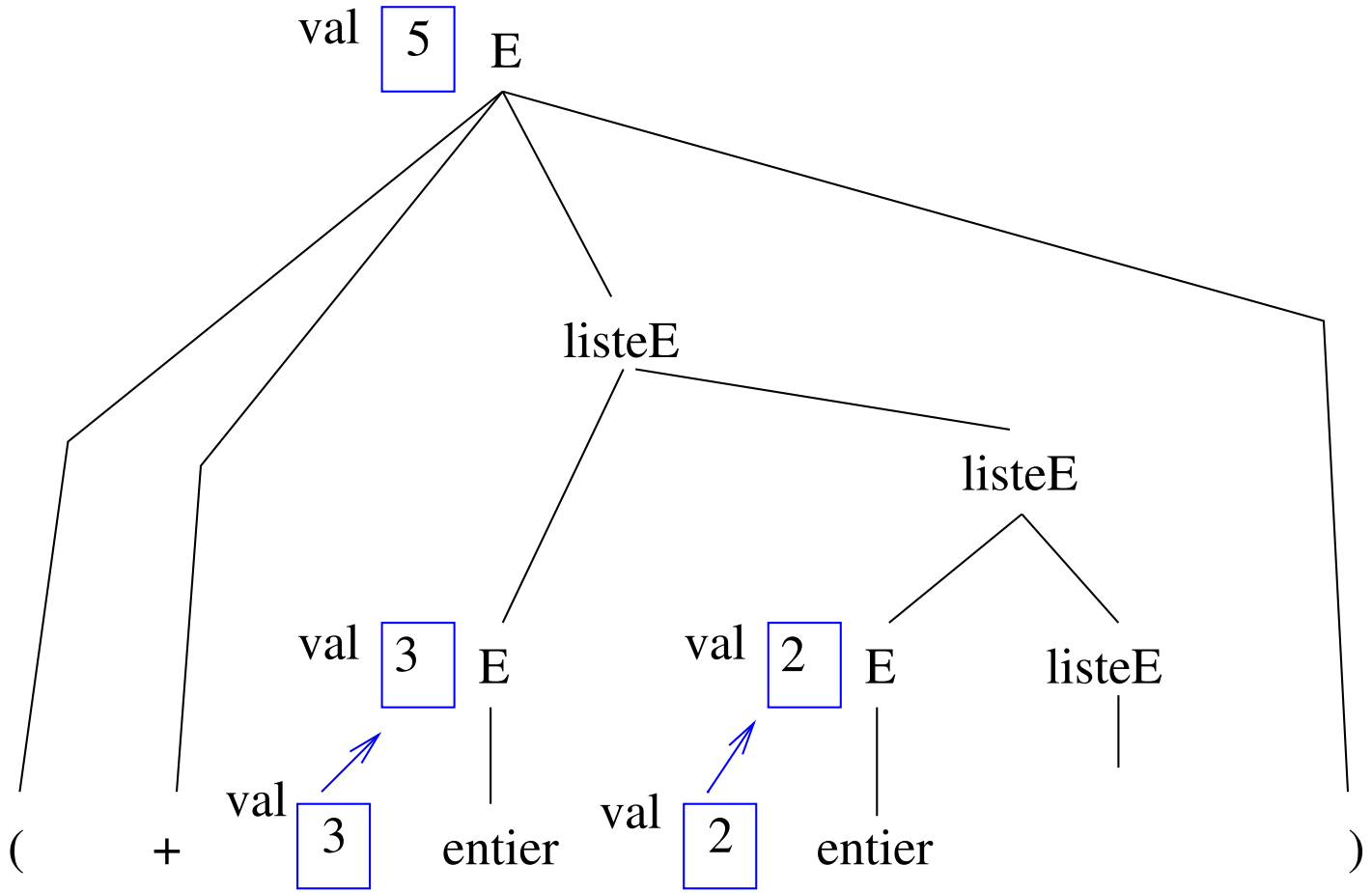
Toute production doit calculer la valeur des attributs synthétisés de son membre gauche (via une action).

