Compilation (#4) : Grammaires attribuées

C. Deleuze & L. Gonnord

Grenoble INP/Esisar

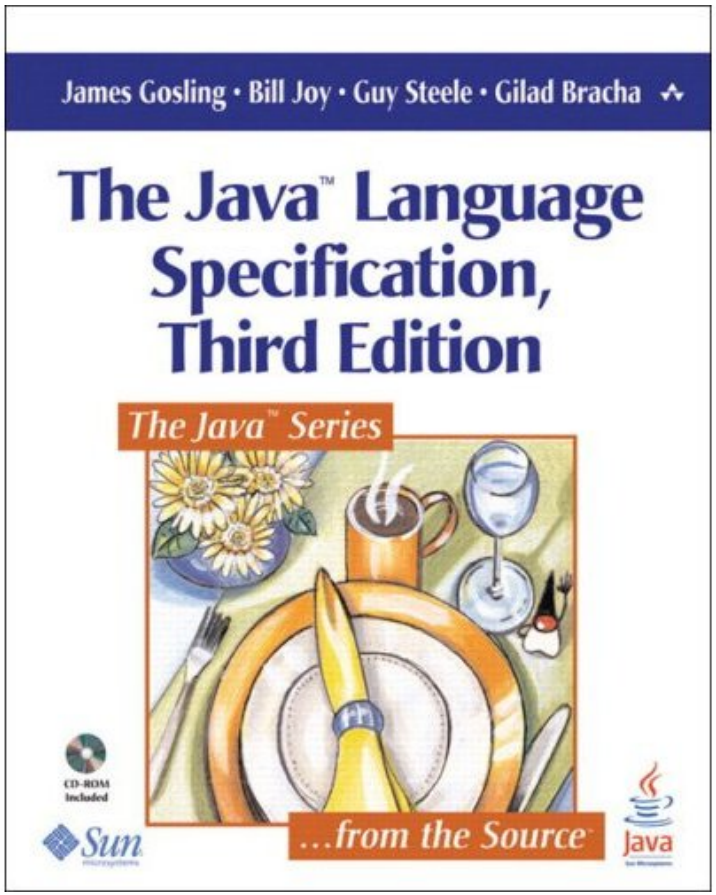
2022-2023

Meaning

How to define the meaning of programs in a given language ?

- Informal description most of the time (natural language, ISO, reference book. . .)
- Unprecise, ambiguous.

Informal Semantics



The Java programming language guarantees that the operands of operators appear to be evaluated in a specific evaluation order, namely, from left to right.

It is recommended that code not rely crucially on this specification.

https://docs.oracle.com/javase/specs/jls/se10/html/jls-15.html#jls-15.7

Formal semantics

The formal semantics mathematically characterises the computations done by a given program:

- useful to design tools (compilers, interpreters).
- mandatory to reason about programs and properties of the language.

Ce que l'on veut

- Ajouter à la grammaire de l'information qui sera traitée pendant l'analyse syntaxique.
- Jusqu'à présent on n'a construit que des accepteurs.
- Nous voulons effectuer des actions/collecter de l'information à chaque étape de l'analyse syntaxique.

2 Définitions dirigées par la syntaxe

- Définition
- Interpréteur : la calculette
- Propagation d'information
- Ordre d'évaluation

Une définition dirigée par la syntaxe est une généralisation d'une grammaire non contextuelle.

- chaque symbole de la grammaire possède un ensemble d'attributs :
  - synthétisés (calculés à partir des valeurs des attributs des fils)
  - hérités (calculés à partir des valeurs des attributs du père et des frères)
- chaque production A $\rightarrow \alpha$ de la grammaire possède un ensemble de règles sémantiques de la forme :

$$b = f(c_1, c_2, ...c_k)$$

où

- f est une fonction, et b est
  - soit un attribut synthétisé de A
  - soit un attribut hérité d'un des symboles de α
- $c_1, c_2, ...c_k$ sont des attributs de symboles quelconques de la production

On dit que b dépend des attributs $c_1, c_2, ...c_k$.

$$b = f(c_1, c_2, ...c_k)$$

où

- f est une fonction, et b est
  - soit un attribut synthétisé de A
  - soit un attribut hérité d'un des symboles de α
- $c_1, c_2, ...c_k$ sont des attributs de symboles quelconques de la production

On dit que b dépend des attributs $c_1, c_2, ...c_k$.

Grammaire attribuée : définition dirigée par la syntaxe dans laquelle les fonctions des règles sémantiques ne peuvent pas avoir d'effet de bord.

