Pas de CM la semaine prochaine

2 CM la dermière semaine : mardi matin en plus
Venez à celui où vous êtes libre le mardi

Théorie des Langages

Cours 10 : Calcul d'attributs, analyse sémantique

Lionel Rieg

Grenoble INP - Ensimag, 1re année

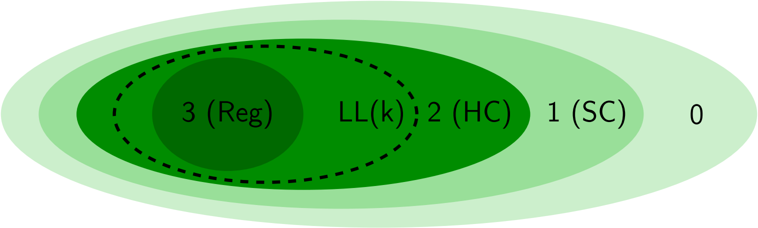
Langage régulier et LL(1)

Théorème

Tout langage régulier est LL(1).

Preuve : Convertir un automate déterministe en grammaire régulière.
Les directeurs sont triviaux (un terminal ou \$ si ε-règle).
Le déterminisme assure que la grammaire est LL(1).

Retour sur la hiérarchie de Chomsky



Calcul sur des arbres

Type	Reconnaissance	Calcul
3	automates finis	transducteurs (sortie sur les transitions)
2	grammaire HC	attributs (calcul sur l'arbre de dérivation)

Idée : chaque nœud (application de règle) fait une étape de calcul

Deux sens de calcul possibles dans un arbre :

- vers le bas : attributs hérités
  connaissance du haut de l'arbre (chemin vers la racine)
  permet de passer des arguments
- vers le haut : attributs synthétisés
  connaissance du sous-arbre (la partie analysée)
  permet de calculer le résultat pour un sous-arbre

Exemple (Valeur d'une expression arithmétique (sans attribut hérité))

$$\text{exp}{\uparrow}n \rightarrow \text{exp}{\uparrow}n_1 + \text{exp}{\uparrow}n_2 \qquad n = n_1 + n_2$$

$$\text{exp}{\uparrow}n \rightarrow \text{exp}{\uparrow}n_1 * \text{exp}{\uparrow}n_2 \qquad n = n_1 * n_2$$

Grammaire attribuée

Profil d'attribut :

pour chaque non-terminal X :

- des attributs hérités $\downarrow x \downarrow y$
- des attributs synthétisés $\uparrow z \uparrow t$

C'est le profil d'attribut de X, noté X$\downarrow \mathbb{N} \downarrow \mathbb{R} \uparrow \{\top, \bot\} \uparrow \mathbb{N}$.

pour chaque terminal $\mathbf{b} \uparrow z \uparrow t$:

- souvent rien du tout
- pas d'attribut hérité (pas de sous-arbre)
- des attributs synthétisés $\uparrow z \uparrow t$, donnés et non calculés

Exemple : exp$\uparrow n \;\rightarrow\; \mathbf{nb} \uparrow n_1 \qquad n = n_1$

Chaque règle X $\rightarrow w$ fixe les attr. hérités et calcule les attr. synthétisés :

- attributs hérités $\downarrow x \downarrow y$: de X vers w
- attributs synthétisés $\uparrow z \uparrow t$: de w vers X

$$\mathsf{X}\downarrow x \uparrow u \;\rightarrow\; \mathsf{X}\downarrow y_1 \uparrow v \; \mathsf{Y}\downarrow y_2 \downarrow y_3 \uparrow w \qquad y_i = f_i(x), \quad u = g(x, v, w)$$

$$\text{ou} \quad \mathsf{X}\downarrow x \uparrow g(x, v, w) \;\rightarrow\; \mathsf{X}\downarrow f_1(x) \uparrow v \; \mathsf{Y}\downarrow f_2(x) \downarrow f_3(x) \uparrow w$$

⚠ On peut avoir $y_2 = f(x, v)$.