Autre exemple : expressions préfixées

$$E \rightarrow \texttt{entier} \mid \texttt{(} + \textit{listeE} \texttt{)} \mid \texttt{(} * \textit{listeE} \texttt{)}$$
$$\textit{listeE} \rightarrow E\ \textit{listeE} \mid \epsilon$$

Calcul de la valeur de l'expression ?

Attribution partielle



Attribution partielle

Attributs synthétisés :

- *val* de type `entier` associé au terminal `entier` ;
- *val* de type `entier` associé au non-terminal E ;

$E \rightarrow$ `entier` $\quad \{ E.val = entier.val \}$
$E \rightarrow$ `(` `+` *listeE* `)` $\quad \{ E.val = ? \}$
$E \rightarrow$ `(` `*` *listeE* `)` $\quad \{ E.val = ? \}$

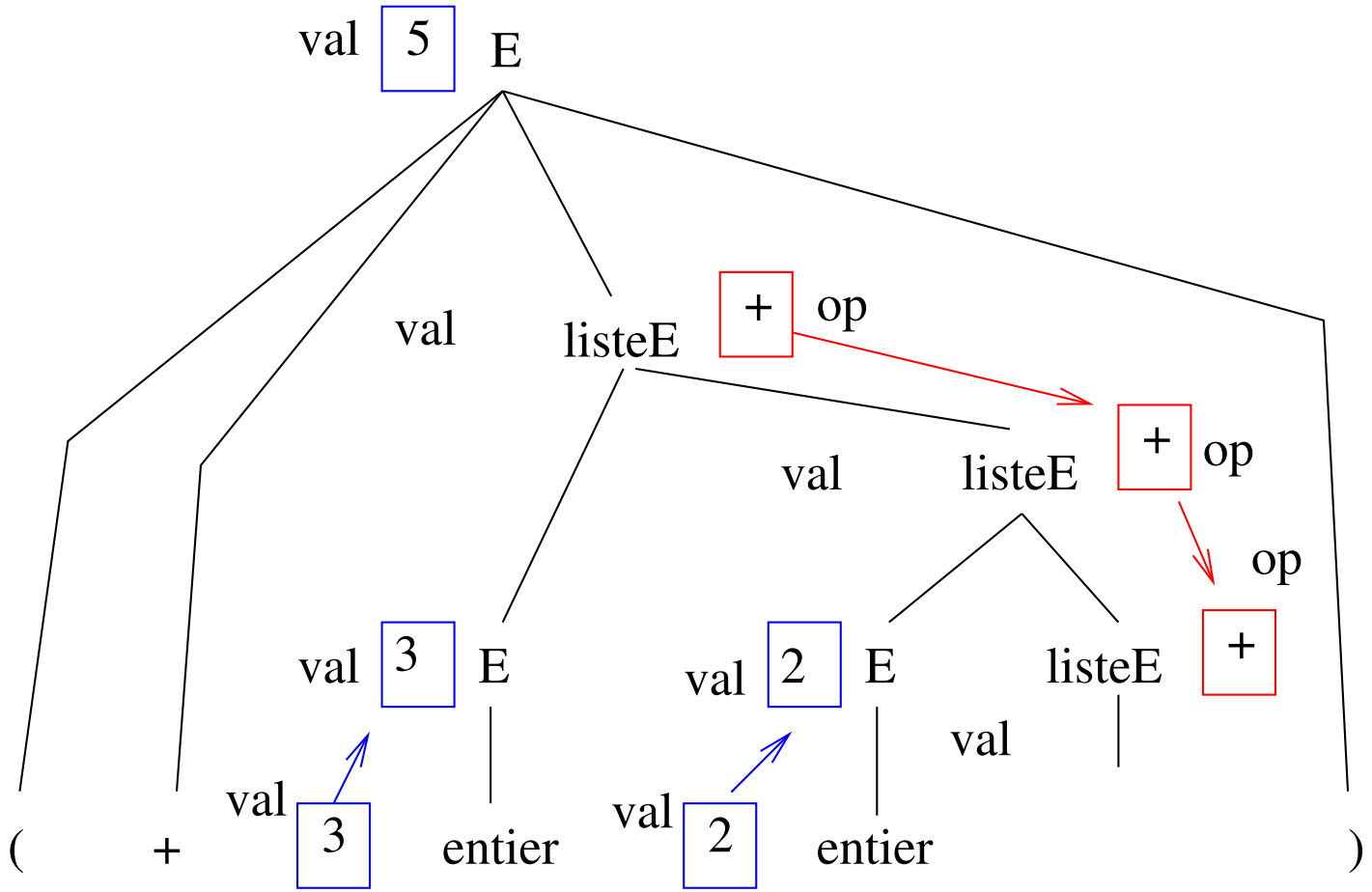
Comment faire pour *listeE* ?

