Translation of Definite Iteration: Pascal For / 1

How does it to do it ?

$$\frac{S \vdash ei \rightarrow (S1, vi) \quad S1 \vdash ef \rightarrow (S2, vf) \quad S2 \vdash e \rightarrow (S3, v) \quad C \equiv a\ c1 \ldots ck \quad k = \lfloor (vf - vi)/v \rfloor + 1 \quad a \equiv i := vi \quad (1 \le j \le k,\ aj \equiv i := vi + j * v \quad cj \equiv c\ aj) \quad S3 \vdash C \rightarrow S4}{S \vdash \text{for } i := ei \text{ to } ef \text{ step } e \text{ do } c \rightarrow S4}$$

(Semantics of Pascal For)

Extending the Simple Grammar

C::= R

R::= For ide:=E To E Step E do C;

Is the translation correct, when the block C can update index ide?

The extended part of the Attribute Grammar

C::= R {C.next:=R.next}

R::= For ide:=E_1 {emit(ide.loc":="E_1.loc)}
To E_2
Step E_3 do {loop-init:=quad;
R.next:=quad;
emit("if" ide.loc [>] E_3.loc "goto --")}
C; {BK(C.nex,loop-init);
emit(ide.loc":="ide.loc [+] E.loc);
emit("goto" loop-init)}

Translation of Definite Iteration: Pascal For / 2

How does it to do it ?

$$\frac{\begin{array}{l} S \vdash e_i \rightarrow (S_1, v_i) \quad S_1 \vdash e_f \rightarrow (S_2, v_f) \quad S_2 \vdash e \rightarrow (S_3, v) \quad \text{(Semantics of Pascal For)} \\ C \equiv a\ c_1 \ldots c_k \quad k = \lfloor (v_f - v_i)/v \rfloor + 1 \\ a \equiv i := v_i \quad (1 \le j \le k,\ a_j \equiv i := v_i + j * v \quad c_j \equiv c\ a_j) \\ S_3 \vdash C \rightarrow S_4 \end{array}}{S \vdash \text{for } i := e_i \text{ to } e_f \text{ step } e \text{ do } c \rightarrow S_4}$$

Extending the Simple Grammar

C::= R

R::= For ide:=E To E Step E do C;

A different translation that guarantees the semanhtics also when the for block C updates the control variable ide.

The extended part of the Attribute Grammar

C::= R {C.next:=R.next}

R::= For ide:=E_1 {emit(ide.loc":="E_1.loc);
j:=newtemp; emit(j":=#1");}
To E_2
Step E_3 do {loop-init:=quad;
R.next:=quad;
emit("if" ide.loc [>] E_2.loc "goto --")}
C; {BK(C.nex,loop-init);
temp:=newtemp;
emit(temp":=" j [*] E_3.loc;
emit(ide.loc":="E_1.loc [+] temp);
emit(j":="j[+]"#1");
emit("goto" loop-init)}

Translation of Definite Iteration: Pascal For / 3

Show the code generated, by using the last attribute grammar, given for For, when the source is the following fragment of a Simple program:

```
For i:=0 to 10 step 4
  do {i:=i-4;j:=j+i}
```

Give firstly, semantics and code generation of the command sequence: {c1;…;cn} for arbitrary non-Empty sequences of comannds.

Translation of Conditional If

How does it to do it ?

(Semantics of conditional If)

$$\frac{S \vdash e \rightarrow (Se,true) \qquad Se \vdash c1 \rightarrow S1}{S \vdash \text{if } e\ c1\ c2 \rightarrow S1} \qquad \frac{S \vdash e \rightarrow (Se,false) \qquad Se \vdash c2 \rightarrow S2}{S \vdash \text{if } e\ c1\ c2 \rightarrow S2}$$

Extending the Simple Grammar

C::= ITE

ITE::= if E then C else C

The extended part of the Attribute Grammar using value-based translation for boolean

C::= ITE {C.next:=ITE.next}

ITE::= if E {else-init:=[quad];
emit(if E.loc":=#false goto --")}
then C_1 {next:=[quad];
emit("goto --");
BK(else-init,quad);}
else C_2 {ITE.next:=C_1.next+next+C_2.next;}

The extended part of the Attribute Grammar using short-circuit translation for boolean

C::= ITE {C.next:=ITE.next}

ITE::= if E {BK(E.true,quad);}
then C_1 {next:=[quad];
emit("goto --")
BK(E.false,quad)}
else C_2 {ITE.next:=C_1.next+next+C_2.next;}

Translation of Conditionals If and If-then

Solving a classical case of ambiguity:

if e then if e1 then c1 else c2

is it meaning:

- if e then (if e1 then c1) else c2 ?