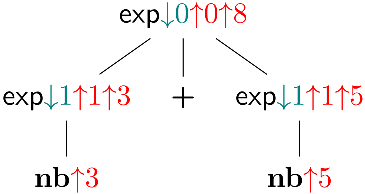
Grammaire attribuée (2)

Dans un arbre

1. attributs hérités descendent
2. attributs synthétisés remontent

Exemple : profondeur + valeur

Profils : exp$\downarrow\mathbb{N}\uparrow\mathbb{N}\uparrow\mathbb{N}$
$\mathbf{nb}\uparrow\mathbb{N}$



exp$\downarrow 0\uparrow 0\uparrow 8$

exp$\downarrow 1\uparrow 1\uparrow 3$ + exp$\downarrow 1\uparrow 1\uparrow 5$

$\mathbf{nb}\uparrow 3$ $\mathbf{nb}\uparrow 5$

Dans l'analyseur

attributs hérités $\downarrow p$ = entrée
attributs synthétisés $\uparrow n$ = résultat

```
def parse_exp($\downarrow p$):
  if current == nb:
    $\uparrow n$ = consume_token(nb)
    return ($\uparrow p$, $\uparrow n$)
  if current == +:
    $\uparrow p_1$,$\uparrow n_1$ = parse_exp($\downarrow (p+1)$)
    consume_token(+)
    $\uparrow p_1$,$\uparrow n_2$ = parse_exp($\downarrow (p+1)$)
    return ($\uparrow p$, $\uparrow (n_1+n_2)$)
```

On ajoute le calcul d'attribut à l'analyseur LL(1) comme arguments et résultats des fonctions.

Exercice

Sur la grammaire suivante, construisez un calcul d'attributs qui donne :

- la profondeur d'imbrication de chaque parenthèse
- le nombre de divisions

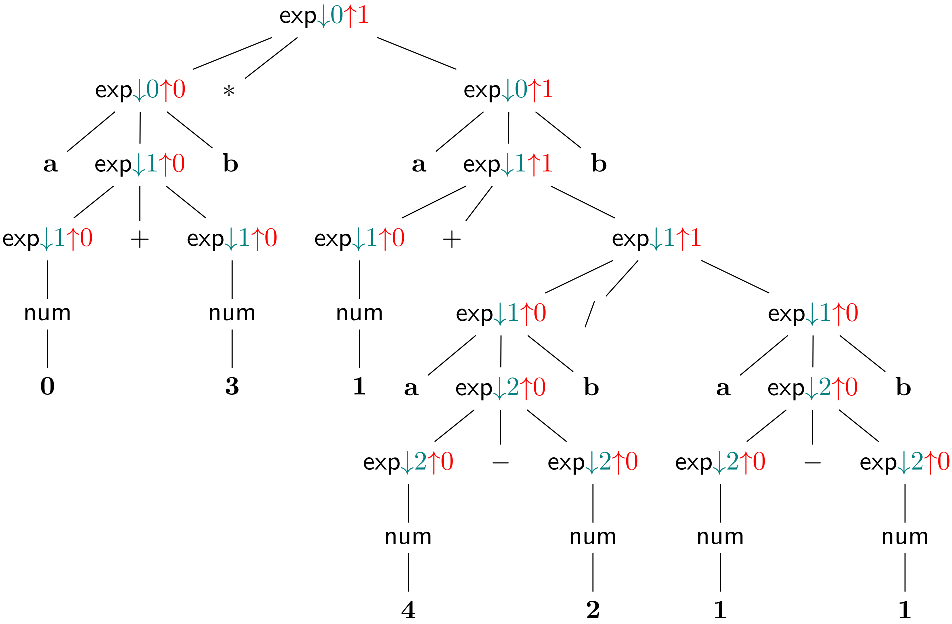
$$\begin{array}{lcl}
\text{exp} & \rightarrow & \text{num} \ \mid \ \text{exp op exp} \ \mid \ (\text{exp}) \\
\text{op} & \rightarrow & + \ \mid \ - \ \mid \ * \ \mid \ / \\
\text{num} & \rightarrow & \mathbf{0} \ \mid \ \ldots \ \mid \ \mathbf{9}
\end{array}$$

Donner l'arbre d'analyse et le calcul d'attributs sur l'entrée $(0+3)*(1+(4-2)/(1-1))$.

