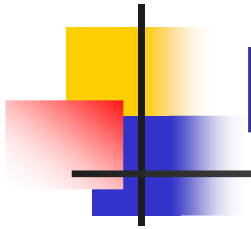
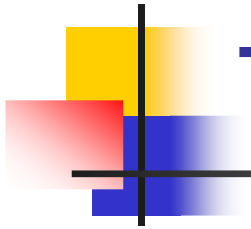
Traduction dirigée par la syntaxe

ESI – École nationale Supérieure en Informatique



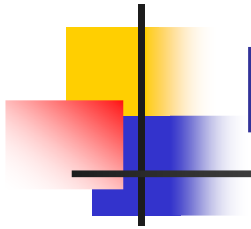
Définition

- Traduction dirigée par la syntaxe i.e. au fur à mesure que se déroule l'analyse syntaxique, on effectue certains traitements.
- Quels traitements ?
- Ces traitements sont communément appelés actions sémantiques ou routines sémantiques.



Traducteur

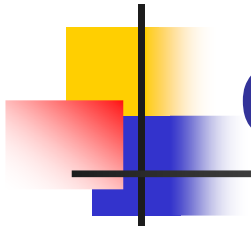
- Un traducteur dirigé par la syntaxe est équivalent à un analyseur syntaxique dans lequel les symboles de la grammaire ont des attributs et des routines sémantiques sont insérées dans le code de cet analyseur



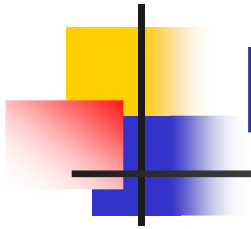
Routines sémantiques

- le contrôle de type, la liaison d'objets et la génération d'une forme intermédiaire, …
- Ces actions ont un "timing" d'exécution.
- C'est au concepteur de les insérer "aux bons endroits" au sein de l'algorithme d'analyse syntaxique.

Résultat de la traduction dirigée par la syntaxe



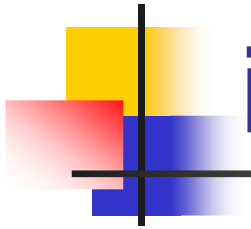
- Plusieurs formes.
  - Pour les évaluateurs d'expressions, tel MS EXCEL, le résultat sera l'évaluation de l'expression mathématique si celle-ci est syntaxiquement correcte.
  - Pour les langages de programmation, la sortie de la traduction est une forme intermédiaire.



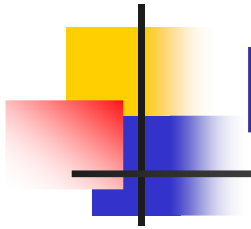
Formes Intermédiaires

- Est une abstraction du programme source :
- Exemples de Formes Intermédiaires :
  - un arbre abstrait (AST Abstract Syntax Tree),
  - un code à trois adresses (quadruplets, triplets, …), …

Pourquoi une forme intermédiaire ?

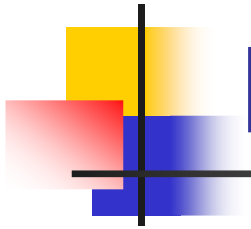


- La transformation vers le langage machine est facilitée. Souvent il existe une bijection entre instruction de la forme intermédiaire et instruction machine.
- La forme intermédiaire est indépendante de la machine i.e. elle n'est pas liée au jeu d'instructions de la machine.
- Des opérations d'optimisation du code produit sont efficacement effectuées sur le code intermédiaire.



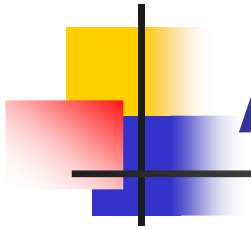
Formes Intermédiaires

- Ruby et Perl sont des exemples de langages de programmation générant un AST (Abstract Syntax Tree).
- Les langages générant un code à trois adresses sont légion.



Formes Intermédiaires

- Certains langages de programmation ne produisent pas un code natif (code machine) mais un code intermédiaire "indépendant de la machine".
- Le langage Java produit du bytecode. L'environnement .Net et Mono produisent du CIL (Common Intermediate Language).

