Traduction dirigée par la syntaxe

1. Grammaires attribuées
2. Exemple d'attributs synthétisés
3. Exemple d'attributs hérités
4. Arbres syntaxiques
5. Définitions S-attribuées
6. Définitions L-attribuées
   (a) schémas de traduction
   (b) évaluation descendante
   (c) évaluation ascendante

Introduction

Les méthodes présentées ici permettent d'incorporer la traduction à l'analyse syntaxique.

L'idée de base est d'ajouter aux règles de la grammaire des *actions sémantiques* portant sur des *attributs* des non-terminaux de la grammaire.

On obtient ainsi des modules de traduction en une seule passe incorporant l'analyse syntaxique.

La méthode peut être utilisée en analyse descendante. Elle consiste alors essentiellement à utiliser de vraies fonctions avec valeurs et arguments au lieu des procédures de la descente récursive.

Elle peut aussi être utilisée en analyse ascendante. Elle peut être en général implémentée en YACC.

Grammaires attribuées

$$\text{Variables} \to \text{attributs} \left| \begin{array}{l} \textit{synthétisés} \\ \textit{hérités} \end{array} \right.$$

Regle	Action
$A \to \alpha$	$b := f(c_1, c_2, \ldots, c_k)$

Le symbole b est un attribut:

1. *synthétisé*: b est un attribut de A et les c_i sont des attributs des symboles formant α

   ou

2. *hérité*: b est un attribut d'un des symboles formant α et les c_i sont des attributs de A ou des symboles formant α

Le symbole f désigne une fonction.

Exemple d'attributs synthétisés

Grammaire attribuée pour une calculette basée sur la grammaire ETF:

REGLE	ACTION
$L \rightarrow E\ \mathbf{n}$	$print(E.val)$
$E \rightarrow E + T$	$E.val := E_1.val + T.val$
$E \rightarrow T$	$E.val := T.val$
$T \rightarrow T * F$	$T.val := T_1.val \times F.val$
$T \rightarrow F$	$T.val := F.val$
$F \rightarrow (E)$	$F.val := E.val$
$F \rightarrow \mathbf{chiffre}$	$F.val := \mathbf{chiffre}.lexval$

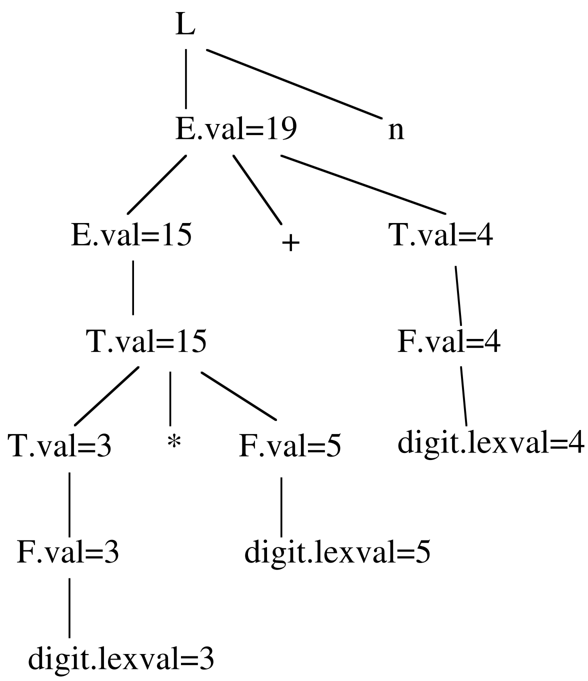

Figure 1: L'arbre décoré de 3*5+4

L'arbre représenté sur la Figure s'appelle *l'arbre décoré*: c'est l'arbre de dérivation sur lequel on ajoute les valeurs des attributs.

Exemple d'attributs hérités

Déclarations de variables dans le style du langage C.

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

L'attribut $L.he$ est hérité: c'est le type des variables constituant la liste L.