Profils : $\text{op}{\uparrow}\mathbb{N}$ $\text{exp}{\downarrow}\mathbb{N}{\uparrow}\mathbb{N}$

$$\begin{array}{lcll}
\text{exp}{\downarrow}p{\uparrow}n & \rightarrow & \text{num} & n = 0 \\
\text{exp}{\downarrow}p{\uparrow}n & \rightarrow & \text{exp}{\downarrow}p{\uparrow}n_1 \ \text{op}{\uparrow}k \ \text{exp}{\downarrow}p{\uparrow}n_2 & n = n_1 + k + n_2 \\
\text{exp}{\downarrow}p{\uparrow}n & \rightarrow & (\text{exp}{\downarrow}p+1{\uparrow}n) & \\
\text{op}{\uparrow}k & \rightarrow & + \ \mid \ - \ \mid \ * & n = 0 \\
\text{op}{\uparrow}k & \rightarrow & / & n = 1 \\
\text{num} & \rightarrow & \mathbf{0} \ \mid \ \ldots \ \mid \ \mathbf{9} &
\end{array}$$

Correction


exp↓0↑1
exp↓0↑0
*
exp↓0↑1
a
exp↓1↑0
b
a
exp↓1↑1
b
exp↓1↑0
+
exp↓1↑0
exp↓1↑0
+
exp↓1↑1
exp↓1↑0
/
exp↓1↑0
num
num
num
0
3
1
a
exp↓2↑0
b
a
exp↓2↑0
b
exp↓2↑0
−
exp↓2↑0
exp↓2↑0
−
exp↓2↑0
num
num
num
num
4
2
1
1

Transformation de grammaire et calcul d'attributs

Niveaux de priorités

Idée : on sépare les règles d'un non-terminal entre plusieurs non-terminaux

$$\begin{array}{lcll} \mathsf{X} \to \mathsf{X} \spadesuit \mathsf{X} & & \mathsf{X}_2 \to \mathsf{X}_2 \spadesuit \mathsf{X}_1 \mid \mathsf{X}_1 & \text{assoc. gauche} \\ \mathsf{X} \to \mathsf{X} \clubsuit \mathsf{X} & \leadsto & \mathsf{X}_1 \to \mathsf{X}_0 \clubsuit \mathsf{X}_1 \mid \mathsf{X}_0 & \text{assoc. droite} \\ \mathsf{X} \to (\,\mathsf{X}\,) \mid \mathbf{a} & & \mathsf{X}_0 \to (\,\mathsf{X}_2\,) \mid \mathbf{a} & \end{array}$$

Et sur le calcul d'attributs ?