2 Définitions dirigées par la syntaxe

- Définition
- Interpréteur : la calculette
- Propagation d'information
- Ordre d'évaluation

Definition

From Wikipedia:

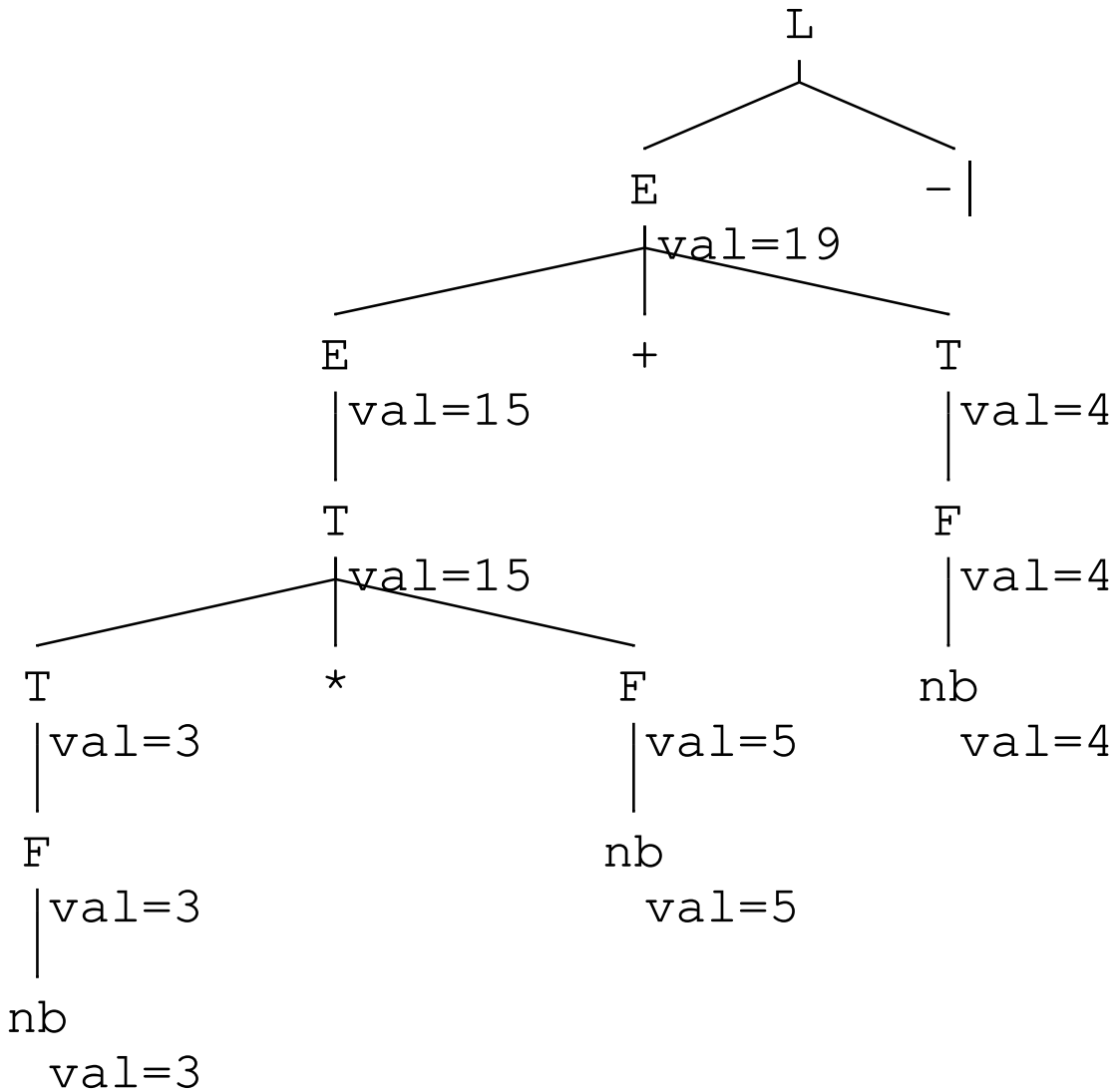
> *In computer science, an interpreter is a computer program that* ***directly executes instructions*** *written in a programming or scripting language, without requiring them previously to have been compiled into a machine language program.*

- An **interpreter** executes the input program according to the programming language **semantics**.

PRODUCTION	RÈGLE SÉMANTIQUE
$L \rightarrow E \dashv$	Imprimer(E.val)
$E \rightarrow E_1 + T$	E.val := E_1.val + T.val
$E \rightarrow T$	E.val := T.val
$T \rightarrow T_1 * F$	T.val := T_1.val * F.val
$T \rightarrow F$	T.val := F.val
$F \rightarrow (E)$	F.val := E.val
$F \rightarrow$ **nb**	F.val := **nb**.val