listeE $\rightarrow$ *E listeE*

l'opérateur à appliquer n'apparaît pas

$\Rightarrow$ il faut que *listeE* connaisse l'opérateur à appliquer

Attribut *op* hérité pour *listeE*



Attribut *op* hérité pour *listeE*

attribut hérité *op*, type char, associé au non-terminal *listeE*

$E \rightarrow (+ \ listeE) \quad \{ \ listeE.op = + \ \}$
$E \rightarrow (* \ listeE) \quad \{ \ listeE.op = * \ \}$
$listeE \rightarrow E \ listeE \quad \{ \ listeE_1.op = listeE_0.op \ \}$

occurrences de définition occurrences d'utilisation

L'attribut hérité « descend » dans l'arbre.

Formellement

L'attribut $Y.a$ est hérité si Y apparaît en partie droite et si la valeur de $Y.a$ est calculée en fonction de la valeur d'attributs associés à des symboles apparaissant en partie gauche et/ou des autres symboles apparaissant en partie droite.

$$X \to X_1 \dots Y \dots X_n \{ Y.a = f(X.x, X_1.x_1, \dots, X_n.x_n) \}$$

pour $Y, X_i \in V_T \cup V_N$.

Toute production doit calculer la valeur des attributs hérités des symboles en partie droite (via une action).

Sur l'arbre syntaxique

Les attributs hérités :

- ≪ descendent ≫ du père
- ou viennent des frères de gauche ou des frères de droite.

En gros : tout ce qui ne remonte pas.

Remarque - cas de l'axiome

L'axiome ne peut avoir d'attributs hérités.

En effet l'axiome « à la racine » n'a ni père ni frères.

Au besoin, on rajoute un axiome bidon.