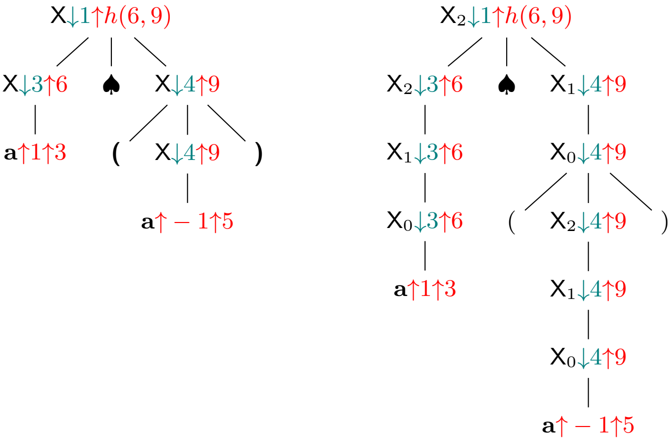
$$\begin{array}{ll} \mathsf{X}{\downarrow}n{\uparrow}p \to \mathsf{X}{\downarrow}f_1(n){\uparrow}p_1 \spadesuit \mathsf{X}{\downarrow}f_2(n){\uparrow}p_2 & p = h(p_1, p_2) \\ \mathsf{X}{\downarrow}n{\uparrow}p \to \mathsf{X}{\downarrow}g_1(n){\uparrow}p_1 \clubsuit \mathsf{X}{\downarrow}g_2(n){\uparrow}p_2 & p = i(p_1, p_2) \\ \mathsf{X}{\downarrow}n{\uparrow}p \to (\,\mathsf{X}{\downarrow}n{\uparrow}p\,) & \\ \mathsf{X}{\downarrow}n{\uparrow}p \to \mathbf{a}{\uparrow}m{\uparrow}k & p = n + \max(m, k) \end{array}$$

Profil des X_i : le même que X, $\mathsf{X}_i{\downarrow}n{\uparrow}p$

$$\begin{array}{ll} \mathsf{X}_2{\downarrow}n{\uparrow}p \to \mathsf{X}_2{\downarrow}f_1(n){\uparrow}p_1 \spadesuit \mathsf{X}_1{\downarrow}f_2(n){\uparrow}p_2 & p = h(p_1, p_2) \\ \mathsf{X}_2{\downarrow}n{\uparrow}p \to \mathsf{X}_1{\downarrow}n{\uparrow}p & \\ \mathsf{X}_1{\downarrow}n{\uparrow}p \to \mathsf{X}_0{\downarrow}g_1(n){\uparrow}p_1 \clubsuit \mathsf{X}_0{\downarrow}g_2(n){\uparrow}p_2 & p = i(p_1, p_2) \\ \mathsf{X}_1{\downarrow}n{\uparrow}p \to \mathsf{X}_0{\downarrow}n{\uparrow}p & \\ \mathsf{X}_0{\downarrow}n{\uparrow}p \to (\,\mathsf{X}_2{\downarrow}n{\uparrow}p\,) & \\ \mathsf{X}_0{\downarrow}n{\uparrow}p \to \mathbf{a}{\uparrow}m{\uparrow}k & p = n + \max(m, k) \end{array}$$

Exemple d'arbre

Pour $f_1(n) = n + 2$ et $f_2(n) = 5 - n$



Factorisation

Idée : on rassemble en une règle le préfixe commun

$$\begin{array}{lcl} \mathsf{exp}_1 & \to & \mathsf{exp}_0 + \mathsf{exp}_1 \;\mid\; \mathsf{exp}_0 \\ \mathsf{exp}_0 & \to & \mathbf{nb} \;\mid\; (\,\mathsf{exp}_1\,) \end{array} \qquad \rightsquigarrow \qquad \begin{array}{lcl} \mathsf{exp}_1 & \to & \mathsf{exp}_0\,\mathsf{exp}'_1 \\ \mathsf{exp}'_1 & \to & +\,\mathsf{exp}_1 \;\mid\; \varepsilon \\ \mathsf{exp}_0 & \to & \mathbf{nb} \;\mid\; (\,\mathsf{exp}_1\,) \end{array}$$

Et sur le calcul d'attributs ?

$$\begin{array}{lcll} \mathsf{exp}_1{\uparrow}n & \to & \mathsf{exp}_0{\uparrow}n_1 + \mathsf{exp}_1{\uparrow}n_2 & n = n_1 + n_2 \\ \mathsf{exp}_1{\uparrow}n & \to & \mathsf{exp}_0{\uparrow}n & \\ \mathsf{exp}_0{\uparrow}n & \to & \mathbf{nb}{\uparrow}n \;\mid\; (\,\mathsf{exp}_1{\uparrow}n\,) & \end{array}$$