Un exemple développé: la construction de l'arbre syntaxique d'une expression

Trois fonctions utilisées:

1. $mkn(op, left, right)$: crée un noeud d'operande d'étiquette *op* avec deux champs contenant des pointeurs vers $left$ et $right$.
2. mkl(**id**,$entree$): crée un noeud identificateur avec un etiquette **id** et un champ contenant $entree$, pointeur vers la table des symboles.
3. mkl(**num**,val): crée un noeud nombre avec l'étiquette **num** et un champ contenant val, la valeur du nombre.

Exemple de calcul:

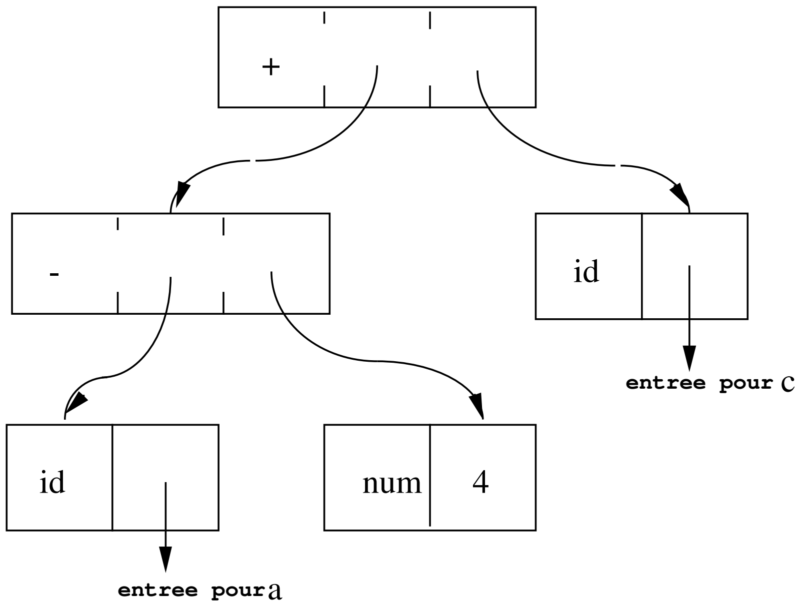

Figure 2: L'arbre syntaxique de $a - 4 + c$

Grammaire attribuée pour les expressions additives avec deux attributs synthétisés $E.ptr$ et $T.ptr$ donnant des pointeurs sur les arbres syntaxiques correspondants.

REGLE	ACTION
$E \rightarrow E + T$	$E.ptr :=$ $mknode('+', E_1.ptr, T.ptr)$
$E \rightarrow E - T$	$E.ptr :=$ $mknode('-', E_1.ptr, T.ptr)$
$E \rightarrow T$	$E.ptr := T.ptr$
$T \rightarrow (E)$	$T.ptr := E.ptr$
$T \rightarrow \mathbf{id}$	$T.ptr :=$ $mkleaf(\mathbf{id}, \mathbf{id}.entree)$
$T \rightarrow \mathbf{num}$	$T.ptr :=$ $mkleaf(\mathbf{num}, \mathbf{num}.entree)$

Evaluation ascendante des grammaires S-attribuées

Grammaires S-attribuées: tous les attributs sont synthétisés.

Le calcul des attributs peut de faire dans la pile de l'analyseur ascendant:

état	valeur
$\dots$	$\dots$
X	$X.x$
Y	$Y.y$
Z	$Z.z$
$\dots$	$\dots$

Si la règle est $A.a := f(X.x, Y.y, Z.z)$, on pourra calculer la valeur de l'attribut de A avant de réduire XYZ en A en fonction des valeurs contenues en $val[top-2], val[top-1]$ et $val[top]$. L'exemple de la calculette se réecrit ainsi:

Règle	Instruction
$L \to E\mathbf{n}$	$print(val[top])$
$E \to E + T$	$val[ntop] :=$ $val[top-2] + val[top]$
$E \to T$	
$T \to T + F$	$val[ntop] :=$ $val[top-2] \times val[top]$
$T \to F$	
$F \to (E)$	$val[ntop] := val[top-1]$
$F \to$**chiffre**	

