CH.4 LA TRADUCTION

- 4.1 Les grammaires attribuées
- 4.2 Les arbres abstraits
- 4.3 Les définitions S-attribuées
- 4.4 Les définitions L-attribuées
- 4.5 La traduction descendante
- 4.6 L'évaluation ascendante des attributs hérités
- 4.7 L'évaluation récursive

4.1 Les grammaires attribuées

Chaque symbole possède des **attributs**.
Aux règles de la grammaire sont ajoutées des **actions** permettant le calcul des attributs.

Exemple :

Règle	**Action**
$A \rightarrow E\$$	*print*(*E.val*)
$E \rightarrow E + T$	$E.val := E_1.val + T.val$
$E \rightarrow T$	$E.val := T.val$
$T \rightarrow T * F$	$T.val := T_1.val * F.val$
$T \rightarrow F$	$T.val := F.val$
$F \rightarrow (E)$	$F.val := E.val$
$F \rightarrow$ **n**	$F.val :=$ **n**.*vallex*

Grammaire attribuée d'une calculette

Situation générale :
Règle $A \rightarrow \alpha$ et actions $b := f(c_1, c_2, ..., c_k)$.

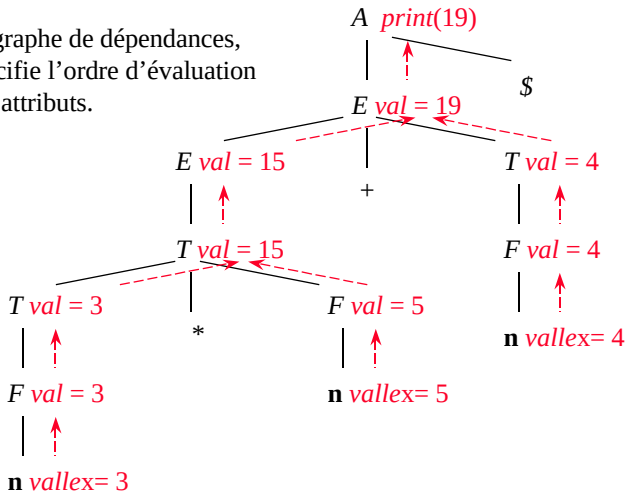
Attributs :
- hérités : b est un attribut d'un symbole constituant α et les c_i sont des attributs des symboles constituant α ou de A ;
- synthétisés : b est un attribut de A et les c_i sont des attributs des symboles constituant α ou des attributs hérités de A.

Dans l'exemple précédent, tous les attributs sont synthétisés.

Arbre décoré : arbre de dérivation sur lequel sont rajoutées les valeurs des attributs.
Problème : ordre d'évaluation des attributs. Si tous les attributs sont synthétisés, l'évaluation peut se faire des feuilles vers la racine.

Exemple : 3 * 5 + 4*$*

En tireté : graphe de dépendances, spécifie l'ordre d'évaluation des attributs.

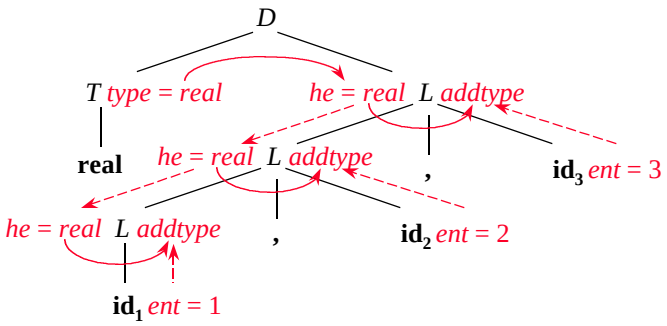


Attributs hérités :
Exemple : Déclaration de variables en C

Règle	**Action**
$D \rightarrow TL$	*L.he* := *T.type*
$T \rightarrow$ **int**	*T.type* := *integer*
$T \rightarrow$ **real**	*T.type* := *real*
$L \rightarrow L$ **, id**	L_1*.he* := *L.he*
	addtype(**id**.*ent*, *L.he*)
$L \rightarrow$ **id**	*addtype*(**id**.*ent*, *L.he*)

L'attribut *L.he* est hérité ; les autres sont synthétisés.