Figure: Définition dirigée par la syntaxe d'une calculette

Arbre décoré pour la calculette



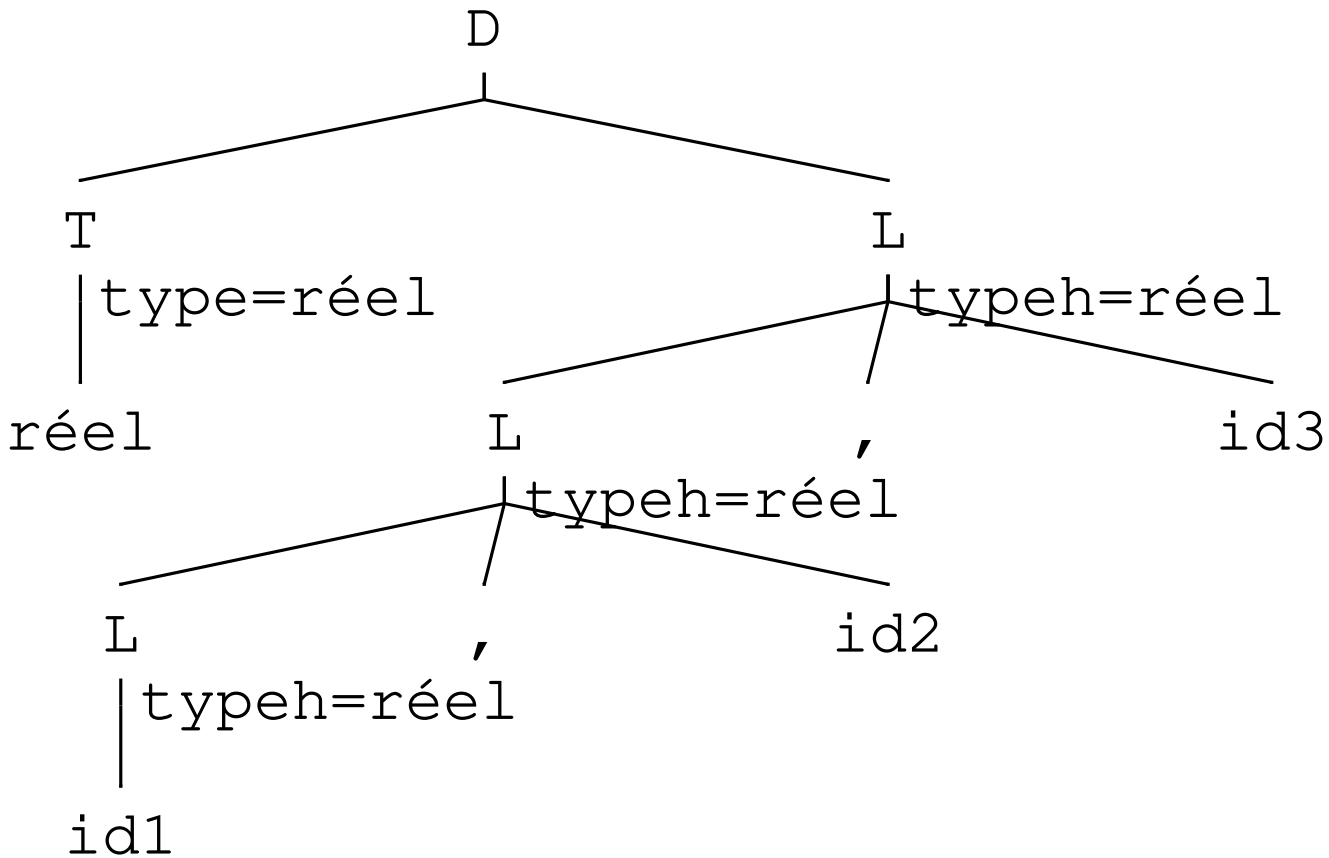
2 Définitions dirigées par la syntaxe

- Définition
- Interpréteur : la calculette
- Propagation d'information
- Ordre d'évaluation

PRODUCTION	RÈGLE SÉMANTIQUE
$D \rightarrow T\ L$	L.typeh := T.type
$T \rightarrow$ **entier**	T.type := entier
$T \rightarrow$ **réel**	T.type := réel
$L \rightarrow L_1$, **id**	L_1.typeh := L.typeh AjouterType(**id**.num, L.typeh)
$L \rightarrow$ **id**	AjouterType(**id**.num, L.typeh)

Figure: Définition dirigée par la syntaxe avec l'attribut hérité typeh

Arbre décoré pour la phrase `réel id1, id2, id3`



2 Définitions dirigées par la syntaxe

- Définition
- Interpréteur : la calculette
- Propagation d'information
- Ordre d'évaluation