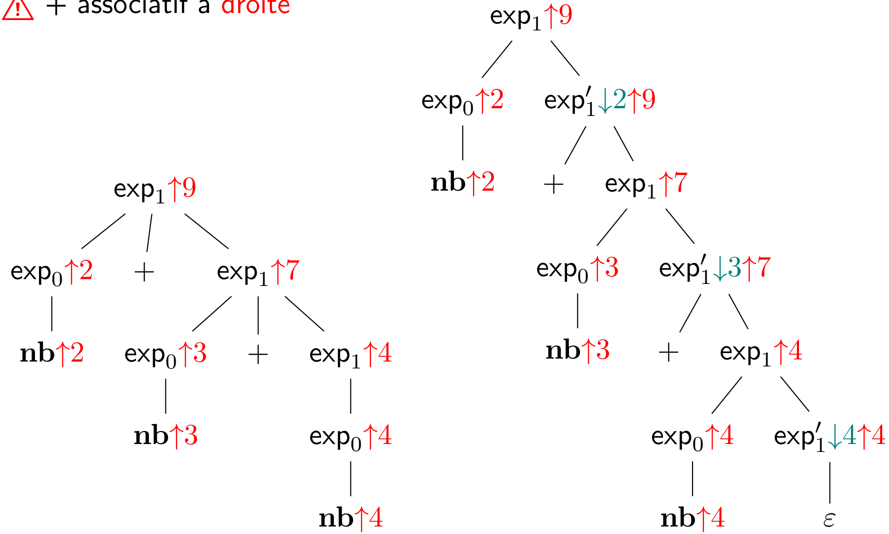
Profils : exp_1, exp_0 comme avant ; $\mathsf{exp}'_1{\downarrow}\mathbb{N}{\uparrow}\mathbb{N}$

$$\begin{array}{lll} \mathsf{exp}_1{\uparrow}n & \to \mathsf{exp}_0{\uparrow}n_1\,\mathsf{exp}'_1{\downarrow}n_1{\uparrow}n & \\ \mathsf{exp}'_1{\downarrow}n_1{\uparrow}n & \to +\,\mathsf{exp}_1{\uparrow}n_2 & n = n_1 + n_2 \\ \mathsf{exp}'_1{\downarrow}n_1{\uparrow}n & \to \varepsilon & n = n_1 \\ \mathsf{exp}_0{\uparrow}n & \to \mathbf{nb}{\uparrow}n \;\mid\; (\,\mathsf{exp}_1{\uparrow}n\,) & \end{array}$$

Exemple de factorisation

Mot lu : $\mathbf{nb}\uparrow 2 + \mathbf{nb}\uparrow 3 + \mathbf{nb}\uparrow 4$

⚠ + associatif à droite



Élimination de la récursion gauche

Idée : on utilise le lemme d'Arden pour reformuler la règle

$$\begin{array}{lcl} \mathsf{exp}_1 & \to & \mathsf{exp}_1 + \mathsf{exp}_0 \;\mid\; \mathsf{exp}_0 \\ \mathsf{exp}_0 & \to & \mathbf{nb} \;\mid\; (\,\mathsf{exp}_1\,) \end{array} \quad \rightsquigarrow \quad \begin{array}{lcl} \mathsf{exp}_1 & \to & \mathsf{exp}_0\,\mathsf{exp}'_1 \\ \mathsf{exp}'_1 & \to & +\,\mathsf{exp}_0\,\mathsf{exp}'_1 \;\mid\; \varepsilon \\ \mathsf{exp}_0 & \to & \mathbf{nb} \;\mid\; (\,\mathsf{exp}_1\,) \end{array}$$

Et sur le calcul d'attributs ?

$$\begin{array}{lcll} \mathsf{exp}_1{\uparrow}n & \to & \mathsf{exp}_1{\uparrow}n_1 + \mathsf{exp}_0{\uparrow}n_2 & n = n_1 + n_2 \\ \mathsf{exp}_1{\uparrow}n & \to & \mathsf{exp}_0{\uparrow}n & \\ \mathsf{exp}_0{\uparrow}n & \to & \mathbf{nb}{\uparrow}n \;\mid\; (\,\mathsf{exp}_1{\uparrow}n\,) & \end{array}$$

