Traduction des Langages
TD

Charles Fallourd

TD 5 : Quadruplet

$$
\begin{aligned}
&< statlist > := < inst >< suite > \\
&\quad < suite > := ; < statlist > | \lambda \\
&\quad\quad < inst > := ... | < ifstat > | < whilestat > | < repeatstat > | < forstat > | < casestat > | ... \\
&< repeatstat > := repeat < statlist > until < exp > endrepeat \\
&\quad < forstat > := forident := < exp1 > step < exp2 > until < exp3 > do < statlist > endfor \\
&\quad < casestat > := case < exp > of < alternative > ; < alternative > * \\
&\qquad\qquad otherwise < statlist > endcase
\end{aligned}
$$

Methodologie pour les trois questions

1. Donner le schéma équivalent en quadruplets.
2. Inventaire des "problèmes" : verification types et gestion des adresses des quadruplets. Ainsi que l'inventaire des solutions apportées.
3. Mise en oeuvre des procédures de descente récursive pour engendrer le schéma en quadruplets voulus en tenant compte des solutions apportées.

Exercice 1

Mettre en oeuvre un processus de traduction en quadruplets des structures de controle pour <repeatstat>.

Schéma équivalent en quadruplet

$\uparrow$
$\vdots$ *quadruplet engendre par* $< statlist >$
$\downarrow$
$\uparrow$
$\vdots$ *quadruplet engendre par* $< exp >$

$\downarrow$
$=?, < exp > .res, 0, DEBUT$ ici la dernière ligne est un quadruplet, (=?, exp.res, 0, DEBUT)

Inventaire des problèmes et solution apportées

1. Verif Type : verifier que le resultlist de EXP(<exp>.res) est une variable déclarée Booléenne. $\Rightarrow$ CHECKTYPE
2. Référence en arrière à DEBUT $\Rightarrow$ variable DEBUT qui contiendra l'@ du 1er quadruplet engendré par STATLIST qui est dans NEXTQUAD

Mise en oeuvre

```
procedure REPEATSTAT is
  Res: String; -- nom de la variable qui contiendra le resultat du test
  Debut: Integer; -- garder l'@ du 1er quadruplet engendre par STATLIST

  begin
    SKIP('repeat');
    Debut := NEXTQUAD;
    STATLIST ;
    SKIP('until');
    EXP(Res);
    CHECKTYPE(Res, Boolean);
    GEN("=?," ^Res^," 0,"^str(Debut));
    SKIP('endrepeat');
  end REPEATSTAT;
```

Exercice 2

De meme pour <forstat>

Schéma équivalent en quadruplet

$\uparrow$
$\vdots$ *quadruplet engendre par* $< exp2 >$
$\downarrow$
$:=, < exp1 > .res, nil, ident \uparrow$
$\vdots$ *quadruplet engendre par* $< exp2 >$
$\downarrow$
$\uparrow$
$\vdots$ *quadruplet engendre par* $< exp3 >$
$\downarrow$
$TEST >?, ident, < exp3 > .res, FIN$
$\uparrow$
$\vdots$ *quadruplet engendre par* $< statlist >$
$\downarrow$
$+, ident, < exp2 > .res, ident$
$goto, nil, nil\ TESTFIN$

ici TEST (> ?), (ident), (<exp3>.res), (FIN), TEST est un label, et c'est un quadruplet jump conditionnel. Si (> ?) alors jump FIN.

Inventaire des problèmes et solution apportées

1. verif type : les 3 résultats res_i des 3 $<exp_i>$ doivent être des variables déclarées *Integer* (prob 1)
2. Référence en avant à FIN :

   engendrer un quadruplet incomplet (4^e champ à nil) (prob 2.)

   conserver l'@ de ce quadruplet incomplet dans une variable TEST (2.1)

   mettre à jour le 4^e champs (@) du quadruplet incomplet quand on connait l'@ FIN ⇒BACKPATCH (2.2)
3. Référence en arrière à TEST ⇒ conserver l'@ du quadruplet TEST (2.3)

Mise en oeuvre

```
procedure FORSTAT is
  Res1, Res2, Res3 : String -- resultats des 3 <exp_i>
  Test : Integer;
  VarBoucle : String;
  begin
    SKIP('for');
        if NEXTS != 'ident' then ERREUR; endif; -- 'ident' = unite lexical 'ident' (genre que
            c'est une variable)
    varBoucle := NEXTS.Valeur ; -- ici on prend la valeur de l'unite lexical ident
    SCAN;
    SKIP(':=');
    EXP(Res1);
    CHECKTYPE(Res1, Integer); -- (prob 1)
    GEN(":=,"^Res1^",nil,"^varBoucle);
    SKIP('step');
    EXP(Res2);
    CHECKTYPE(Res2, Integer); -- (prob 1)
    SKIP('until');
    EXP(Res3);
    CHECKTYPE(Res3, Integer); -- (prob 1)
    Test := NEXTQUAD; -- (prob 2.2)
    GEN(">?,"^VarBoucle^","^Res3^",nil"); -- (prob 2.1)
    SKIP('do');
    STATLIST;
    GEN("+,"^VarBoucle^","^Res2^","^VarBoucle);
    GEN("goto,nil,nil"^str(teST));
    BACKPATCH([Test], NEXTQUAD); -- (prob 3)
    SKIP('endfor');
  end FORSTAT ;
```

⋀ est l'opérateur de concaténation !