Exemple : real id_1, id_2, id_3



L'évaluation est possible parce que le graphe des dépendances est **acyclique**. Évaluation selon un **ordre topologique**.

4.2 Les arbres abstraits

Représentation permettant de dissocier analyse syntaxique et traduction.

Les règles sont représentées comme des opérateurs, les symboles étant les opérandes. Les variables de la grammaire ont disparu, et donc les productions unitaires.

Exemple : Expressions additives.

On utilise trois fonctions auxiliaires :

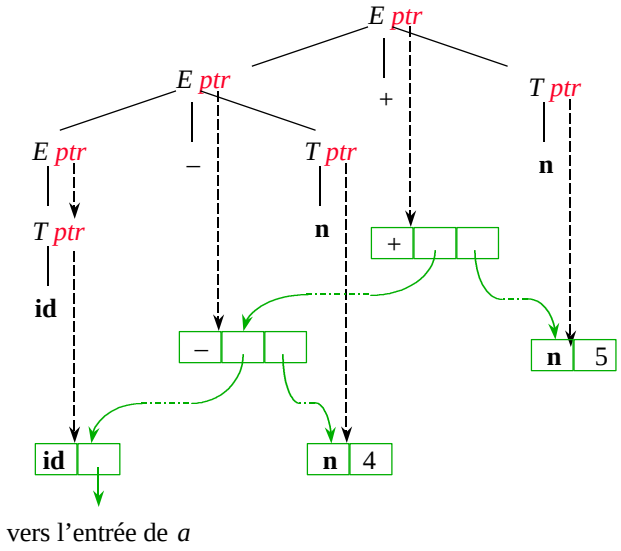
1. *mknode*(*op*, *left*, *right*) crée un nœud "opérateur binaire" d'étiquette *op* avec deux champs contenant des pointeurs vers *left* et *right* ;
2. *mkleaf*(**id**, *ent*) crée un nœud "identificateur" d'étiquette **id** avec un champ contenant un pointeur vers la table des symboles;
3. *mkleaf*(**n**, *val*) crée un nœud "nombre" d'étiquette **n** avec un champ contenant la valeur *val* du nombre.

Grammaire attribuée réalisant cette construction.

Règle	**Action**
$E \rightarrow E + T$	$E.ptr$:= *mknode*('+',$E_1.ptr$, $T.ptr$)
$E \rightarrow E - T$	$E.ptr$:= *mknode*('-',$E_1.ptr$, $T.ptr$)
$E \rightarrow T$	$E.ptr$:= $T.ptr$
$T \rightarrow (E)$	$T.ptr$:= $E.ptr$
$T \rightarrow$ **id**	$T.ptr$:= mkleaf(**id**, **id**.*ent*)
$T \rightarrow$ **n**	$T.ptr$:= mkleaf(**n**, **n**.*val*)

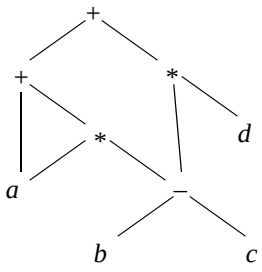
Tous les attributs sont synthétisés.

Construction de l'arbre abstrait de $a - 4 + 5$



On peut obtenir une représentation plus compacte en modifiant les fonctions *mkenode* et *mkleaf* ; si le nœud existe déjà, on renvoie un pointeur vers celui-ci.
On a ainsi une représentation par **graphe orienté acyclique**.

Exemple : $a + a * (b - c) + (b - c) * d$



4.3 Les définitions S-attribuées

Une grammaire est **S-attribuée** lorque tous les attributs sont synthétisés.

L'analyse ascendante se prête bien à l'évaluation des attributs d'une grammaire S-attribuée.