Graphe de dépendances

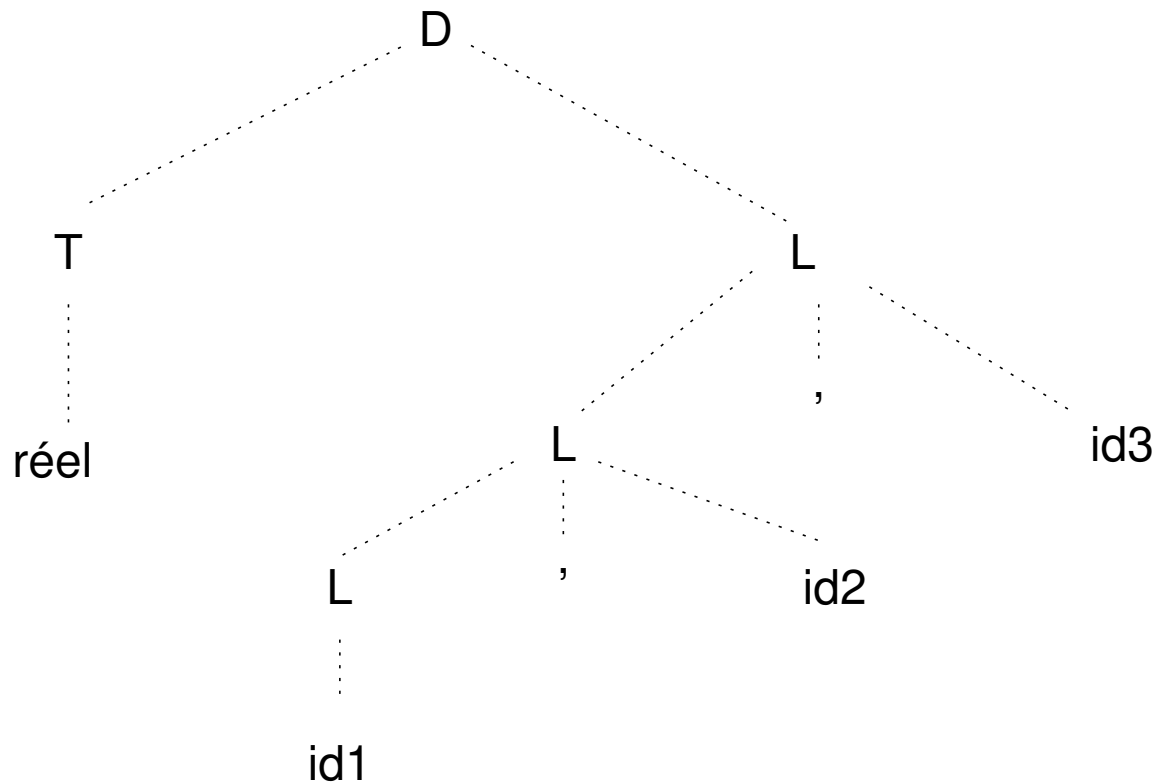
La valeur de certains attributs est calculée à partir de celle d'autres attributs, ce qui impose un ordre d'évaluation sur les règles sémantiques.
Les interdépendances entre attributs peuvent être décrites par un graphe orienté appelé graphe de dépendances. Ce graphe contient :

- un sommet pour chaque attribut,
- un arc de c à b si b dépend de c

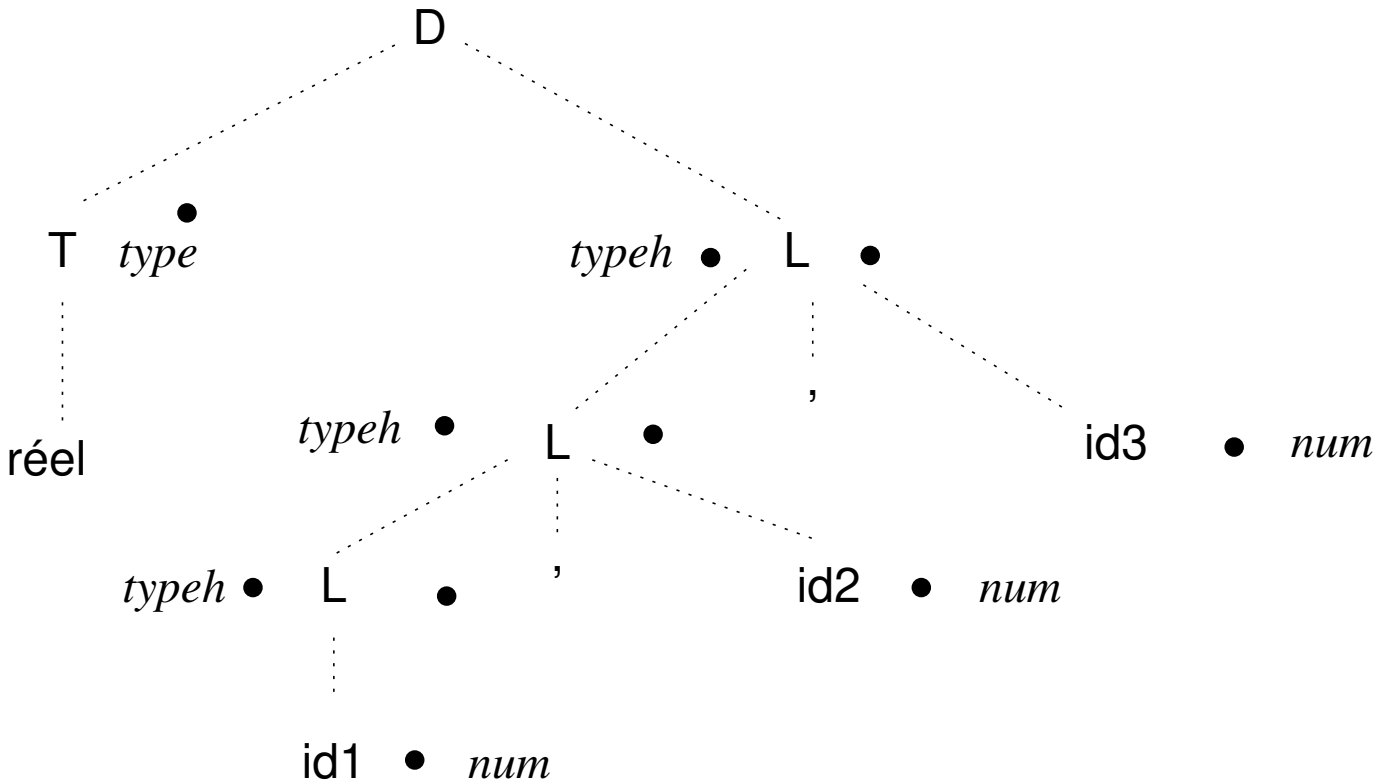
```
pour chaque nœud n de l'arbre syntaxique faire
   pour chaque attribut a du symbole de la
   grammaire étiquetant n faire
        construire un sommet dans le graphe de dép. pour a
pour chaque nœud n de l'arbre syntaxique faire
   pour chaque règle sémantique b:=f(c1, c2... ck)
   associée à la production appliquée en n faire
      pour i de 1 à k faire
           construire un arc du sommet correspondant à ci
           au sommet correspondant à b
```