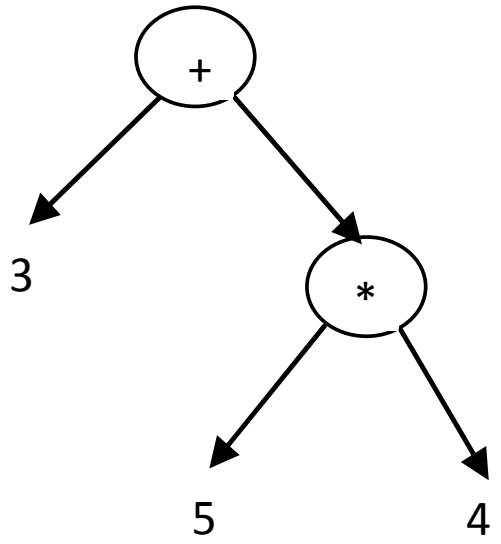


Arbre abstrait

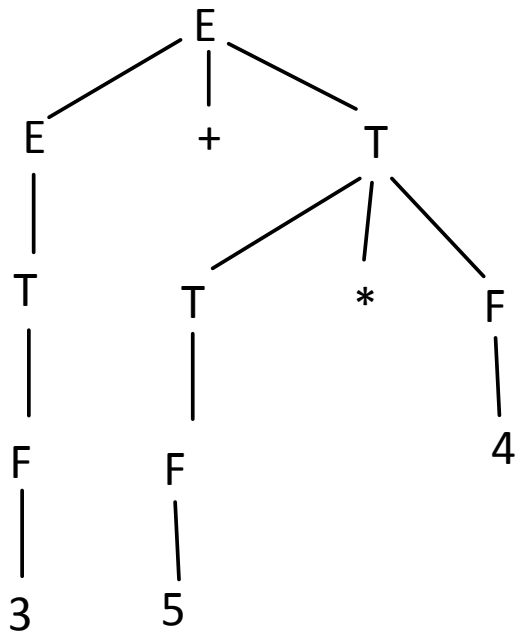
- La représentation sous format d'arbres abstraits est une forme condensée de l'arbre syntaxique.
- L’arbre syntaxique matérialise toutes les dérivations utilisées lors de l’analyse syntaxique.
- L’arbre abstrait ne garde que les informations utiles sous une structure d’arbre

Arbre abstrait

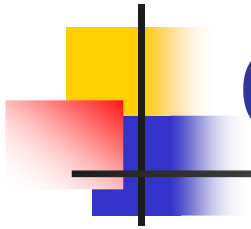
- Exemple :



Arbre abstrait



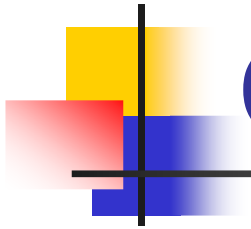
Arbre syntaxique



Code à 3 adresses

- C'est une des formes intermédiaires les plus utilisées. Cette forme est assez proche du jeu d'instructions machine.
- Chaque instruction dans un code à trois adresses est définie par un 4-tuple ou quadruplet de la forme :

(op, source1, source2, destination)



Code à 3 adresses

destination:= source1 op source2

- source1, source2 et destination sont des variables, constantes ou variables temporaires générés par le compilateur.
- Op représente un opérateur tel un opérateur arithmétique ou un branchement.

Code à 3 adresses – Exemple 1

- code à trois adresses de l'expression suivante

a := b * (- c) + b * (- c)

1. tmp1 :=-c
2. tmp2 :=b*tmp1
3. tmp3 :=-c
4. tmp4 :=b*tmp3
5. tmp5 :=tmp2+tmp4
6. a :=tmp5

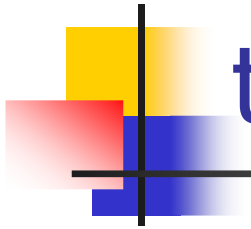
Code à 3 adresses – Exemple 2

- code à trois adresses du fragment de programme C suivant : If (a>=b) x++; else x--;

1. cmp a b
2. JumpLess 4.
3. x:=x+1
4. Jump 6.
5. x:=x-1

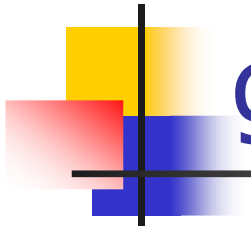
-

Comment effectuer une traduction dirigée par la syntaxe



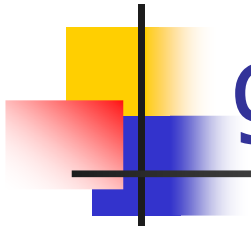
- Définir l'objectif
- Ecrire le code des routines à insérer dans l'algorithme d'analyse syntaxique
- Utiliser les attributs de la grammaire et structures de données adéquates

Attributs des symboles de la grammaire



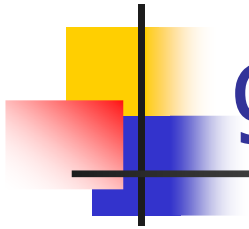
- Pour effectuer une traduction dirigée par la syntaxe efficace chaque symbole de la grammaire utilisé dans une analyse sera muni d’attributs.
- Les attributs d'un symbole peuvent être des champs d'une structure (valeur, type, ...).