A tout état correspond une lettre (terminal ou variable) par laquelle on atteint cet état. Dans la pile, on peut donc faire le calcul des attributs synthétisés, la valeur des attributs nécessaires se trouvant à une place relative fixe du sommet de la pile

Si on a la règle $A \rightarrow XYZ$ avec l'action $A.a := f(X.x, Y.y, Z.z)$, avant réduction, on trouve les valeurs $X.x$, $Y.y$, $Z.z$ comme $val[top-2]$, $val[top-1]$ et $val[top]$; Après réduction, on place $A.a$ comme nouvelle $val[top]$.

Exemple : expressions arithmétiques et calculette.

Règle	Action
0 $A \rightarrow E\$$	$print(val[top])$
1 $E \rightarrow E + T$	$val[ntop] := val[top-2] + val[top]$
2 $E \rightarrow T$	
3 $T \rightarrow T * F$	$val[ntop] := val[top-2] * val[top]$
4 $T \rightarrow F$	
5 $F \rightarrow (E)$	$val[ntop] := val[top-1]$
6 $F \rightarrow$ **n**	$val[ntop] :=$ **n**$.vallex$

Dans le membre de droite, *top* est le sommet de la pile avant réduction.

Dans le membre de gauche, *ntop* est le sommet de la pile après réduction.

L'automate *LR* est celui qu'on a déjà construit. La lecture d'un nombre provoque l'empilement de sa valeur (seul terminal ayant un attribut).

Exemple : (3 + 2) * 4 *$*

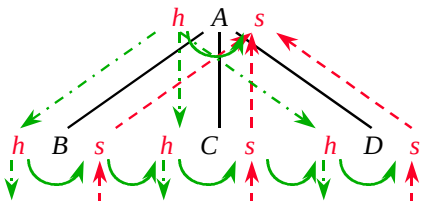
État	Valeur	Mot restant
0	*	(3 + 2) * 4 *$*
0 4	**	3 + 2) * 4 *$*
0 4 5	**3	+ 2) * 4 *$*
0 4 3	**3	+ 2) * 4 *$*
0 4 2	**3	+ 2) * 4 *$*
0 4 8	**3	+ 2) * 4 *$*
0 4 8 6	**3*	2) * 4 *$*
0 4 8 6 5	**3*2	) * 4 *$*
0 4 8 6 3	**3*2	) * 4 *$*
0 4 8 6 9	**3*2	) * 4 *$*
0 4 8	**5	) * 4 *$*
0 4 8 11	**5*	* 4 *$*
0 3	*5	* 4 *$*
0 2	*5	* 4 *$*

État	Valeur	Mot restant
0 2 7	*5*	4 *$*
0 2 7 5	*5*4	*$*
0 2 7 10	*5*4	*$*
0 2	*20	*$*
0 1	*20* *print*(20)	*$*

Le signe * indique qu'il n'y a pas d'attribut correspondant.
Une seule pile contenant des couples état-valeur suffit.

4.4 Les grammaires L-attribuées

Les attributs hérités ne dépendent que des attributs hérités de la variable en partie gauche de règle et des attributs des symboles se trouvant avant.
De la sorte, le calcul des attributs peut être fait par un parcours en profondeur de l'arbre de dérivation, avec priorité de gauche à droite.



Schémas de traduction :

On insère les actions dans les règles comme suit.

1. Un attribut hérité d'une variable en partie droite de règle est calculé par une action intervenant avant cette variable ;
2. Une action ne peut faire référence à un symbole placé à droite de cette action ;
3. Un attribut synthétisé de la variable en partie gauche de règle est calculé après que tous les arguments dont il dépend ont été calculés (l'action est en général placée en fin de règle).

Les grammaires L-attribuées peuvent être converties en schémas de traduction.