Par exemple, si on analyse l'expression $3 * 5 + 4$, on obtient la suite de mouvements:

Input	Etat	Valeur	Règle
$3 * 5 + 4\mathbf{n}$	_	_	
$*5 + 4\mathbf{n}$	3	3	
$*5 + 4\mathbf{n}$	F	3	$F \to$**chiffre**
$*5 + 4\mathbf{n}$	T	3	$T \to F$
$5 + 4\mathbf{n}$	$T*$	3 _	
$+4\mathbf{n}$	$T * 5$	3 _ 5	
$+4\mathbf{n}$	$T * F$	3 _ 5	$F \to$**chiffre**
$+4\mathbf{n}$	T	15	$T \to T * F$

Grammaires L-attribuées

Principe: Le calcul des attributs est effectué par un parcours *en profondeur* de l'arbre de dérivation suivant le schéma:

procedure $parcours\ (n : noeud)$:
debut
  pour chaque fils m de n, de gauche à droite
    faire debut
      evaluer les attributs hérités de m;
      $parcours(m)$
  fin;
  évaluer les attributs synthétisés de n
fin

Définition des grammaires L-attribuées

Pour chaque règle $A \to X_1 X_2 \ldots X_n$, les attributs sont ou bien synthétisés, ou bien, si c'est un attribut de X_j, il ne dépend que de:

1. les attributs des variables $X_1, X_2, \ldots, X_{j-1}$ à gauche de X_j et
2. les attributs hérités de A.

Schemas de traduction

Principe: les actions sont insérées dans les règles de la grammaire avec les règles suvantes:

1. Un atribut hérité concernant une variable du membre droit de la règle doit être calculé par une action intervenant avant son occurrence.
2. Une action ne peut pas faire référence à un symbole placé à droite de cette action.
3. Un attribut synthétisé de la variable à gauche ne peut être calculé qu'après que tous les arguments de sa fonction aient été déja calculés (l'action correspondante sera souvent placée à droite à la fin de la règle)

Exemple 1: Traduction en forme suffixe des expressions additives.

$$\begin{array}{lcl} E & \rightarrow & T R \\ R & \rightarrow & \textbf{addop}\, T\{print(\textbf{addop.}lexeme)\}R \mid \epsilon \\ T & \rightarrow & \textbf{num}\{print(\textbf{num}.val)\} \end{array}$$

Exemple 2: Un schéma qui ne satisfait pas aux conditions:

$$\begin{array}{lcl} S & \rightarrow & AA\,\{A_1.he := 1; A_2.he := 2\} \\ A & \rightarrow & a\,\{print(A.he)\} \end{array}$$

La première condition n'est pas satisfaite: l'attribut hérité $A.he$ n'est pas défini lorsqu'un ordre d'écriture est lancé durant une visite en profondeur de l'arbre de dérivation de aa.

Le langage EQN

Le langage EQN est un langage de composition de texte scientifique permettant d'écrire des formules mathématiques. On considère la grammaire L-attribuée décrivant les *boîtes* qui contiennent ces formules. Les boîtes sont représentées par une variable B. La construction $B \to B\,\mathbf{sub}\,B$ représente la disposition de deux boites en indice l'une de l'autre avec la réduction de taille convenable pour la seconde.

On utilise deux attributs: $B.ps$ qui dénote le *corps de caractères*. C'est un attribut hérité. Et $B.ht$, qui est un attribut synthétisé notant la *hauteur* de la boîte.

On utilise aussi deux procedures: $shrink$ qui permet de réduire la taille d'une boite suivant un facteur d'echelle, et $dispose$ qui donne la hauteur d'une boite faite par mise en indice.