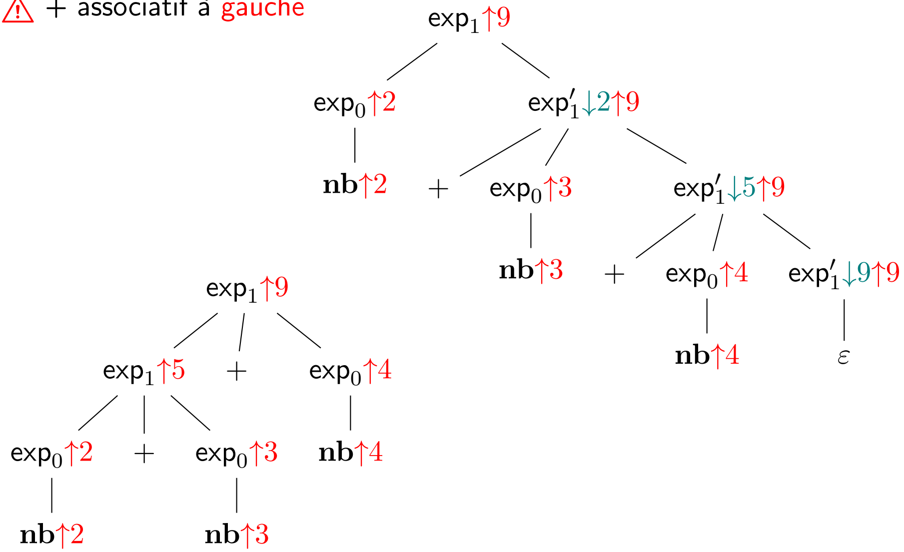
Profils : exp_1, exp_0 comme avant ; $\mathsf{exp}'_1{\downarrow}\mathbb{N}{\uparrow}\mathbb{N}$

$$\begin{array}{lll} \mathsf{exp}_1{\uparrow}n & \to \mathsf{exp}_0{\uparrow}n_1\,\mathsf{exp}'_1{\downarrow}n_1{\uparrow}n & \\ \mathsf{exp}'_1{\downarrow}n_1{\uparrow}n & \to +\,\mathsf{exp}_0{\uparrow}n_2\,\mathsf{exp}'_1{\downarrow}(n_1+n_2){\uparrow}n & \\ \mathsf{exp}'_1{\downarrow}n_1{\uparrow}n & \to \varepsilon & n = n_1 \\ \mathsf{exp}_0{\uparrow}n & \to \mathbf{nb}{\uparrow}n \;\mid\; (\,\mathsf{exp}_1{\uparrow}n\,) & \end{array}$$

Exemple d'élimination de la récursion gauche

Mot lu : $\mathbf{nb}\uparrow 2 + \mathbf{nb}\uparrow 3 + \mathbf{nb}\uparrow 4$

⚠ + associatif à gauche



Ordre des calculs

Analyseur construit

```
def parse_exp1():
  $\uparrow n_1$ = parse_exp0()
  $\uparrow n$ = parse_exp1P($\downarrow n_1$)
  return $\uparrow n$
```

```
def parse_exp1P($\downarrow n_1$):
  if current == $+$:
    consume_token(+)
    $\uparrow n_2$ = parse_exp0()
    $\uparrow n$ = parse_exp1P($\downarrow (n_1 + n_2)$)
    return $\uparrow n$
  else:
    return $\uparrow n_1$
```

Dépendance des calculs

Utilisation des résultats d'une ligne vers la suivante
$\rightsquigarrow$ de gauche à droite dans l'arbre $=$ sens de lecture
Après lecture, on ne garde que le résultat du calcul.
Grammaire L-attribuée

Si dépendance de droite à gauche,
il faut garder tout l'arbre pour le calcul d'attributs.