Ex : attribut nb de type entier associé à S, hérité

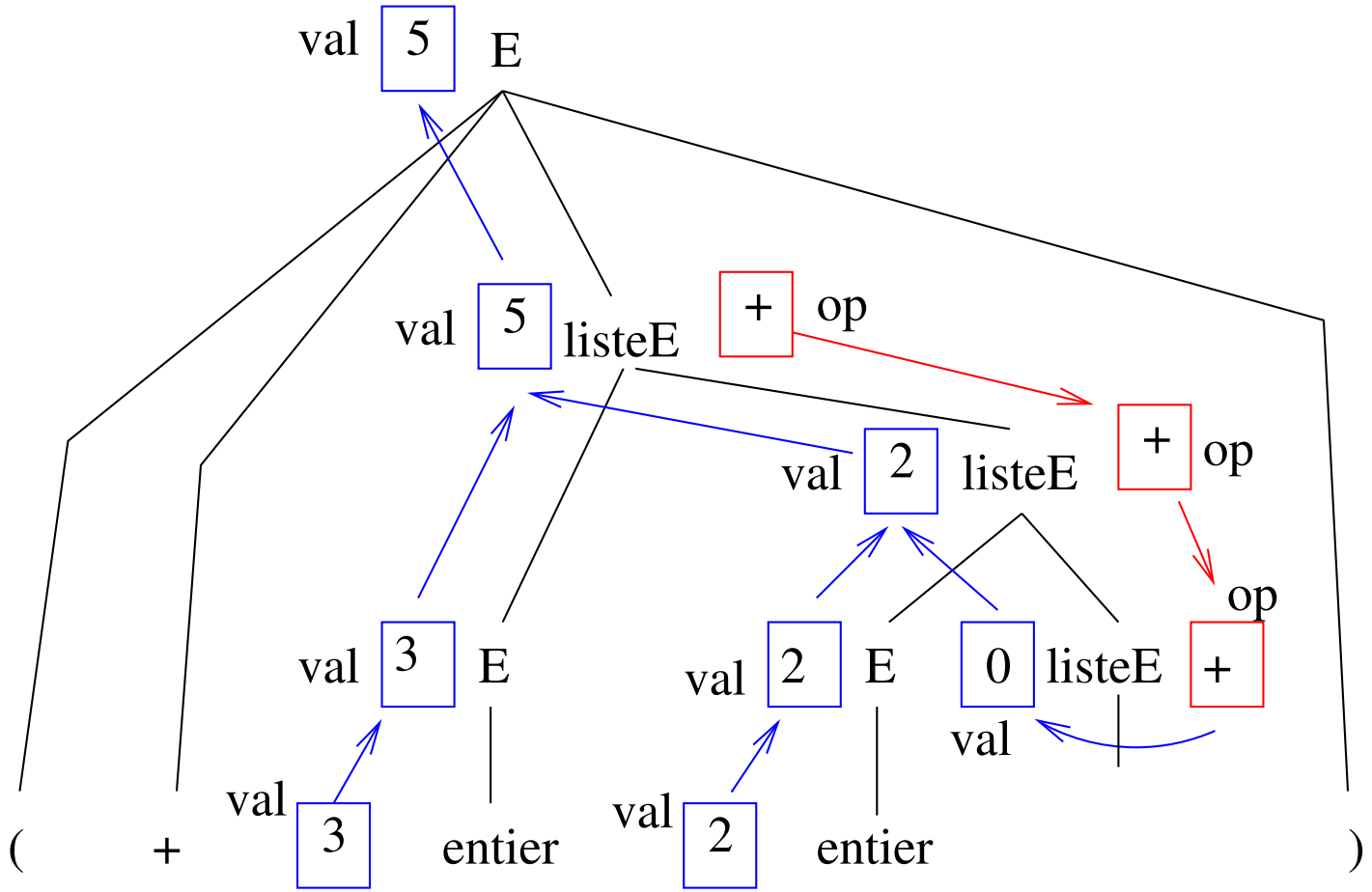
$S \to aaSb \quad \{ S_1.nb = S_0.nb + 2 \}$

$S \to \epsilon \quad \{ \text{afficher}(S.nb) \}$

$S' \to S \quad \{ S.nb = 0 \}$

Pour finir l'exemple

Attribut *val* synthétisé pour *listeE*



Attribution pour les synthétisés

$E \rightarrow (\ +\ listeE\ )$ $\quad \{\ E.val = listeE.val\ \}$

$E \rightarrow (\ *\ listeE\ )$ $\quad \{\ E.val = listeE.val\ \}$

$listeE \rightarrow E\ listeE$ $\quad \{$ if $listeE_0.op == +$

$\quad listeE_0.val = E.val + listeE_1.val$

else

$\quad listeE_0.val = E.val * listeE_1.val\ \}$

$listeE \rightarrow \epsilon$ $\quad \{$ if $listeE.op == +$

$\quad listeE.val = 0$

else

$\quad listeE.val = 1\ \}$

Attribution complète

$E \rightarrow (\ +\ listeE\ )$ $\{\ listeE.op = +$
$E.val = listeE.val\ \}$

$E \rightarrow (\ *\ listeE\ )$ $\{\ listeE.op = *$
$E.val = listeE.val\ \}$

$listeE \rightarrow E\ listeE$ $\{\ listeE_1.op = listeE_0.op$
if $listeE_0.op == +$
$listeE_0.val = E.val + listeE_1.val$
else
$listeE_0.val = E.val * listeE_1.val\ \}$

$listeE \rightarrow \epsilon$ $\{$ if $listeE.op == +$
$listeE.val = 0$
else
$listeE.val = 1\ \}$

Se passer des hérités : grammaire S-attribuée

Pour toute grammaire attribuée GA, il existe une grammaire attribuée GA' :

- qui engendre le même langage ;
- calculant les mêmes valeurs d'attributs ;

et qui possède uniquement des attributs synthétisés.

Une grammaire attribuée ne comportant que des attributs synthétisés est dite S-attribuée.

Comment faire pour les expressions préfixées ?

Exemple des expressions préfixées

Au lieu de descendre l'opérateur aux opérandes. . .

on remonte les opérandes à l'opérateur !