Voici la grammaire L-attribuée:

Règle	Action
$S \to B$	$B.bs := 10$
	$S.ht := B.ht$
$B \to B\ B$	$B_1.ps := B.ps$
	$B_2.ps := B.ps$
	$B.ht := max(B_1.ht, B_2.ht)$
$B \to B\ \mathbf{sub}\ B$	$B_1.ps := B.ps$
	$B_2.ps := shrink(B.ps)$
	$B.ht := disp(B_1.ht, B_2.ht)$
$B \to \mathbf{text}$	$B.ht := \mathbf{text}.h \times B.ps$

Et le schéma de traduction:

$$
\begin{array}{lll}
S \to & & \{B.ps := 10\} \\
 & B & \{S.ht := B.ht\} \\
B \to & & \{B_1.ps := B.ps\} \\
 & B & \{B_2.ps := B.ps\} \\
 & B & \{B.ht := max(B_1.ht, B_2.ht)\} \\
B \to & & \{B_1.ps := B.ps\} \\
 & B & \\
 & \mathbf{sub} & \{B_2.ps := shrink(B.ps)\} \\
 & B & \{B.ht := disp(B_1.ht, B_2.ht)\} \\
B \to & \mathbf{text} & \{B.ht := \mathbf{text}.h \times B.ps\}
\end{array}
$$

Elimination de la récursivité à gauche

En vue de la traduction en analyse descendante, on étend aux schémas de traduction la méthode d'élimination de la récursivité à gauche.

Exemple: Le schéma de traduction suivant avec une grammaire récursive à gauche:

$$
\begin{array}{ll}
E \rightarrow E + T & \{E.val := E_1.val + T.val\} \\
E \rightarrow E - T & \{E.val := E_1.val - T.val\} \\
E \rightarrow T & \{E.val := T.val\} \\
T \rightarrow (E) & \{T.val := E.val\} \\
T \rightarrow \mathbf{num} & \{E.val := \mathbf{num}.val\}
\end{array}
$$

Est transformé en:

$$
\begin{array}{lll}
E \rightarrow & T & \{R.he := T.val\} \\
 & R & \{E.val := R.s\} \\
R \rightarrow & + & \\
 & T & \{R_1.he := R.he + T.val\} \\
 & R & \{R.s := R_1.s\} \\
R \rightarrow & - & \\
 & T & \{R_1.he := R.he - T.val\} \\
 & R & \{R.s := R_1.s\} \\
R \rightarrow & \epsilon & \{R.s := R.he\} \\
T \rightarrow & (& \\
 & E & \\
 &) & \{T.val := E.val\} \\
T \rightarrow & \mathbf{num} & \{T.val := \mathbf{num}.val\}
\end{array}
$$

L'attribut hérité $R.he$ sert à recueillir la valeur de l'expression située à gauche.

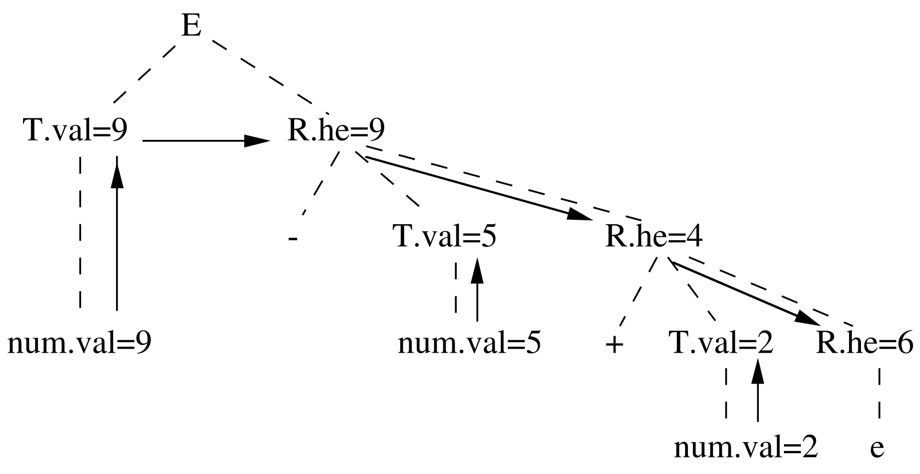

Figure 3: Evaluation de 9-5+2

Principe général: on part du schéma de traduction:

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

Que l'on remplace par:

$$\begin{array}{lll} A \to & X & \{R.he := f(X.x)\} \\ & R & \{A.a := R.s\} \\ R \to & Y & \{R_1.he := g(R.he, Y.y)\} \\ & R & \{R.s := R_1.s\} \\ R \to & \epsilon & \{R.s := R.he\} \end{array}$$

Exemple: On peut appliquer cette méthode au calcul de l'arbre syntaxique. On obtient le schéma suivant:

$$
\begin{array}{lll}
E \rightarrow & T & \{R.he := T.nptr\} \\
 & R & \{E.nptr := R.s\} \\
R \rightarrow & + & \\
 & T & \{R_1.he := \\
 & & mknode(`+', R.he, T.nptr)\} \\
 & R & \{R.s := R_1.s\} \\
R \rightarrow & - & \\
 & T & \{R_1.he := \\
 & & mknode(`-', R.he, T.nptr)\} \\
 & R & \{R.s := R_1.s\} \\
R \rightarrow & \epsilon & \{R.s := R.he\} \\
T \rightarrow & (& \\
 & E & \\
 &) & \{T.nptr := E.nptr\} \\
T \rightarrow & \mathbf{num} & \{T.nptr := \\
 & & mkleaf(\mathbf{num}, \mathbf{num}.val)\}
\end{array}
$$

Méthode générale de construction d'un traducteur descendant

Donnée: Un scéma de traduction associé à une grammaire $LL(1)$.

Résultat: Code pour un traducteur.

Méthode:

1. Pour chaque variable A, construire une fonction ayant un paramètre pour chaque attribut hérité de A et qui rend la valeur des attributs synthétisés de A (on suppose pour simplfier que A n'a qu'un seul attribut synthétisé). On a de plus une variable locale à la fonction pour chaque attirbut de chaque variable en membre droit d'une règle de dérivation de A.
2. Le code pour A décide de la règle à appliquer en fonction du symbole courant du texte.
3. Le code associé à une règle fait les actions suivantes en lisant le membre droit de gauche à droite:
   (a) Pour un lexème X avec attribut synthétisé x, conserver la valeur de x dans la variable locale crée pour $X.x$. Executer ensuite une action de lecture de X.

(b) Pour une variable B, engendrer une instruction $c := B(b_1, b_2, \ldots, b_k)$ où $b_1, b_2, \ldots, b_k$ sont les valeurs des attributs hérités de B et c est la variable pour l'attribut synthétisé de B.

(c) Pour une action, recopier le code.

Exemple

On applique la méthode au calcul de l'arbre syntaxique. On utilise trois fonctions:

function E: $\uparrow$ syntax_tree_node;

function $R(he$: syntax_tree_node$)$: $\uparrow$ syntax_tree_node;

function T : $\uparrow$ syntax_tree_node;

La procedure pour l'analyse de R s'écrit ainsi, en rassemblant deux règles en une:

procedure R;
begin
    if *lookahead* = **addop then begin**
        match(**addop**); T; R;
    end
    else begin /*ne rien faire*/
    end
end;

On obtient la fonction pour R:

function *R(he:* $\uparrow$ syntax_tree_node;
    var *nptr, hl, sl, s*: $\uparrow$ syntax_tree_node;
        addoplexeme: **char**;

begin
    if *lookahead* = **addop then begin**
        /*regle $R \rightarrow$ **addop** T R*/
        addoplexeme := lexval;
        match(**addop**)
        nptr := T;
        hl:= mknode(addoplexeme, he, nptr);
        sl := R(hl);
        s := sl;
    end
    else *s := he;* /* regle $R \rightarrow \epsilon$ */
    return *s*
end

Evaluation ascendante des attributs hérités

Le problème à résoudre est que, en analyse ascendante, on ne peut effectuer les actions qu'en fin de règle, i. e. au moment où on réduit. On va montrer d'abord par des exemples puis par une méthode générale comment procéder.