Utilisation du calcul d'attributs pour la reconnaissance

Le calcul d'attribut permet de dépasser les limites des langages HC.

Exercice : Reconnaissance de $\{a^n b^n c^n \mid n \in \mathbb{N}\}$

1. Donner une grammaire LL(1) pour le langage $\{a^n b^n c^p \mid n, p \in \mathbb{N}\}$.
2. Munir votre grammaire d'un calcul d'attribut qui calcule $f(w) \overset{\text{def}}{=} |w|_b - |w|_c$.
3. En déduire un algorithme de reconnaissance pour $\{a^n b^n c^n \mid n \in \mathbb{N}\}$.
4. Peut-on généraliser cette méthode à $\{a^n b^n c^n d^n \mid n \in \mathbb{N}\}$?

Correction

1. $$\begin{array}{lcll} \mathsf{S} & \to & \mathsf{T}\,\mathsf{C} & \{\mathbf{a},\mathbf{c},\$\} \end{array}$$
$$\left.\begin{array}{lcll} \mathsf{T} & \to & \mathbf{a}\,\mathsf{T}\,\mathbf{b} & \{\mathbf{a}\} \\ \mathsf{T} & \to & \varepsilon & \{\mathbf{b},\mathbf{c},\$\} \end{array}\right\}\checkmark$$
$$\left.\begin{array}{lcll} \mathsf{C} & \to & \mathbf{c}\,\mathsf{C} & \{\mathbf{c}\} \\ \mathsf{C} & \to & \varepsilon & \{\$\} \end{array}\right\}\checkmark$$

2. $$\begin{array}{lcll} \mathsf{S}\uparrow n & \to & \mathsf{T}\uparrow k\;\mathsf{C}\uparrow p & n = k - p \\ \mathsf{T}\uparrow n & \to & \mathbf{a}\,\mathsf{T}\uparrow k\;\mathbf{b} & n = k + 1 \\ \mathsf{T}\uparrow n & \to & \varepsilon & n = 0 \\ \mathsf{C}\uparrow n & \to & \mathbf{c}\,\mathsf{C}\uparrow p & n = p - 1 \\ \mathsf{C}\uparrow n & \to & \varepsilon & n = 0 \end{array}$$

3. On effectue le calcul d'attributs et on vérifie que $f(w) = 0$.

4. On pourrait calculer à la place $f(w) = \langle |w|_b, |w|_c, |w|_d \rangle$ puis vérifier que les trois sont égaux.

Analyse sémantique

Erreurs dans un programme

Types d'erreurs :

- lexicales : `3.gh`
- grammaticales : **if** . . . **else** . . . **else** . . .
- de définition et déclaration : `a + 2` (sans avoir défini `a`)
- de types : `3 + "toto"`
- de valeurs : `sqrt(-4)`
- de résultat
- d'accès aux ressources
- etc.

Peut-on repérer un maximum de ces erreurs à l'avance ?
(l'analyse syntaxique n'est pas suffisante)

Analyse sémantique = vérifier que le programme a un sens
Indécidable donc réponse approchée

Langage jouet

$$
\begin{array}{lcl}
\text{prog} & \rightarrow & \text{decls insts} \\
\text{decls} & \rightarrow & \text{decl decls} \mid \varepsilon \\
\text{decl} & \rightarrow & \mathbf{idf} : \text{type} \,; \\
\text{type} & \rightarrow & \mathbf{int} \mid \mathbf{float} \mid \mathbf{bool} \mid \text{type} * \\
\text{insts} & \rightarrow & \text{affect insts} \mid \varepsilon \\
\text{affect} & \rightarrow & \text{lvalue} := \text{exp} \,; \\
\text{lvalue} & \rightarrow & \mathbf{idf} \mid * \,\text{lvalue} \\
\text{exp} & \rightarrow & \mathbf{num} \mid \mathbf{true} \mid \mathbf{false} \mid \text{lvalue} \mid \text{exp op exp} \mid (\,\text{exp}\,) \\
\text{op} & \rightarrow & + \mid - \mid * \mid / \mid \mathbf{and} \mid \mathbf{or}
\end{array}
$$

Que peut-on vouloir vérifier sur de tels programmes ?