Attributs synthétisés :

- *val* pour *E* et *entier*, type entier, calcul idem avant
- *liste* pour *listeE*, type list<entier>

Attribution globale : tous synthétisés

$E \rightarrow$ (+ *listeE*) $\quad \{ E.val = \texttt{calcule}(+, listeE.liste) \}$

$E \rightarrow$ (* *listeE*) $\quad \{ E.val = \texttt{calcule}(*, listeE.liste) \}$

$listeE \rightarrow E\ listeE \quad \{$ liste_tmp $= listeE_1.liste$
liste_tmp.ajoutTete($E.val$)
$listeE_1.liste =$ liste_tmp $\}$

$listeE \rightarrow \epsilon \quad \{ listeE.liste = \texttt{new list<entier>} \}$

où `calcule(operateur, liste)` est programmée à part :

- liste vide : retourne l'élément neutre de `operateur`
- liste non vide : retourne l'accumulateur résultant de l'application de l'opérateur aux élements de la liste

2 approches différentes

Approche pragmatique (TP) :

- on collecte les données en calculant une structure de données
- on effectue les calculs sur cette structure

Approche plus ≪ théorie du langage ≫ :

- on fait circuler les données
- les calculs sont faits au fur et à mesure

Rappel : comment $\neq$ quand

La grammaire attribuée indique comment calculer la valeur des attributs.

Mais pas quand.

Ni dans quel ordre.

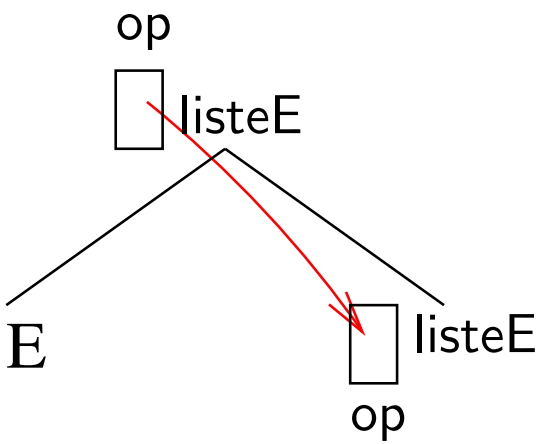
Graphe de dépendances

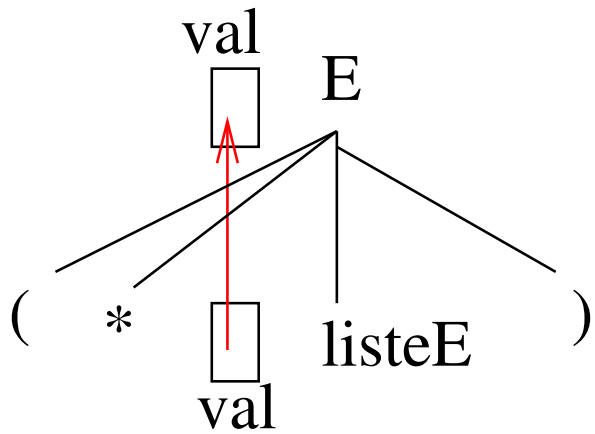
Dépendances de données ds les actions ("=" lu comme " :=") ;

$\Rightarrow$ construction d'un graphe de dépendances.