L'idée générale est d'insérer des marqueurs dans les règles qui, en s'effaçant produisent les actions. On peut alors évaluer en plus les attributs hérités dans la pile.

La méthode présentée permet de traiter toutes les grammaires L-attribuées associées à une grammaire $LL(1)$ et certaines (mais pas toutes) ayant une grammaire sous-jacente $LR(1)$.

Elimination des actions imbriquées dans les règles

On insère des marqueurs dans les règles avec des règles d'effacement réalisant les actions. Par exemple, on remplace le schéma de traduction:

$$
\begin{array}{lcl}
E & \rightarrow & T R \\
R & \rightarrow & +T\ \{print(`+')\}R \mid -T\{print(`-')\} \\
 & & \mid \epsilon \\
T & \rightarrow & \mathbf{num}\ \{print(\mathbf{num}.val)\}
\end{array}
$$

par:

$$
\begin{array}{lcl}
E & \rightarrow & T R \\
R & \rightarrow & +T\ M\ R \mid -T\ N\ R \mid \epsilon \\
T & \rightarrow & \mathbf{num}\{print(\mathbf{num}.val)\} \\
M & \rightarrow & \epsilon\ \{print(`+')\} \\
N & \rightarrow & \epsilon\ \{print(`-')\}
\end{array}
$$

Attributs hérités sur la pile

Les valeurs des attributs hérités qui sont des copies de celle d'un attribut synthétisé d'une autre variable peuvent etre localisés dans la pile.

Exemple: Considérons la grammaire des déclarations de variables

$$
\begin{array}{lll}
D \rightarrow & T & \{L.he := T.type\} \\
 & L & \\
T \rightarrow & \mathbf{int} & \{T.type := integer\} \\
T \rightarrow & \mathbf{real} & \{T.type := real\} \\
L \rightarrow & & \{L_1.he := L.he\} \\
 & L, \mathbf{id} & \{addtype(\mathbf{id}.entree, L.he)\} \\
L \rightarrow & \mathbf{id} & \{addtype(\mathbf{id}.entree, L.he)\}
\end{array}
$$

La suite des mouvements de l'analyseur sur la donnée

int p,q,r

est:

INPUT	état	regle utilisée
int p,q,r	_	
p,q,r	**int**	
p,q,r	T	$T \rightarrow$ **int**
,q,r	T p	
,q,r	$T\ L$	$L \rightarrow$ **id**
q,r	$T\ L$, q	
,r	$T\ L$	$L \rightarrow L, \mathbf{id}$
r	$T\ L$,	
	$T\ L$, r	
	$T\ L$	$L \rightarrow L, \mathbf{id}$
	D	$D \rightarrow T\ L$

Comme on peut trouver la valeur de $L.he$ dans $T.type$ juste à côté dans la pile,on peut implémenter ce schéma en analyse ascendante comme:

REGLE	PROGRAMME
$D \rightarrow TL$	
$T \rightarrow$ **int**	$val[ntop] := integer$
$T \rightarrow$ **real**	$val[ntop] := real$
$L \rightarrow L, \mathbf{id}$	$addtype(val[top], val[top-3])$
$L \rightarrow \mathbf{id}$	$addtype(val[top], val[top-1])$

Un exemple plus difficile

Considérons la grammaire attribuée suivante:

$$\begin{array}{rc}
REGLE & ACTION \\
S \to aAC & C.h := A.s \\
S \to bABC & C.h := A.s \\
C \to c & C.s := g(C.h)
\end{array}$$

En réduisant c en C,on ne peut pas savoir si la valeur de $C.h$ se trouve en $val[top-1]$ ou en $val[top-2]$. Pour pouvoir transmettre l'attribut de A à C, il faut introduire une règle de *recopie*:

$$\begin{array}{rc}
REGLE & ACTION \\
S \to aAC & C.h := A.s \\
S \to bABMC & M.h := A.s; C.h := M.s \\
C \to c & C.s := g(C.h)
\end{array}$$

Les règles de recopie peuvent aussi être utilisées quand il ne s'agit pas seulement de recopie. Par exemple:

$$S \to aAC \;\; C.h := f(A.s)$$

peut être remplacé par:

$$\begin{array}{ll} S \to aANC & N.h := A.s; C.h := N.s \\ N \to \epsilon & N.s := f(N.h) \end{array}$$

Quand on fait la réduction par $N \to \epsilon$ on trouve la valeur de $N.i$ a la place de $A.s$, c'est à dire dans $val[top-1]$.