Exercice 3

De meme pour <casestat>

Schéma équivalent en quadruplet

$\uparrow$
$\vdots$ *quadruplet engendrer par* $< exp >$
$\downarrow$
$\neq ?\ ,\ < exp > .res,\ nb1,\ ALT2$
$\uparrow$
$\vdots$ *quadruplet engendrer par* $< alternative > pourS1$
$\downarrow$
$goto, nil, nill,\ FIN$
$ALT2\ \neq ?\ ,\ < exp > .res,\ nb1,\ ALT3$
$\uparrow$
$\vdots$ *quadruplet engendrer par* $< alternative > pourS2$
$\downarrow$
$goto, nil, nill,\ FIN$
$\vdots\ ALT3$
$ALT_p\ \neq ? < exp > .res,\ nb_p,\ OTHER$
$\vdots$ *quadruplet engendrer par* $< alternative > pourSp$
$goto, nil, nill,\ FIN$
$OTHER$
$\uparrow$
$\vdots$ *quadruplet pour* S_{p+1}
$\downarrow$
FIN

Inventaire des problèmes et solution apportées

1. Verif type : Integer de <exp>.res $\Rightarrow$ CHECKTYPE α
2. Reference en avant à ALT_{i+1}
   mémoriser dans une variable ALT (β_1) l'adresse du quadruplet engendré incomplet et maj du champs @ de ce quadruplet quand on connait. β_2 $Rightarrow \beta_3$ $\Rightarrow$ BACKPATCH
3. Réference en avant à FIN
   mémoriser les @ des quadruplet engendrés (γ_1) incompletdans une liste (γ_2)
   et on met à jours les champs @ de tous les audruplet de la liste avec la même @ FIN quand on la connait
   $\Rightarrow$ BACKPATCH γ_3
4. Verification "sémantique"
   $\Rightarrow$ chaque nb_i doit être différent. On doit donc avoir une liste de nombre. on suppose qu'on a un outil MEMBER(E,L) $= VRAI\ si\ e\ \epsilon\ L$

Mise en oeuvre

```
procedure CASESTAT is
    Res: String ;-- resultatde EXP
    ListFin: list of Integer;
    ListNbre: list of Integer;
    begin
        SKIP('case');
        EXP(Res);
        CHECKTYPE(Res,Integer); -- α
        SKIP('of');
        ListFin: MAKELIST;
        ListNbre: MAKELIST; -- on creer les listes vide
        ALTERNATIVE(Res,ListNbre);
        -- NEXTQUAD -1 contient l'@ du
        -- goto, nil,nil,nil
        ListFin := MERGE(ListFin, [NEXTQUAD-1]); -- γ1
        While NEXTS = ';' loop
            SKIP(';'); -- ou SCAN puisqu'on sait déjà que c'est un ';'
            ALTERNATIVE(Res,ListNbre);
            ListFin := MERGE(ListFin, [NEXTQUAD-1]);
        endloop;
        SKIP('otherwise');
        STATLIST;
        BACKPATCH(ListFin,NEXTQUAD); -- γ3
        SKIP('endcase');
    end CASESTAT;
```

```
procedure ALTERNATIVE(Res in String, ListNbre in out) is
    ALT: Integer;
    begin
        if NEXTS ≠ 'number' then ERREUR; endif;
        if MEMBER(NEXTS.Valeur, ListNbre) then ERREUR; endif; -- α
        ListNbre := MERGE([NEXTS.Valeur], ListNbre);
        ALT := NEXTQUAD; -- β1
        GEN("≠ ?,"^Res^","^str(NEXTS.Valeur)^",nil"); -- β2
        SCAN;
        SKIP(':');
        STATLIST;
        GEN("goto,nil,nil,nil"); -- γ2
        BACKPATCH([ALT], NEXTQUAD); -- β3
    end ALTERNATIVE;
```

TD 6
Génération de code & analyse ascendante

Todo : * action sémantique = ?

Méthode de la descente récursive

$< statlist > :=< inst >< suite >$

$< suite > :=; < statlist > |\lambda$

$< inst > := ...| < ifstat > | < whilestat > | < repeatstat > | < forstat > | < casestat > |...$

$< loopstat > := ident : loop < statlist > endloop$

$< exitstat > := when < exp > exitident$

```
1  A: loop
       ...
3      when E1 exit A;
       ...
5      B : loop
           ...
7          when E2 exit A;
           ...
9          when E3 exit B;
           ...
11         endloop;
       ...
13     when E4 exit A;
       ...
15     endloop
```

Schéma équivalent en quadruplet de l'exemple

$DbutA \uparrow$

$\vdots ...(quelquechose...)$

$\downarrow$

$\uparrow$

$\vdots$ *quadruplet engendr pour* E_1

$\downarrow$

$\neq ?\ ,\ E1.res,\ 1,\ finAIl\ s'agit\ de\ < exitstat >$

$\uparrow$

$\vdots ...(quelquechose...)$

$\downarrow$

$Dbut\ B$

$\uparrow$

$\vdots ...(quelquechose...)$

$\downarrow$

$\uparrow$

$\vdots$ *quadruplet pour* E_2

$\downarrow$

$\uparrow$

$\downarrow$

$\neq ?\ ,\ E2.res,\ 1,\ finAIl\ s'agit\ de\ < exitstat >$

$\uparrow$

$\vdots$...($quelquechose$...)