Exemple 1 : traduction en forme suffixe des expressions additives :

$E \rightarrow TR$
$R \rightarrow$ **addop** $T\{print$ (**addop**.*lexème*)$\}R \mid \varepsilon$
$T \rightarrow$ **n** $\{print($**n**.*vallex*$)\}$

Exemple2 : incorrect :

$S \rightarrow AA$ $\{A_1.he := 1 ; A_2.he := 2\}$
$A \rightarrow a$ $\{print(A.he)\}$

Dans le parcours de l'arbre l'ordre d'écriture est lancé avant que son paramètre soit défini.
Il faut intercaler le calcul des attributs hérités avant les A.

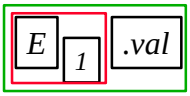
Exemple 3 : le langage de composition de formules EQN

C'est un langage de description de formules mathématiques.

L'objet de base est la **boîte** *B*. Ses attributs sont la **taille** de base du corps des caractères qu'elle contient, *B.ps*, attribut hérité et sa **hauteur** *B.ht*, attribut synthétisé.

On utilise les fonctions *shrink* qui rétrécit la taille suivant un facteur d'échelle (pour indices) et *disp* quicalcule la hauteur d'une boîte obtenue lors d'une mise en indice.

Par exemple, E sub 1 .val spécifie, dans un environnement 12 points, la composition suivante :



On peut construire la grammaire L-attribuée et le schéma de traduction suivants :

Règle	**Action**
$S \rightarrow B$	$B.ps := 10$
	$S.ht := B.ht$
$B \rightarrow BB$	$B_1.ps := B.ps$
	$B_2.ps := B.ps$
	$B.ht := max(B_1.ht, B_2.ht)$
$B \rightarrow B$ **sub** B	$B_1.ps := B.ps$
	$B_2.ps := shrink(B.ps)$
	$B.ht := max(B_1.ht, B_2.ht)$
$B \rightarrow$ **texte**	$B.ht :=$ **text**$.h * B.ps$

Schéma de traduction
$S \rightarrow B$ $\{B.ps := 10\}$ $\{S.ht := B.ht\}$
$B \rightarrow B$ $\{B_1.ps := B.ps\}$ B $\{B_2.ps := B.ps\}$ B $\{B.ht := max(B_1.ht, B_2.ht)\}$
$B \rightarrow B$ $\{B_1.ps := B.ps\}$ **sub** $\{B_2.ps := shrink(B.ps)\}$ B $\{B.ht := max(B_1.ht, B_2.ht)\}$
$B \rightarrow$ **texte** $\{B.ht :=$ **text**$.h * B.ps\}$

4.5 La traduction descendante

Implantation de la traduction descendante pour les grammaires S-attribuées.

Problème : Élimination de la récursivité à gauche.

Grammaire résultante L-attribuée, sous forme de schéma de traduction, par introduction d'une nouvelle variable.

Principe :

$A \rightarrow AY$	$\{A.a := g(A_1.a, Y.y)\}$
$A \rightarrow X$	$\{A.a := f(X.x)\}$

Devient :

$A \rightarrow$	X	$\{R.he := f(X.x)\}$
	R	$\{A.a := R.s\}$
$R \rightarrow$	Y	$\{R_1.he := g(R.he, Y.y)\}$
	R	$\{R.s := R_1.s\}$
$R \rightarrow$	ε	$\{R.s := R.he\}$

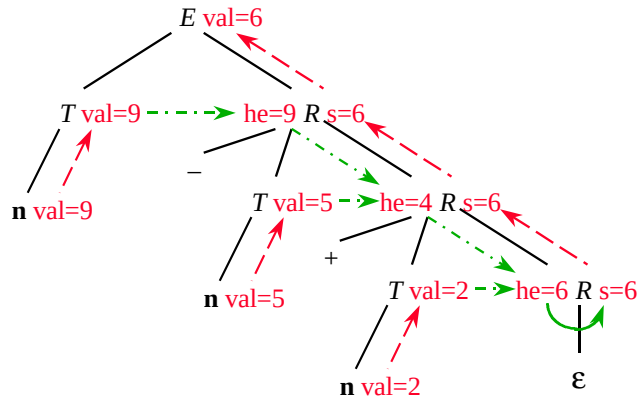
Exemple :