$listeE \rightarrow E\ listeE$
$\{\ listeE_1.op = listeE_0.op\ \}$

$E \rightarrow (\ *\ listeE\ )$
$\{\ E.val = listeE.val\ \}$





Ordre d'évaluation et graphe de dépendances

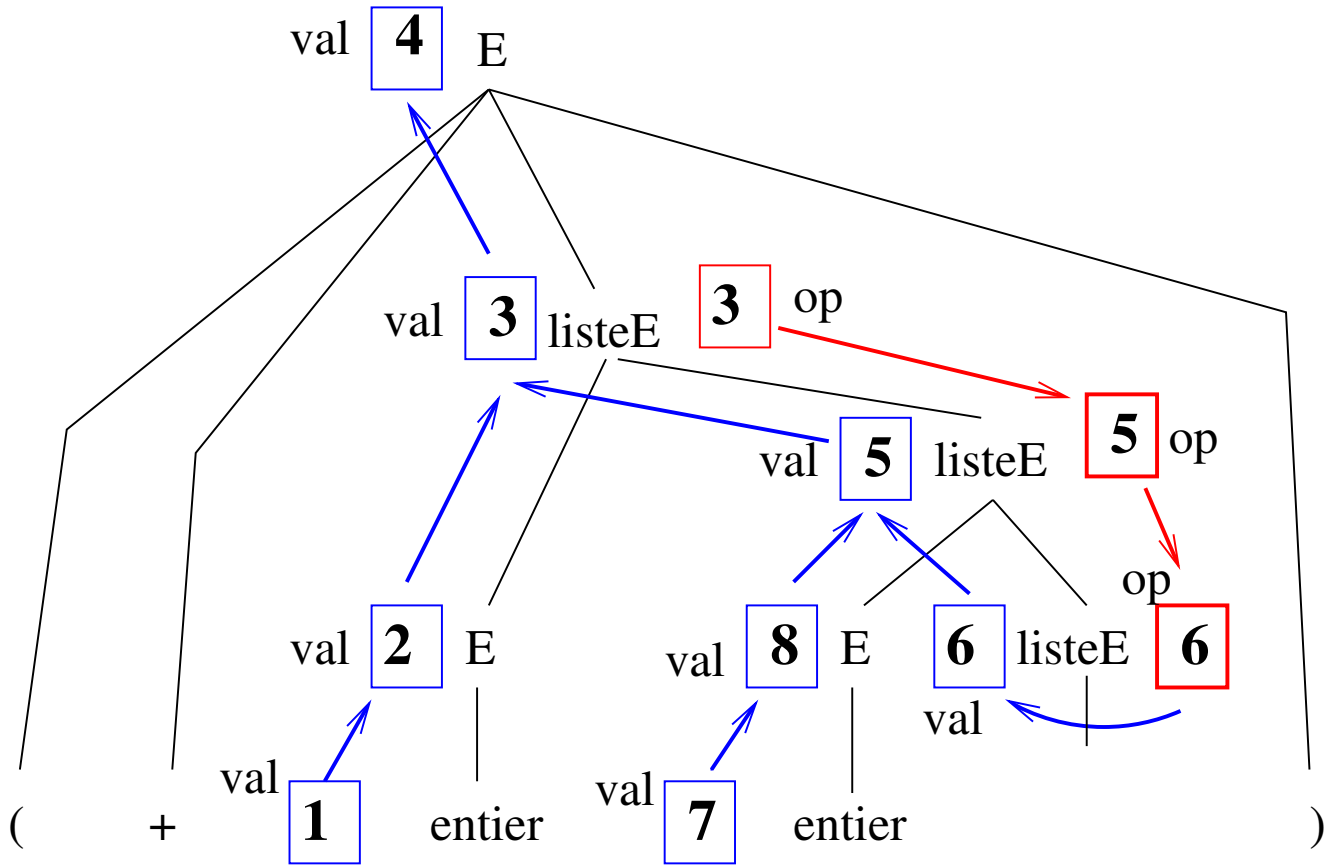
Si ce graphe est cyclique : grammaire mal formée.

Sinon (grammaire bien formée) : on peut trouver un ordre d'évaluation, une numérotation des actions.

On ne verra pas comment dans ce cours : seulement des exemples avec numérotation de nœuds

Attention : Dans l'ex qui suit le numéro en gras dans les cases font référence au numéro du nœud, pas à la valeur de l'attribut

Exemple



Ex : (**1**, val) < (**2**, val) < (**7**, val) < (**8**, val) < (**3**, op) < (**5**, op) < (**6**, op) < (**6**, val) < (**5**, val) < (**3**, val) < (**4**, val)

Problème et solutions

Problème :

- détection des dépendances cycliques pour un arbre syntaxique donné ;
- en cas d'absence de dépendances cycliques : calcul d'un ordre.

Deux grandes approches :

- utilisation d'un évaluateur d'attributs ;
- restriction des grammaires attribuées pour garantir l'absence de dépendances cycliques

  + effectuer les règles sémantiques pendant l'analyse syntaxique = TP