$\downarrow$

$\uparrow$

$\vdots$ $quadruplet\ pour\ E_3$

$\downarrow$

$\neq ?\ ,\ E3.res,\ 1,\ finBIl\ s'agit\ de\ <exitstat>$

$\uparrow$

$\vdots$...($quelquechose$...)

$\downarrow$

$goto, nil, nil,' DebutB'$

$finB$

$\uparrow$

$\vdots$...($quelquechose$...)

$\downarrow$

$\uparrow$

$\vdots$ $quadruplet\ pour\ E_4$

$\downarrow$

$\neq ?\ ,\ E4.res,\ 1,\ finAIl\ s'agit\ de\ <exitstat>$

$\uparrow$

$\vdots$...($quelquechose$...)

$\downarrow$

$goto, nil, nil, DebutA$

$finA$

Inventaire des problèmes et solution apportées

1. Vérifier le type booléen des E_i.res (des exitstat). $\Rightarrow$ CHEKTYPE.
2. Référence en arrière à Début $\Rightarrow$ mémoriser l'@ dans une variable.
3. Référence en avant à Fin $\Rightarrow$ engendrer un quad incomplet, mémoriser son @ dans liste de quad incomplets et on met à jour le champs @ avec BACKPATCH. $\Rightarrow$ Associer la liste de quad incomplets à un nom de boucle.

Attention!! On ne veut pas de boucles imbriquées de même nom.
On peut vouloir le même nom sur deux boucles "consécutives".
$\Rightarrow$ On va utiliser la tables des symboles TDS. Dans l'exemple suivant on pourrait rajouter d'autres informations.

lqi = liste des quadruplets incomplets.

	nomBoucle	lqi	type		
1					
P	A	TDS[P].lqi	loop		
Q	B				

On va vérifier que le nom d'une boucle imbriqué n'est pas déjà dans la table
⇒ SEARCH(S :string, out p :int, T :type) . Si p = 0, n'existe pas dans TDS.
nom de boucle et même type boucle différent ! Si c'est un autre type, pas grave.
Pour ajouter à TDS ⇒ ENTER(S :String, out P :int)
CANCEL(S :String, T :type) ⇒ va supprimer l'entré de type T et de nom S.

```
procedure LOOPSTAT(in S:String)
    -- S est le nom de la boucle (voir ident ... apparemment la meme chose)
    P: Integer;
    DEBUT: Integer;
    begin
        SKIP('loop');
        SEARCH(S,P, 'loop');
        si P!=0 then ERREUR;
        -- on a vérifié que le nom de boucle n'est pas
        -- déjà utilisé comme boucle englobante

        -- on met le nom S dans TDS
        ENTER(S,P);
        TDS[P].types := 'loop';
        TDS.[P].lqi := MAKELIST;

        -- on mémorise l'@ du premier quad DEBUT
        DEBUT := NEXTQUAD;
        STATLIST;
        SKIP('endloop');
        GEN("goto,nill,nill"^str(DEBUT));
        BACKPATCH(TDS[P].lqi,NEXTQUAD);

        -- effacer le nom de la boucle de la TPS
        CANCEL(S,'loop');
    end LOOPSTAT
```

```
procedure EXITSTAT is
    Res: String;
    P: Integer;
    begin
    SKIP('when');
    EXP(RES);
    CHECKTYPE(Res, Boolean);
    SKIP('exit');
    if NEXTS != 'identicateur' then ERREUR;
    SEARCH(NEXTS.String, P, 'loop');
    if P = 0 then ERREUR;
    TDS[P].lqi := MERGE(TDS[P].lqi,[NEXTQUAD]);
    GEN("=?,"^Res^",s,nill");
    SCAN;
    end EXITSTAT
```

Traduction par analyse ascendante

$$< forstat > := ident :=< exp1 > step < exp2 > until < exp3 > do < statlist > endfor \quad (1)$$

$$< loopstat > := ident : loop < statlist > endloop \quad (2)$$

Question 1

$\Rightarrow$ analyseur LR(k). le non terminal le plus à Droite.
k=1 dans la réalité.
Ces analyseur trouvent une grammaire et la réduit.
On part du mot et on remonte vers l'axiome.
A se dérive en B et B se réduit en A. (ouai d'un coup c'est plus clair ><)

Pour se faire :
même schéma en quadruplet,
même problemes et solutions.
MAIS la mise en oeuvre est différente. Ce n'est plus des procédures de descente récursive.