Attributs des symboles de la grammaire



- Chaque symbole de la grammaire dans la pile d'analyse "pointera" sur ses attributs.
- Ces attributs peuvent être classés en deux catégories que sont :
  - les attributs hérités
  - et les attributs synthétisés

Attributs des symboles de la grammaire



- **Attribut synthétisé :** Une valeur synthétisée d'un symbole de la grammaire est calculée à partir des descendants de ce symbole dans l'arbre syntaxique.
- **Attribut hérité :** Une valeur héritée d'un symbole de la grammaire est calculée à partir des ascendants et frères de ce symbole dans l'arbre syntaxique.
- **Arbre décoré :** On appelle arbre syntaxique décoré (ou arbre annoté), un arbre syntaxique muni des valeurs d'attributs en chaque sommet de l'arbre.

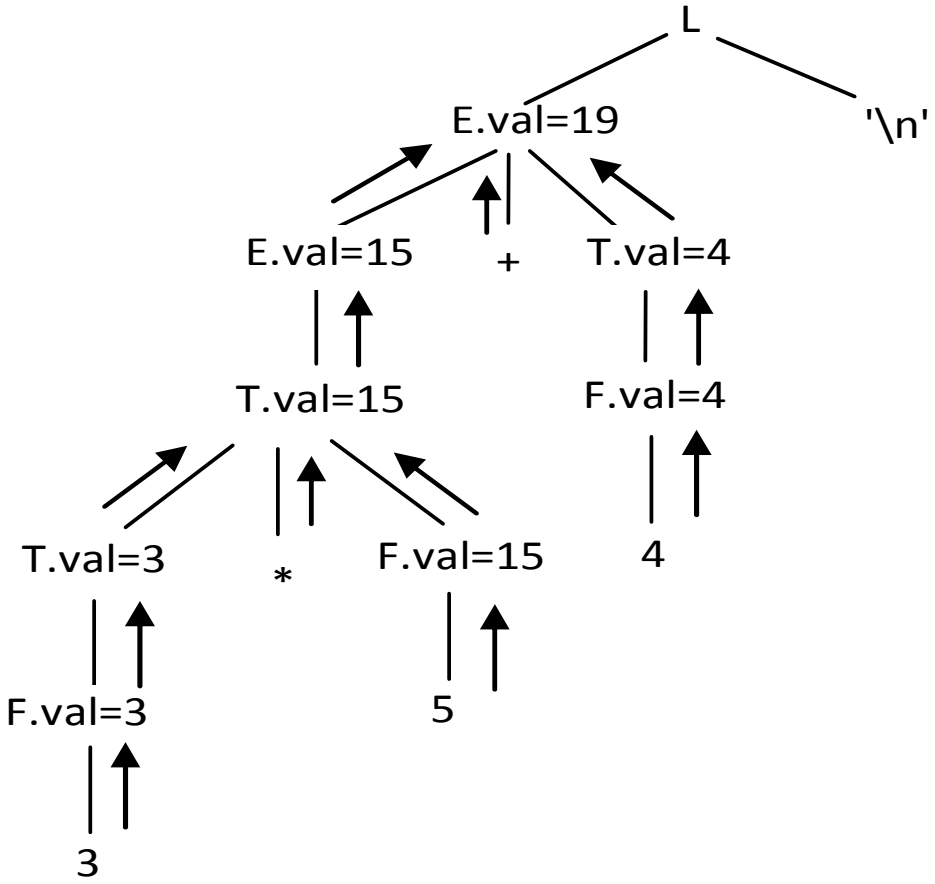
Traduction utilisant des attributs synthétisés

- Evaluation d'une expression arithmétique : **X.val** l'attribut associé à un symbole **X** de la grammaire.

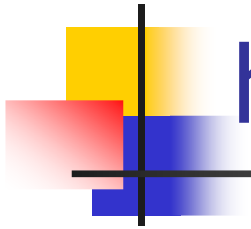
Production	Routine sémantique
L → E '\n'	Imprimer (E.val)
E → E + T	E.val := E.val + T.val
E → T	E.val := T.val
T → T * F	T.val := T.val * F.val
T → F	T.val := F.val
F → (E)	F.val := E.val
F → chiffre	F.val := Chiffre.val

Traduction utilisant des attributs synthétisés

- Arbre décoré pour l'expression suivante : 3 * 5 + 4



Traduction utilisant des attributs hérités



- Dans ce type d'analyse, la valeur d'un symbole de la grammaire est calculée à partir des ascendants ou "frères" dans l'arbre syntaxique.
- Les attributs hérités sont utilisés par exemple dans la déclaration des variables dans plusieurs langages de programmation (comme le langage C par exemple)