$E \rightarrow E + T$	$\{E.val := E_1.val + T.val\}$
$E \rightarrow E - T$	$\{E.val := E_1.val - T.val\}$
$E \rightarrow T$	$\{E.val := T.val\}$
$T \rightarrow (E)$	$\{T.val := E.val\}$
$T \rightarrow \mathbf{n}$	$\{T.val := \mathbf{n}.val\}$

Devient :

$E \rightarrow$	T	$\{R.he := T.val\}$
	R	$\{E.val := R.s\}$
$R \rightarrow$	$+T$	$\{R_1.he := R.he + T.val\}$
	R	$\{R.s := R_1.s\}$
$R \rightarrow$	$-T$	$\{R_1.he := R.he - T.val\}$
	R	$\{R.s := R_1.s\}$
$R \rightarrow$	ε	$\{R.s := R.he\}$
$T \rightarrow (E)$		$\{T.val := E.val\}$
$T \rightarrow \mathbf{n}$		$\{T.val := \mathbf{n}.val\}$

Évaluation de 9 – 5 + 2 :



Méthode générale de programmation.

Donnée : Un schéma de traduction non récursif à gauche associé à une grammaire L-attribuée. On suppose qu'il n'existe qu'un attribut synthétisé par variable.

Résultat : Code pour un traducteur.

Méthode :

1. Pour chaque variable *A* construire une fonction qui retourne la valeur de l'attribut synthétisé de *A*. Ses paramètres sont les attributs hérités de *A*. Pour chaque attribut de chaque variable apparaissant en membre droit d'une production de *A*, on a une variable locale.

2. Le code pour *A* décide de la règle en fonction du symbole de prévision (grammaire LL(1)).

3. Le code associé à une règle effectue les actions suivantes :

(a) Pour un terminal (unité lexicale) X avec attribut synthétisé $X.x$, sauver sa valeur courante dans la variable locale créée pour cet attribut. Vérifier ensuite la validité de ce terminal en le lisant dans le flot d'entrée.

(b) Pour une variable B, affecter $c := B(b_1, b_1, ..., b_k)$ où $b_1, b_1, ..., b_k$ sont les variables locales créées pour les attributs hérités de B et c est celle pour l'attribut synthétisé de B.

(c) Pour une action, copier son code à sa place, en remplaçant chaque référence à un attribut par la variable locale correspondante, sauf pour la dernière, qui retourne la valeur de l'attribut synthétisé de A et qui retourne donc la valeur de la fonction associée.

Exemple : grammaire précédente, augmentée de $A \rightarrow E\$$

```
function E() ;
 var tval, rhe, rs : integer ;
 begin  tval := T() ; rhe := tval ; rs := R(rhe) ; E := rs
 end ;
function R(x : integer) ;
 var tval, r1he, r1s : integer ;
 begin
  if prevision = '+'  then {regle R > +TR}
    begin correspond('+') ; tval := T(); r1he := x + tval ; r1s := R(r1he) ; R := r1s
    end
  else if prevision = '–' then {regle R > –TR}
    begin correspond('–') ; tval := T(); r1he := x – tval ;  r1s := R(r1he) ; R := r1s
    end
  else R := x {regle R > epsilon}
 end;
```

```
functionT() ;
  var eval,  nval: integer ;
  begin
    if prevision = '('  then {regle T > (E)}
      begin correspond('(') ; eval := E() ; correspond(')') ; T := eval
      end
    else if prevision = nombre then {regle T > n}
      begin nval := nombre_val ; correspond(nombre) ; T := nval
      end
    else writeln('erreur syntaxique')
  end ;
procedure A ;
  var eval : integer ;
  begin  eval := E() ; correspond('$') ; println(eval)
  end ;
```

On peut optimiser ce code, mais attention à l'ordre d'évaluation !