Exemple: Retour à EQN.

REGLE	ACTION
$S \to LB$	$B.ps := L.s$
	$S.ht := B.ht$
$L \to \epsilon$	$L.s := 10$
$B \to BMB$	$B_1.ps := B.ps$
	$M.h := B.ps$
	$B_2.ps := M.s$
	$B.ht :=$
	$\max(B_1.ht, B_2.ht)$
$B \to$	
B **sub** N B	$B_1.ps := B.ps$
	$N.h := B.ps$
	$B_2.ps := N.s$
	$B.ht :=$
	$disp(B_1.ht, B_2.ht)$
$B \to$ **text**	$B.ht :=$ **text**$.h \times B.ps$
$M \to \epsilon$	$M.s := M.h$
$N \to \epsilon$	$N.s := shrink(N.h)$

Ceci peut maintenant être implémenté comme:

REGLE	PROGRAMME
$S \to L\ B$	$val[ntop] := val[top]$
$L \to \epsilon$	$val[ntop] := 10$
$B \to B\ M\ B$	$val[ntop] :=$ $\max(val[top-2], val[top])$
$B \to$ B **sub** $N\ B$	$val[ntop] :=$ $disp(val[top-3], val[top])$
$B \to$ **text**	$val[ntop] :=$ $val[top] \times val[top-1]$
$M \to \epsilon$	$val[ntop] := val[top-1]$
$N \to \epsilon$	$val[ntop] :=$ $shrink(val[top-2])$

Algorithme général

Donnée: Grammaire L-attribuée associée à une grammaire $LL(1)$ (si la grammaire de départ est $LR(1)$, elle ne le reste pas toujours après ajout des marqueurs).

Résultat: Un analyseur qui calcule les valeurs des attributs sur la pile d'analyse.

Méthode: On suppose que chaque variable A a un attribut hérité $A.h$ et que chaque symbole X a un attribut synthétisé $X.s$.

Remplacer chaque règle $A \to X_1 \dots X_n$ par $A \to M_1 X_1 \dots M_n X_n$. L'attribut $X_j.s$ sera rangé dans la pile avec X_j. L'attribut X_h apparait associé à M_j.

Au cours de l'analyse, l'attribut hérité $A.h$ se trouvera toujours juste au dessous de la place de M_1.

Quand on réduit ϵ à M_j, $A.h$ se trouve en $val[top-2j+2]$, $X_1.h$ est dans $val[top-2j+3]$ et ainsi de suite.

Simplifications: Si X_j n'a pas d'attribut hérité, on n'a pas besoin du marqueur M_j.

Si $X_1.h$ existe mais est calculé par une règle de recopie $X_1.h := A.h$, on peut supprimer M_1.

Conclusion

On peut, en résumé, noter que :

1. Les schémas de traduction permettent d'incorporer la traduction à l'analyse syntaxique pour obtenir un module de traduction en une seule passe.
2. Les attributs synthétisés se calculent aisément
   (a) En analyse descendante comme *valeurs* des fonctions traduisant les non-terminaux de la grammaire.
   (b) En analyse ascendante directement dans la pile d'analyse.
3. Les attributs hérités peuvent s'évaluer :
   (a) En analyse descendante comme *paramètres* des fonctions.
   (b) En analyse ascendante, dans la pile, en introduisant des marqueurs.

	Analyse descendante	Analyse ascendante
Grammaire	LL	LR
Attributs synthétisés	Valeurs des fonctions	Calcul direct dans la pile
Attributs hérités	Paramètres des fonctions	Calcul dans la pile avec marqueurs