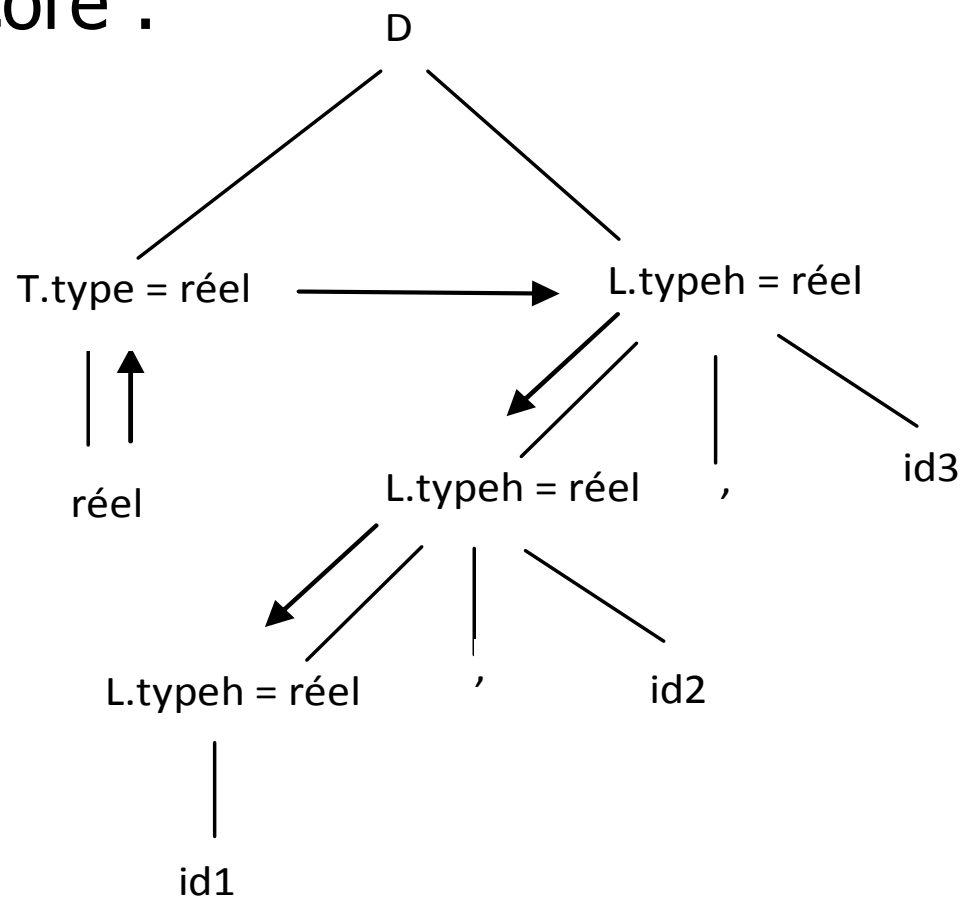
Traduction utilisant des attributs hérités

- Exemple :

Production	Routine sémantique
D $\rightarrow$ T L	L.typeh $\leftarrow$ T.type
T $\rightarrow$ 'entier'	T.type $\leftarrow$ entier
T $\rightarrow$ 'réel'	T.type $\leftarrow$ réel
L $\rightarrow$ L_1 , id	L_1.typeh $\leftarrow$ L.typeh
L $\rightarrow$ id	Ajouter id à la table des symboles

Traduction utilisant des attributs hérités

- Arbre décoré :



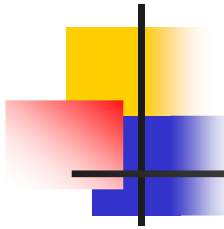
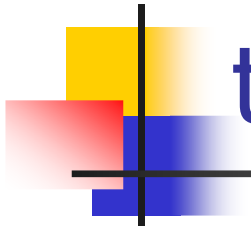


Schéma de traduction

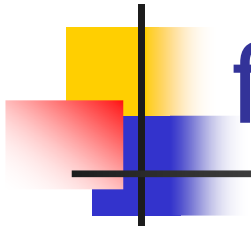
- Un schéma de traduction est une abstraction du traducteur dirigée par la syntaxe. Il est défini par :
  - la grammaire syntaxique (grammaire de type 2) dans laquelle des attributs sont associés aux symboles de la grammaire
  - et des actions sémantiques (code en C ou autre) sont insérées dans les MDP (Membres Droits de Productions).

Conception d'un schéma de traduction



- La conception d'un schéma de traduction est une étape cruciale. Elle est spécifique à chaque problématique. Tâche résumée par les étapes suivantes :
  - Trouver la "bonne grammaire"
  - Assurer que la valeur d'un attribut est disponible quand une action s'y réfère.
  - Insérer les routines sémantiques aux bons attributs (en fonction du problème à résoudre).