4.6 L'évaluation ascendante

Problème : les actions ne peuvent être effectuées qu'en fin de règle, lorsqu'on réduit. La méthode présentée permet de traiter toutes les grammaires L-attribuées associées à une grammaire LL(1), et un certain nombre d'autres grammaires LR(1) (mais pas toutes !) par une méthode ascendante.

Élimination des actions intérieures

On insère des **marqueurs** se récrivant uniquement comme mot vide, prenant en charge les actions.

Exemple :

Schéma de traduction imprimant la forme postfixée d'une expression avec sommes et différences.

$E \rightarrow TR$
$R \rightarrow +T\{print('+')\}R \mid -T\{print('-')\}R \mid \varepsilon$
$\quad \rightarrow$ **n** $\{print(\mathbf{n}.val)\}$

Devient :

$E \rightarrow TR$
$R \rightarrow +TMR \mid -TNR \mid \varepsilon$
$M \rightarrow \varepsilon\ \{print('+')\}$
$N \rightarrow \varepsilon\ \{print('-')\}$
$\quad \rightarrow$ **n** $\{print(\mathbf{n}.val)\}$

De la sorte, les actions peuvent être effectuées juste avant la réduction.

Attributs hérités sur la pile d'analyse

Cas fréquent : attribut hérité = copie d'un attribut synthétisé déjà présent sur la pile. On peut donc utiliser cet attribut à la place de l'attribut hérité

Exemple : déclaration de type de variables.

$D \rightarrow$	T	$\{L.he := T.type\}$
	L	
$T \rightarrow$	**int**	$\{T.type := integer\}$
$T \rightarrow$	**real**	$\{T.type := real\}$
$L \rightarrow$		$\{L_1.he := L.he\}$
	L **,** **id**	$\{addtype(\mathbf{id}.ent, L.he)\}$
$L \rightarrow$	**id**	$\{addtype(\mathbf{id}.ent, L.he)\}$

Ici, *L.he* est toujours une recopie de *T.type*, juste à côté dans la pile.

Pour simplifier, les états sont remplacés par les variables.

état	flot d'entrée	règle utilisée
-	**int** *p,q,r*	
int	*p,q,r*	
T	*p,q,r*	$T \rightarrow$ **int**
T p	*,q,r*	
T L	*,q,r*	$L \rightarrow$ **id**
T L ,	*q,r*	
T L , q	*,r*	
T L	*,r*	$L \rightarrow L$,**id**
T L ,	*r*	
T L , r		
T L		$L \rightarrow L$,**id**
D		$D \rightarrow TL$

Programme associé :

Règle	Programme
$D \rightarrow TL$	
$T \rightarrow$ **int**	*val*[*ntop*] := *integer*
$T \rightarrow$ **real**	*val*[*ntop*] := *real*
$L \rightarrow L$ **, id**	*addtype*(*val*[*top*] , *val*[*top* – 3])
$L \rightarrow$ **id**	*addtype*(*val*[*top*] , *val*[*top* – 1])

Ceci n'est possible que si l'on sait où dans la pile se trouve la valeur de l'attribut nécessaire.

Exemple :

Règle	Action
$S \rightarrow aAC$	*C.h* := *A.s*
$S \rightarrow bABC$	*C.h* := *A.s*
$C \rightarrow c$	*C.s* := *g*(*C.h*)

C.h est une copie de *A.s*.
Mais on ne sait pas si la valeur cherchée se trouve en *val*[*top* – 1] ou en *val*[*top* – 2].

Règle	Action
$S \rightarrow aAC$	$C.h := A.s$
$S \rightarrow bABMC$	$M.h := A.s$; $C.h := M.s$
$C \rightarrow c$	$C.s := g(C.h)$
$M \rightarrow \varepsilon$	$M.s := M.h$

De la sorte, la valeur de $C.h$ est toujours la valeur de l'attribut synthétisé de la variable immédiatement à sa gauche. Elle se trouve donc toujours en $top - 1$ lorsqu'on en a besoin dans $C.s := g(C.h)$.