Figure: Algorithme de construction du graphe de dépendances

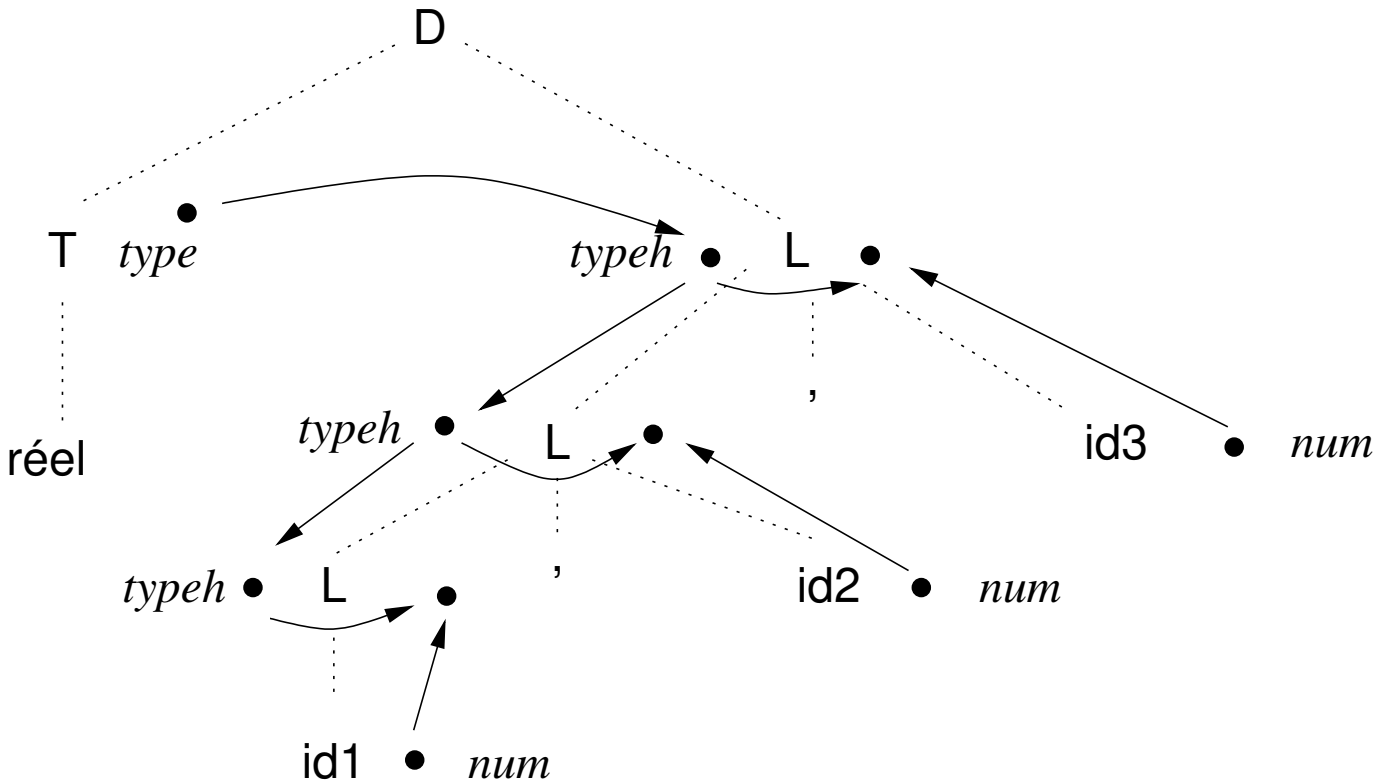
Graphe de dépendances pour l'arbre décoré précédent