Méthodologie de génération de la forme intermédiaire



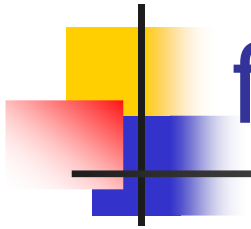
- Pour concevoir un schéma de traduction générant une forme intermédiaire le concepteur devra d'abord définir la forme à obtenir.
- Par exemple la traduction d'une instruction if-then-else en code à 3 adresses peut donner lieu à deux formes intermédiaires.

Méthodologie de génération de la forme intermédiaire

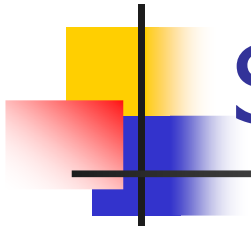
- Pour concevoir un schéma de traduction générant une forme intermédiaire le concepteur devra d'abord définir la forme à obtenir.
- Par exemple la traduction d'une instruction if-then-else en code à 3 adresses peut donner lieu à deux formes intermédiaires.

Source	Forme intermédiaire 1	Forme intermédiaire 2
if (a>b) then a++ else b--	**1.** CMP a,b **2.** JumpLessEqual **5** **3.** a := a+1 **4.** Jump **6** **5.** b :=b-1 **6.**	**1.** CMP a,b **2.** JumpGreater **5** **3.** b := b-1 **4.** Jump **6** **5.** a :=a+1 **6.**

Méthodologie de génération de la forme intermédiaire

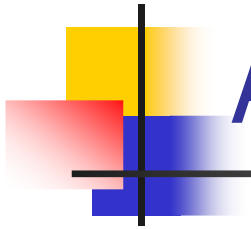


- Le concepteur devra choisir exclusivement une forme à produire.
- Dans le cas général, on choisira naturellement la forme qui produit le moins d'instructions et dont la traduction est adaptée au sens de lecture du programme source.



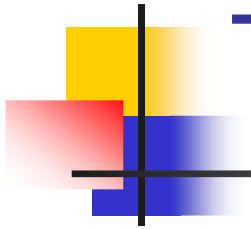
Structures de données utilisées

- Pour la traduction qui génère du code à trois adresses, nous utiliserons dans les routines sémantiques :
- une structure de données appelée ***QUAD*** qui contiendra les instructions quadruplets
- une procédure ***Générer-Quadruplet (code-op, source1, source2, destination)*** qui insère dans la structure ***QUAD*** le 4-tuple donné en argument
- un pointeur dénommé ***Quadcourant*** qui pointe sur la structure ***QUAD*** à la position courante.



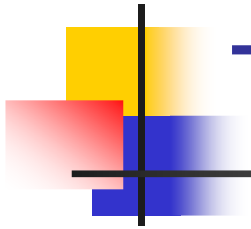
Traducteur Descendant ou Ascendant

- Pour chaque type d'analyse syntaxique (descendante ou ascendante) le traducteur aura ses spécifités.



Traduction descendante

- Certaines grammaires sont "naturellement" récursives gauches (comme par exemple la grammaire des expressions arithmétiques) ou non factorisées (grammaire de la structure if-then-else).
- Le positionnement des routines sémantiques pour ces grammaires est relativement simple.
- Par contre, si on prend une grammaire non récursive gauche ou factorisée pour ces cas, le positionnement des routines sémantiques n'est pas évident



Traduction descendante

- Le processus de traduction consiste à donner la grammaire "naturelle", positionner les routines sémantiques et enlever ensuite la récursivité gauche et procéder à la factorisation pour pouvoir faire une traduction descendante.
- Les routines sémantiques gardent leurs positions acquises dans la grammaire naturelle.
- **Remarque :**
  - Le type de certains attributs peut changer

Traduction descendante

- Schéma de traduction pour l'évaluation des expressions arithmétiques avec récursivité gauche de la grammaire :

$E \rightarrow E_D + T$	{E.val := E_D.val + T.val}
$E \rightarrow E_D - T$	{E.val := E_D.val - T.val}
$E \rightarrow T$	{E.val := T.val}
$T \rightarrow (E)$	{T.val := E.val}
$T \rightarrow nb$	{T.val := nb.val}

Traduction descendante

- Schéma de traduction descendant :

E →	T	{R.h := T.val}
	R	{E.val := R.s}
R →	+	
	T	{R_D.h := R.h + T.val}
	R_D	{ R.s := R_1.s}
R →	-	
	T	{R_D.h := R.h - T.val}
	R_D	{ R.s := R_D.s}
R →	ε	{ R.s := R.h}
T →	(E)	{T.val := E.val}
T →	nb	{T.val := nb.val}