- les types
- la bonne déclaration des identifiants utilisés
- la déclaration unique de chaque identifiant
- les opérations sur les pointeurs (*, +, -)

On va concevoir des calculs d'attributs pour ce faire.

Portée (scope)

Vérification que les variables utilisées sont bien définies

Qu'est-ce qui définit une variable ?

- une affectation `x = 42`
- un appel de fonction (les arguments)

Calcul d'attributs : variables déclarées D et/ou variables utilisées U

$$\begin{array}{lcl}
\text{prog}\uparrow D\uparrow U & \rightarrow & \text{decls}\uparrow D\ \text{insts}\downarrow D\uparrow U \\
\text{decls}\uparrow D & \rightarrow & \text{decl}\uparrow n\ \text{decls}\uparrow D_1 \quad D = D_1 \cup \{n\} \ \mid\ \varepsilon \quad D = \emptyset \\
\text{decl}\uparrow n & \rightarrow & \mathbf{idf}\uparrow n : \text{type} ; \\
\text{type} & \rightarrow & \mathbf{int} \mid \mathbf{float} \mid \mathbf{bool} \mid \text{type}\,* \\
\text{insts}\downarrow D\uparrow U & \rightarrow & \text{affect}\downarrow D\uparrow U_1\ \text{insts}\downarrow D\uparrow U_2 \mid \varepsilon \\
\text{affect}\downarrow D\uparrow U & \rightarrow & \text{lvalue}\uparrow n := \text{exp}\uparrow U_1 ; \quad U = U_1 \cup \{n\} \quad \text{assert}\ n \in D \\
\text{lvalue}\uparrow n & \rightarrow & \mathbf{idf}\uparrow n \mid *\,\text{lvalue}\uparrow n \\
\text{exp}\uparrow U & \rightarrow & \mathbf{num} \mid \mathbf{true} \mid \mathbf{false} \mid \text{lvalue}\uparrow n \quad U = \{n\} \\
 & \mid & \text{exp}\uparrow U_1\ \text{op}\ \text{exp}\uparrow U_2 \quad U = U_1 \cup U_2 \mid (\,\text{exp}\uparrow U\,) \\
\text{op} & \rightarrow & + \mid - \mid * \mid / \mid \mathbf{and} \mid \mathbf{or}
\end{array}$$

⚠ Certains langages (Perl, LISP) ont des portées dynamiques.

Autres propriétés

- Déclaration unique des identifiants
  $\rightsquigarrow$ seulement dans la partie déclaration
- Typage d'une affectation
  $\rightsquigarrow$ environnement de typage + vérification des affectations
- Typage des opérations sur les pointeurs
  - déréférencement $*$
  - addition entre pointeur et entier
  - soustraction entre pointeur et entier/pointeur
    ⚠ soustraction entre deux pointeurs permise si dans le même bloc
- Surcharge (choix entre plusieurs types possibles)

Plus de détail sur le typage la semaine prochaine

Souvent, ces analyses sont réalisées sur une structure dédiée et non sur l'arbre d'analyse grammaticale.
$\rightsquigarrow$ Arbre de syntaxe abstraite (AST) = première représentation intermédiaire (IR)

Construction d'un AST

La structure d'arbre est adaptée pour comprendre et utiliser la structure d'un programme.
Mais l'arbre d'analyse LL(1) est trop détaillé :

- niveaux de priorité ininterressant pour la suite
- pas besoin des non-terminaux
- sucre syntaxique : plusieurs écritures pour une même opération (let/where, +=, etc.)
- etc.

⇝ autre structure d'arbre que celle de l'analyse (proche de la grammaire ambiguë de départ)

Est-que l'ambiguïté est un problème ici ?