Graphe de dépendances pour l'arbre décoré précédent



Graphe de dépendances pour l'arbre décoré précédent

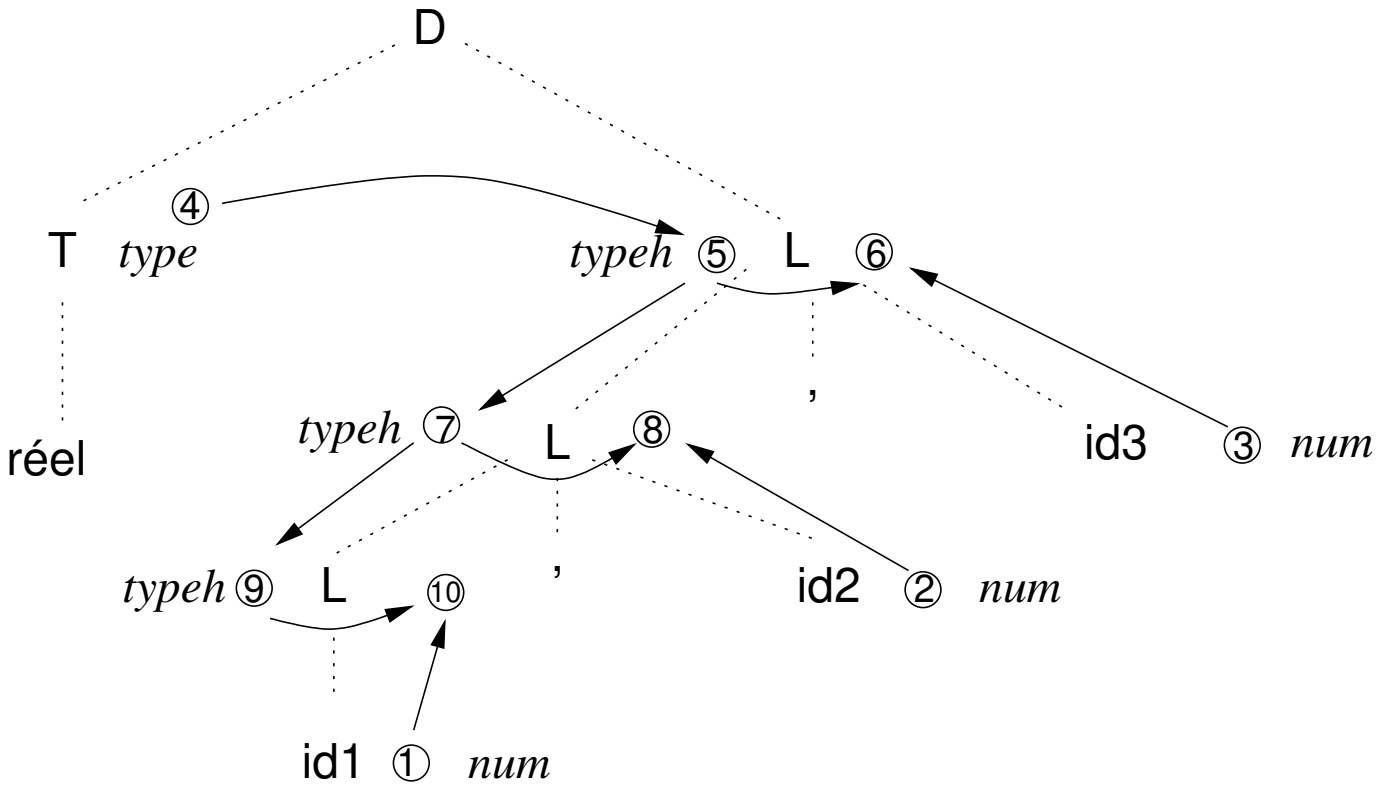


Tri topologique

Un tri topologique d'un graphe orienté acyclique est un ordonnancement quelconque $m_1, m_2 ... m_k$ des sommets du graphe tel que pour tous les arcs $m_i \rightarrow m_j$, m_i apparaît avant m_j dans l'ordonnancement.

Tout tri topologique d'un graphe de dépendances donne un ordre valide dans lequel les règles sémantiques peuvent être évaluées.