Traduction descendante
Cas général

- Schéma de traduction avec récursivité à gauche :

$A \rightarrow A_D \ Y$	$\{A.s := g(A_D.s, Y.s)\}$
$A \rightarrow X$	$\{A.s := f(X.s)\}$

- Schéma de traduction descendant :

$A \rightarrow$	X	$\{R.h := f(X.s)\}$
	R	$\{A.s := R.s\}$
$R \rightarrow$	Y	$\{R_D.h := g(R.h,Y.s)\}$
	R_D	$\{R.s := R_D.s\}$
$R \rightarrow$	ε	$\{R.s := R.h\}$

Traduction descendante
Cas général

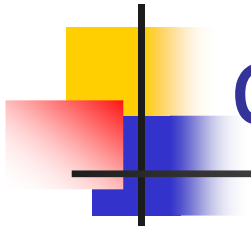
- Schéma de traduction sans factorisation:

$A \rightarrow X\ Y$	{A.s := g(X.s, Y.s)}
$A \rightarrow X\ Z$	{A.s := f(X.s, Z.s)}

- Schéma de traduction descendant :

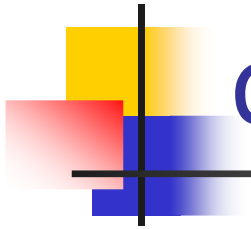
$A \rightarrow$	X	{R.h := X.s}
	R	{A.s := R.s}
$R \rightarrow$	Y	{R.s := g(R.h,Y.s)}
$R \rightarrow$	Z	{R.s := f(R.h,Z.s)}

Conception d'un traducteur descendant



- Pour chaque non terminal A, construire une fonction ayant un paramètre formel pour chaque attribut hérité et retournant toutes les valeurs des attributs synthétisés
- Ecrire le code de chaque fonction en suivant les principes donnés dans le chapitre consacré aux analyses syntaxiques descendantes ;
- Copier le code associé à toute action sémantique dans le corps de la fonction à une position équivalente à sa position dans le schéma de traduction.

Conception d'un traducteur descendant



- Donner le traducteur par descente récursive d’un évaluateur des expression arithmétiques.
- Donner le traducteur par descente récursive d’un générateur de code à 3 adresses des expression arithmétiques.
- Donner le traducteur descendant de l’instruction if-then-else générant des quadruplets.

Traduction descendante de l'instruction if

- Grammaire non factorisée générant l'instruction if :

  <Instr-if> → **if** <Cond> **then** <Instr>

  | **if** <Cond> **then** <Instr> **else** <Instr>

- Grammaire factorisée générant l'instruction if :