Règle	Programme
$S \rightarrow aAC$	
$S \rightarrow bABMC$	
$C \rightarrow c$	$val[ntop] := g(val[top - 1])$
$M \rightarrow \varepsilon$	$val[ntop] := val[top - 1]$

On peut aussi utiliser des marqueurs lorsque les règles ne sont pas seulement des règles de recopie.

Exemple :

$S \rightarrow aAC$ $\quad C.h := f(A.s)$

devient :

$S \rightarrow aAMC$ $\quad M.h := A.s$; $C.h := M.s$

$M \rightarrow \varepsilon$ $\quad M.s := f(M.h)$

Exemple : Retour à EQN, avec incorporation de marqueurs.

Originale

Règle	Action
$S \rightarrow B$	$B.ps := 10$
	$S.ht := B.ht$
$B \rightarrow BB$	$B_1.ps := B.ps$
	$B_2.ps := B.ps$
	$B.ht := max(B_1.ht, B_2.ht)$
$B \rightarrow B$ **sub** B	$B_1.ps := B.ps$
	$B_2.ps := shrink(B.ps)$
	$B.ht := disp(B_1.ht, B_2.ht)$
$B \rightarrow$ **texte**	$B.ht :=$ **texte**$.h * B.ps$

Avec marqueurs

Règle	Action
$S \rightarrow LB$	$B.ps := L.s$
	$S.ht := B.ht$
$L \rightarrow \varepsilon$	$L.s := 10$
$B \rightarrow BMB$	$B_1.ps := B.ps$
	$M.h := B.ps$
	$B_2.ps := M.s$
	$B.ht := max(B_1.ht, B_2.ht)$
$B \rightarrow B$ **sub** NB	$B_1.ps := B.ps$
	$N.h := B.ps$
	$B_2.ps := N.s$
	$B.ht := disp(B_1.ht, B_2.ht)$
$B \rightarrow$ **texte**	$B.ht :=$ **texte**$.h * B.ps$
$M \rightarrow \varepsilon$	$M.s := M.h$
$N \rightarrow \varepsilon$	$N.s := shrink(N.h)$

D'où le programme suivant :

Règle	Programme
$S \rightarrow LB$	$val[ntop] := val[top]$
$L \rightarrow \varepsilon$	$val[ntop] := 10$
$B \rightarrow BMB$	$val[ntop] := max(val[top - 2] , val[top])$
$B \rightarrow B$ **sub** NB	$val[ntop] := disp(val[top - 3] , val[top])$
$B \rightarrow$ **texte**	$val[ntop] := val[top] * val[top - 1]$
$M \rightarrow \varepsilon$	$val[ntop] := val[top - 1]$
$N \rightarrow \varepsilon$	$val[ntop] := shrink(val[top - 2])$

Algorithme général :

Donnée : Définition L-attribuée avec grammaire $LL(1)$;
(si grammaire $LR(1)$, elle peut ne pas le rester)
Résultat : Analyseur calculant les attributs sur la pile ;

Règles de la forme : $A \rightarrow X_1 X_2 \ldots X_n$; pour simplifier,un seul attribut hérité par variable, $A.h$ et un seul attribut synthétisé par symbole, $X.s$.

Règle	Actions
$A \rightarrow X_1 X_2 \ldots X_k \ldots X_n$;	$\ldots, X_k.h := f_k(A.h,\ldots, X_i.h, X_i.s,\ldots), \ldots ,$ $A.s := g(A.h,\ldots, X_i.h, X_i.s,\ldots)$

On remplace par : $A \rightarrow M_1 X_1 M_2 X_2 \ldots M_n X_n$; L'attribut $X_k.h$ est remplacé par un attribut synthétisé $M_k.s$.