Graphe de dépendances pour l'arbre décoré précédent



- Une définition est S-attribuée si elle ne comporte que des attributs **s**ynthétisés.

- Une définition est S-attribuée si elle ne comporte que des attributs **s**ynthétisés.

- Une définition est L-attribuée si tout attribut hérité de X_j, $1 \leqslant j \leqslant n$ de la partie droite de la production A $\rightarrow$ X_1 X_2 ... X_n ne dépend que :
  1. des attributs des symboles X_1, X_2, ... X_{j-1} (à gauche de X_j dans la production)
  2. des attributs hérités de A.

3 Cas particuliers

- Définitions S-attribuées en ascendant
- Définitions S-attribuées en descendant
- Définitions L-attribuées

- évaluation possible pendant l'analyse ascendante

- évaluation possible pendant l'analyse ascendante
- sans construire l'arbre
- l'analyseur conserve dans sa pile les symboles **et** leur attribut
- à chaque réduction, calcule l'attribut du symbole vers lequel il réduit

Cas des générateurs LALR
yacc/bison/cup/...

Exemple : la calculette sur 3*5+4⊣

Pile	Entrée	Action
	nb_3*nb_5+nb_4 ⊣	

Exemple : la calculette sur 3*5+4⊣

Pile	Entrée	Action
	nb_3*nb_5+nb_4 ⊣	D

Exemple : la calculette sur 3*5+4⊣

Pile	Entrée	Action
	nb_3*nb_5+nb_4 ⊣	D
nb_3	*nb_5+nb_4 ⊣	

Exemple : la calculette sur 3*5+4⊣

Pile	Entrée	Action
	nb_3*nb_5+nb_4 ⊣	D
nb_3	*nb_5+nb_4 ⊣	R7

Exemple : la calculette sur 3*5+4⊣

Pile	Entrée	Action
	nb_3*nb_5+nb_4 ⊣	D
nb_3	*nb_5+nb_4 ⊣	R7
F_3	*nb_5+nb_4 ⊣	

Exemple : la calculette sur 3*5+4⊣

Pile	Entrée	Action
	nb_3*nb_5+nb_4 ⊣	D
nb_3	*nb_5+nb_4 ⊣	R7
F_3	*nb_5+nb_4 ⊣	R5

Exemple : la calculette sur 3*5+4⊣

Pile	Entrée	Action
	nb_3*nb_5+nb_4 ⊣	D
nb_3	*nb_5+nb_4 ⊣	R7
F_3	*nb_5+nb_4 ⊣	R5
T_3	*nb_5+nb_4 ⊣	

Exemple : la calculette sur 3*5+4⊣

Pile	Entrée	Action
	nb_3*nb_5+nb_4 ⊣	D
nb_3	*nb_5+nb_4 ⊣	R7
F_3	*nb_5+nb_4 ⊣	R5
T_3	*nb_5+nb_4 ⊣	D

Exemple : la calculette sur 3*5+4⊣

Pile	Entrée	Action
	nb_3*nb_5+nb_4 ⊣	D
nb_3	*nb_5+nb_4 ⊣	R7
F_3	*nb_5+nb_4 ⊣	R5
T_3	*nb_5+nb_4 ⊣	D
T_3*	nb_5+nb_4 ⊣	

Exemple : la calculette sur 3*5+4⊣

Pile	Entrée	Action
	nb_3*nb_5+nb_4 ⊣	D
nb_3	*nb_5+nb_4 ⊣	R7
F_3	*nb_5+nb_4 ⊣	R5
T_3	*nb_5+nb_4 ⊣	D
T_3*	nb_5+nb_4 ⊣	D

Exemple : la calculette sur 3*5+4⊣

Pile	Entrée	Action
	nb_3*nb_5+nb_4 ⊣	D
nb_3	*nb_5+nb_4 ⊣	R7
F_3	*nb_5+nb_4 ⊣	R5
T_3	*nb_5+nb_4 ⊣	D
T_3*	nb_5+nb_4 ⊣	D
T_3*nb_5	+nb_4 ⊣	

Exemple : la calculette sur 3*5+4⊣

Pile	Entrée	Action
	nb_3*nb_5+nb_4 ⊣	D
nb_3	*nb_5+nb_4 ⊣	R7
F_3	*nb_5+nb_4 ⊣	R5
T_3	*nb_5+nb_4 ⊣	D
T_3*	nb_5+nb_4 ⊣	D
T_3*nb_5	+nb_4 ⊣	R7

Exemple : la calculette sur 3*5+4⊣

Pile	Entrée	Action
	nb_3*nb_5+nb_4 ⊣	D
nb_3	*nb_5+nb_4 ⊣	R7
F_3	*nb_5+nb_4 ⊣	R5
T_3	*nb_5+nb_4 ⊣	D
T_3*	nb_5+nb_4 ⊣	D
T_3*nb_5	+nb_4 ⊣	R7
T_3*F_5	+nb_4 ⊣	