  <Instr-if> → **if** <Cond> **then** <Instr> <X>

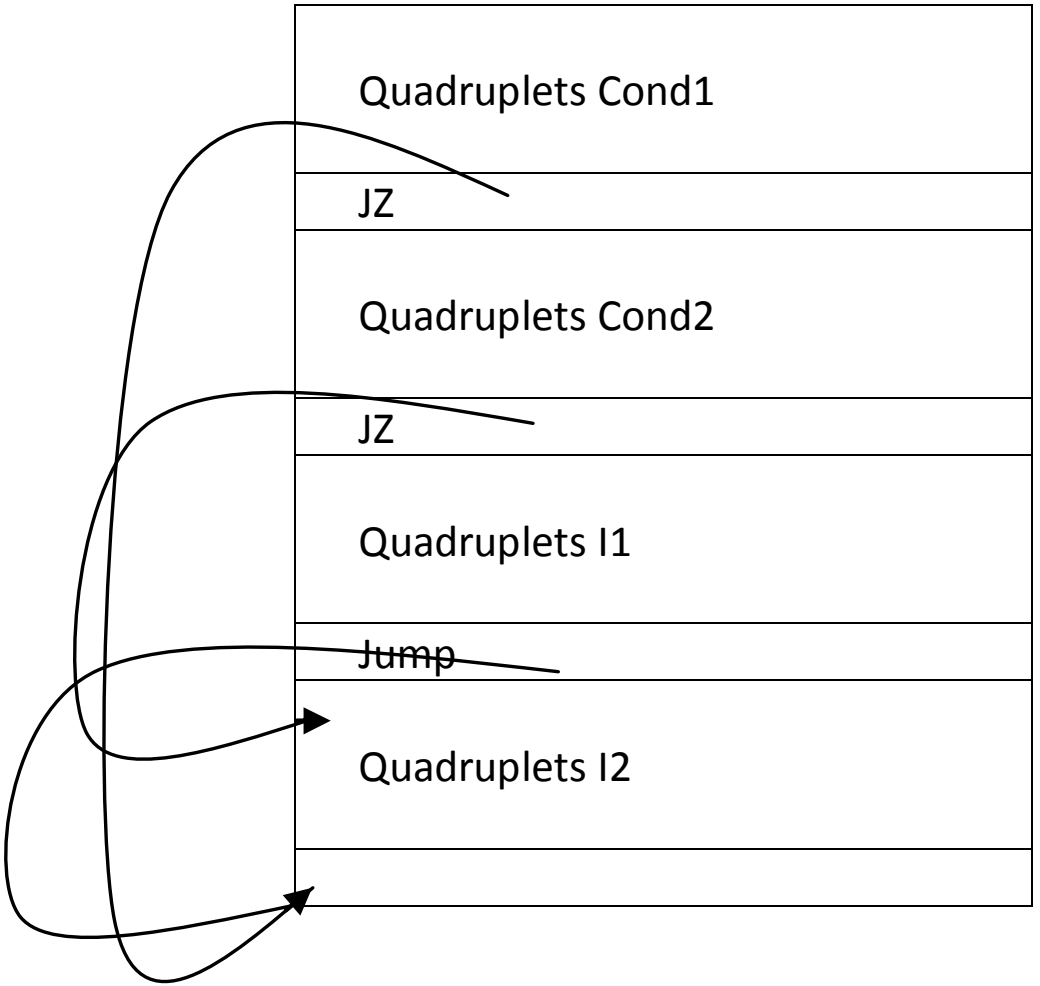
  <X> → **else** <Instr> | ε

Traduction descendante de l'instruction if

- Exemple d'instruction avec if imbriquées :

  if Cond1
  then if Cond2
  then I1
  else I2 ;

- Forme Intermédiaire de l'exemple : →

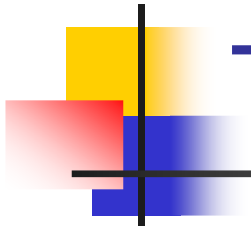


Traduction descendante de l'instruction if

```
Instr-if()
{ case tc of :
    "if" : {tc:=ts;
            Cond(); /* Quadruplets de condition */
            if (tc = "then")
            then   Quad(Quadcourant) := <JZ, , , >;
              Save-JZ:=Quadcourant;/*Save-JZ variable locale */
              Quadcourant ++;
              Instr(); /* traite l'instruction qui suit then */
              X(Save-JZ);
            else "Erreur syntaxique" ;
            end-if ;
    default : "Erreur syntaxique" ;
end-case ;
}
```

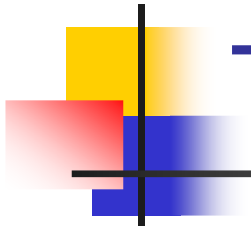
Traduction descendante de l'instruction if

```
X(Save-JZ)
{  case tc of :
"else" : {   tc := ts;
             Quad(Quadcourant) := <Jump, , , >;
             Save-Jump := Quadcourant;
             Quadcourant ++;
             Quad(Save-JZ).4 := Quadcourant;
             Instr(); /* l'instruction qui suit else */
             Quad(Save-Jump).4 := Quadcourant;
           }
"Suivant X" :       Quad(Save-JZ).4 := Qc;
default : "Erreur syntaxique" ;
end-case ;
```



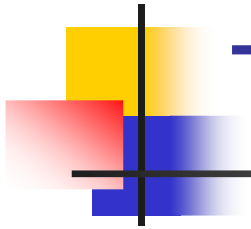
Traduction Ascendante

- **<u>Quand exécuter une routine sémantique lors d'une analyse ascendante ?</u>**
- Dans un traducteur ascendant, les routines sémantiques sont exécutées lors des réductions c'est à dire juste avant que le MDP (Membre Droit de Production) ne soit réduit.
- La réduction étant un point de repère de l'analyse, les attributs des symboles du MDP en cours de réduction sont disponibles sur la pile d'analyse.



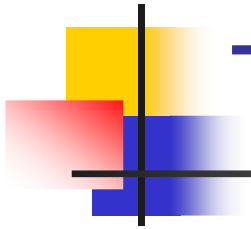
Traduction Ascendante

- **Elimination des actions intérieures**
- Puisque les routines sémantiques sont exécutées lors des réductions, il faudra éliminer toutes les actions intérieures d'un schéma de traduction pour pouvoir une traduction dirigée par la syntaxe ascendante.