Règles	Actions
$A \rightarrow M_1 X_1 M_2 X_2 \ldots M_k X_k \ldots M_n X_n$;	$\ldots, X_k.h := M_k.s,\ldots$ $A.s := g(A.h,\ldots, M_i.s, X_i.s,\ldots)$ $M_k.h := f_k(A.h,\ldots, M_i.s, X_i.s,\ldots)$
$M_k \rightarrow \varepsilon$;	$M_k.s := M_k.h$

On peut montrer que l'attribut hérité $A.h$ est toujours juste avant la position de M_1. En effet, $X_k.h$ ne sert qu'au calcul de $X_k.s$ et est obtenu par recopie de l'attribut immédiatement précédent.

Quand on réduit ε à M_k, on trouve $A.h$ à $val[top - 2k + 2]$, puis $M_1.s$ à $val[top - 2k + 3]$, $X_1.s$ à $val[top - 2k + 4]$ etc.

Remarques : Si X_k n'a pas d'attribut hérité (cas des terminaux), il est inutile d'introduire M_k.
On peut souvent se passer de M_1, surtout si l'attribut $X_1.h$ s'obtient par une règle de recopie de $A.h$.
Dans ces cas, les valeurs précédentes des places des attributs ne sont plus applicables.

4.7 L'évaluation récursive

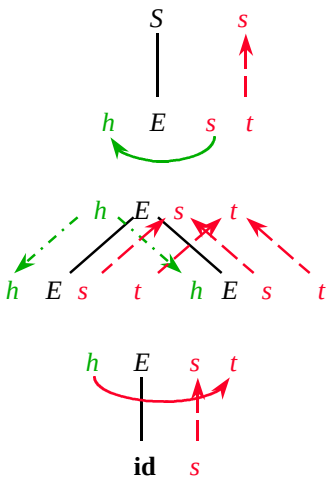
Parfois, l'évaluation ne peut pas être faite en une passe.
L'arbre d'analyse est alors construit explicitement et l'évaluation des attributs peut être faite si le graphe de dépendance des attributs est acyclique.

On peut évaluer les attributs de manière récursive si l'ordre d'évaluation de ceux-ci est le même dans toutes les productions d'une même variable.

L'exemple qui suit comporte deux attributs synthétisés s et t et un hérité h. Le cadre est celui de la surcharge des types d'un identificateur. Le premier, s, représente l'ensemble des types possibles, h est l'information contextuelle et t est le type final déterminé par les deux premiers.

Règle	Action
$S \rightarrow E$	$E.h := g(E.s)$
	$S.s := E.t$
$E \rightarrow EE$	$E.s := fs(E_1.s, E_2.s)$
	$E_1.h := fh1(E.h)$
	$E_2.h := fh2(E.h)$
	$E.t := ft(E_1.t, E_2.t)$
$E \rightarrow$ **id**	$E.s :=$ **id**$.s$
	$E.t := l(E.h)$

Graphes de dépendances



Les attributs de E peuvent toujours être évalués dans l'ordre s, h, t.

Par conséquent, la procédure pour calculer $E.t$ doit avoir un paramètre pour $E.h$.

D'où le pseudo-code suivant, où n désigne un nœud de l'arbre d'analyse et $Fils(n,i)$ est le i-ième fils de n.

```
fonction Es(n) ;
  début selon que la production au nœud n vaut
    E → EE : s1 := Es(Fils(n,1)) ; s2 := Es(Fils(n,2)) ; retour fs(s1,s2) ;
    E → id : retour id.s ;
    autre : Erreur
    fin
  fin ;

fonction Et(n, h) ;
  début selon que la production au nœud n vaut
    E → EE : h1 := fh1(h) ; t1 := Et(Fils(n,1), h1) ;
                h2 := fh2(h) ; t1 := Et(Fils(n,2), h2) ; retour ft(t1,t2) ;
    E → id : retour l(h) ;
  autre : Erreur
    fin
  fin ;

fonction Ss(n) ;
  début s := Es(Fils(n,1)) ; h := g(s) ; t := Et(Fils(n,1),h) ; retour t
  fin ;
```