Exemple : la calculette sur 3*5+4⊣

Pile	Entrée	Action
	nb_3*nb_5+nb_4 ⊣	D
nb_3	*nb_5+nb_4 ⊣	R7
F_3	*nb_5+nb_4 ⊣	R5
T_3	*nb_5+nb_4 ⊣	D
T_3*	nb_5+nb_4 ⊣	D
T_3*nb_5	+nb_4 ⊣	R7
T_3*F_5	+nb_4 ⊣	R4

Exemple : la calculette sur 3*5+4⊣

Pile	Entrée	Action
	nb_3*nb_5+nb_4 ⊣	D
nb_3	*nb_5+nb_4 ⊣	R7
F_3	*nb_5+nb_4 ⊣	R5
T_3	*nb_5+nb_4 ⊣	D
T_3*	nb_5+nb_4 ⊣	D
T_3*nb_5	+nb_4 ⊣	R7
T_3*F_5	+nb_4 ⊣	R4
T_{15}	+nb_4 ⊣	

Exemple : la calculette sur 3*5+4⊣

Pile	Entrée	Action
	nb_3*nb_5+nb_4 ⊣	D
nb_3	*nb_5+nb_4 ⊣	R7
F_3	*nb_5+nb_4 ⊣	R5
T_3	*nb_5+nb_4 ⊣	D
T_3*	nb_5+nb_4 ⊣	D
T_3*nb_5	+nb_4 ⊣	R7
T_3*F_5	+nb_4 ⊣	R4
T_{15}	+nb_4 ⊣	R3
...		

Exemple : la calculette sur 3*5+4⊣

Pile	Entrée	Action
	nb_3*nb_5+nb_4 ⊣	D
nb_3	*nb_5+nb_4 ⊣	R7
F_3	*nb_5+nb_4 ⊣	R5
T_3	*nb_5+nb_4 ⊣	D
T_3*	nb_5+nb_4 ⊣	D
T_3*nb_5	+nb_4 ⊣	R7
T_3*F_5	+nb_4 ⊣	R4
T_{15}	+nb_4 ⊣	R3
...		
E_{19} ⊣		R1 → affiche 19

Exemple bison

```
%%
toplevel : toplevel exp '\n' { printf("val is %d\n", $2); }
         | {}
;
exp : exp '+' term { $$ = $1 + $3;}
    | term { $$ = $1;}
;
term : term '*' factor { $$ = $1 * $3;}
     | factor { $$ = $1;}
;
factor : '(' exp ')' { $$ = $2;}
       | INTVAL { $$ = $1;}
       ;
%%
```

3 Cas particuliers

- Définitions S-attribuées en ascendant
- Définitions S-attribuées en descendant
- Définitions L-attribuées

- analyseur à descente récursive
- chaque fonction retourne les attributs du nœud
  - appels récursifs donnent attributs des fils
  - calcul (règle sémantique de la production)

PRODUCTION	RÈGLE SÉMANTIQUE
$E \rightarrow T\ R$	E.val := T.val + R.val
$T \rightarrow P$	T.val := P.val
$T \rightarrow nb$	T.val := nb.val
$P \rightarrow (\ E\ )$	P.val := E.val
$R \rightarrow +\ E$	R.val := E.val
$R \rightarrow \varepsilon$	R.val := 0

Figure: Avec la grammaire du chapitre 2.a

Avec ANTLR

```
start : expr EOF {print($expr.v)} ;

expr returns [int v]
   : terme reste_expr {$v = $terme.v + $reste_expr.v } ;

terme returns [int v]
   : expr_par {$v = $expr_par.v }
   | NB {$v = $NB.int } ;

expr_par returns [int v]
   : '(' expr ')' {$v = $expr.v } ;

reste_expr returns [int v]
   : '+' expr {$v = $expr.v}
   | {$v = 0 } ;
```

Descente récursive

```
int parse(void) {
  int r = expression();
  consomme_ul(FDF);
  return r;
}

int expression(void) {
  int r1,r2;

  switch(prochaine_ul()) {
  case NB:
  case '(': r1=terme(); r2=reste_expression(); break;
  default: erreur();
  }
  return(r1+r2);
}

int terme(void) {
```

3 Cas particuliers

- Définitions S-attribuées en ascendant
- Définitions S-attribuées en descendant
- Définitions L-attribuées

- en descendant : ok car on progresse de gauche à droite
  - appels récursif : attributs en paramètres
  - mais les hérités ne doivent pas dépendre de synthétisés...
- en ascendant : possible aussi à certaines conditions...

Bilan

1. Program Semantics

2. Définitions dirigées par la syntaxe
   - Définition
   - Interpréteur : la calculette
   - Propagation d'information
   - Ordre d'évaluation

3. Cas particuliers
   - Définitions S-attribuées en ascendant
   - Définitions S-attribuées en descendant
   - Définitions L-attribuées