Traduction Ascendante

- **Elimination des actions intérieures : cas LR**
- **<u>Procédé d'élimination des actions intérieures.</u>**
- Utiliser des non terminaux M_i dits marqueurs qui engendrent ε ;
- Remplacer chaque action intérieure par un marqueur distinct ;
- "Rattacher" l'action à la fin de la production $M_i \rightarrow \varepsilon$ où M_i est un marqueur.



Traduction Ascendante

- **Elimination des actions intérieures : cas PS**
- **Procédé d'élimination des actions intérieures.**
- Dans le cas d'analyse syntaxique ascendante de précédence, le procédé par "découpage de MDP" est adopté. Il aura pour but d'amener les routines sémantiques en fin de MDP par l'ajout de nouveaux non-terminaux générant des sous-MDP.

Schéma de traduction ascendant instruction if

- Exemple d'instruction if :

```
if Cond1
then        if Cond2
            then    Instr1
            else    Instr2
else        if Cond3
            then    Instr3
            else    Instr4
```

Schéma de traduction ascendant instruction if

- Forme intermédiaire de l' exemple précédent :

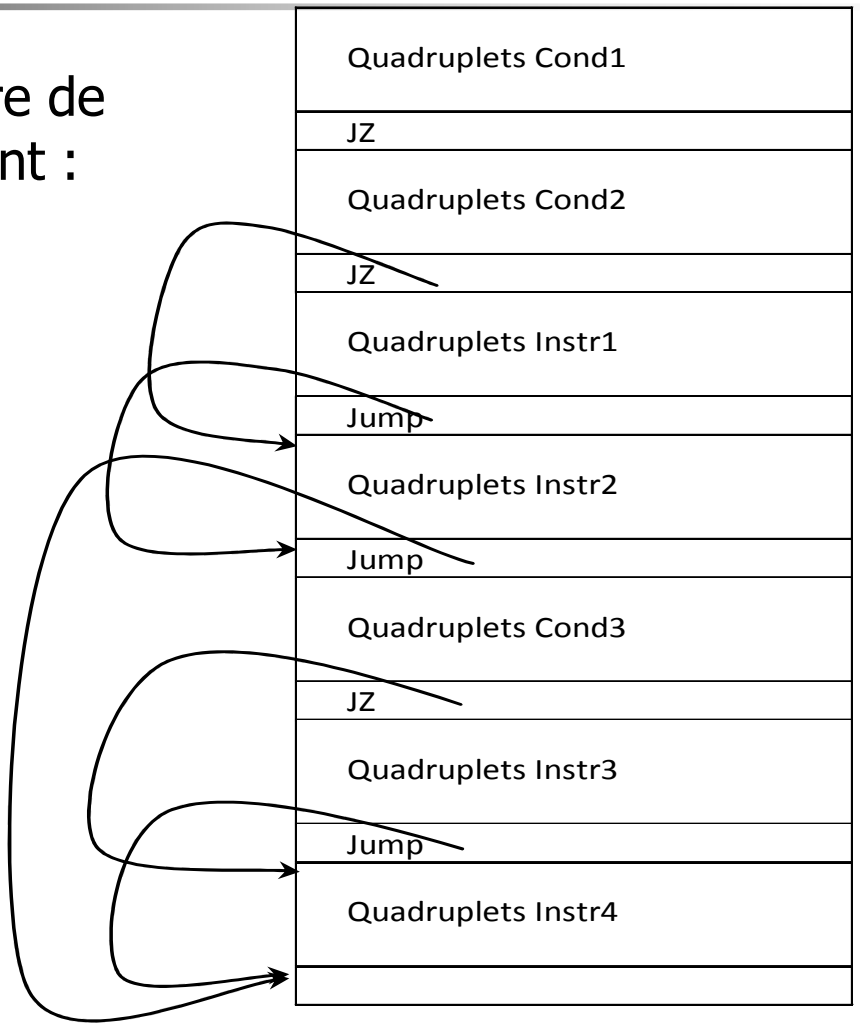


Schéma de traduction ascendant instruction if

- Grammaire qui permet de générer les instructions if :

<inst-if> $\rightarrow$ if <cond> then <instr> else <instr>

| if <cond> then <instr>

Schéma de traduction ascendant instruction if

<inst-if> →	if <cond> M1 then <instr> M2 else <instr> M3		
	| if <cond> M1 then <instr > M4		
M1 → ε	{	M1.val:=Quadcourant ;	
		QUAD(Quadcourant ++):=(JZ, , ,)	}
M2 → ε	{	M2.val:=Quadcourant;	
		QUAD(QUADCOURANT++):=(JMP, , ,)	
		QUAD(M1.val).4 := Quadcourant;	}
M3 → ε	{	QUAD(M2.val).4 := Quadcourant;	}
M4 → ε	{	QUAD(M1.val).4 := Quadcourant;	}