- if e then (if e1 then c1 else c2) ?

Noting the use of concrete syntax (in the high part) and abstract syntax (in the low part of the sentence)

Extending the Simple Grammar

C::= U | S
U::= if E then C | if E then S else U
S::= A | W | F | R | {C} | if E then S else S

Extending the Simple Grammar

C::= Z | IT | ITE
Z::= A | W | F | R | {C}
ITE::= if E then Z else C
IT::= if E then C

else is always, associated with the nearest preceding if

Translation of Conditionals If and If-then / 2

Extending the Simple Grammar

C::= U | S
U::= if E then C | if E then S else U
S::= A | W | F | R | {C} | if E then S else S

Extending the Simple Grammar

C::= Z | IT | ITE
Z::= A | W | F | R | {C}
ITE::= if E then Z else C
IT::= if E then C

Extend each grammar with the attributes for code generation

Translation of Conditional Case: C Switch

How does it to do it ?

(Semantics of C Switch)

$$\frac{S \vdash e \rightarrow (S1, v) \qquad S1 \vdash swV\ (v)\ q \rightarrow S2}{S \vdash switch\ (e)\ q \rightarrow S2}$$

$$\frac{v \in \{w1 \ldots wk\} \qquad S \vdash c \rightarrow S1}{S \vdash swV\ (v)\ case\ w1 \ldots wk: c\ q \rightarrow S1}$$

$$\frac{v \notin \{w1 \ldots wk\} \quad S \vdash swV\ (v)\ q \rightarrow S1}{S \vdash swV\ (v)\ case\ w1 \ldots wk: c\ q \rightarrow S1}$$

$$\frac{S \vdash c \rightarrow S1}{S \vdash swV\ (v)\ default: c \rightarrow S1}$$

Extend the grammar of Simple with productions for command *switch*

Translation of Conditional Case: C Switch

How does it to do it ?

(Semantics of C Switch)

$$\frac{S \vdash e \rightarrow (S1, v) \qquad S1 \vdash swV\ (v)\ q \rightarrow S2}{S \vdash switch\ (e)\ q \rightarrow S2}$$

$$\frac{v \in \{w1 \ldots wk\} \qquad S \vdash c \rightarrow S1}{S \vdash swV\ (v)\ case\ w1 \ldots wk: c\ q \rightarrow S1}$$

$$\frac{v \notin \{w1 \ldots wk\} \quad S \vdash swV\ (v)\ q \rightarrow S1}{S \vdash swV\ (v)\ case\ w1 \ldots wk: c\ q \rightarrow S1}$$

$$\frac{S \vdash c \rightarrow S1}{S \vdash swV\ (v)\ default: c \rightarrow S1}$$

Extending the Simple Grammar

C::= S
S::= switch E Q
Q::= case V L : C Q
Q::= default : C
L::= V L
L::= ε
V::= Num | Char

Extend the grammar above with the actions for for the code generation

Translation of Conditional Case: C Switch

Attributes:

*in**loc***:- location returned from the translation of the switch guard expression
- inherited of {Q,L}
- used for generating code that compares the value of the guard expression with the labels of the statements

inInit:- list of the target incompletes that must be completed with the the address of the next label checking stm.
- inherithed of {L}

Cinit: - list of the target incompletes that must completed with the address starting the code of the case command
- synthesized of {L}

False: - list of the target incompletes that must completed with the address starting the code of the next case guard
- synthesized of {L}
- used to generate code that transfer the control to the next case statement.

Extending the Simple Grammar

C::= S
S::= switch E Q
Q::= case V L : C Q
Q::= default : C
L::= V L
L::= ε
V::= Num | Char

Extend the grammar above with the actions for for a code generation using the attributes introduced in the top of the slide

Conditional Case: C Switch Applying the Translation

Show the code generated, by using the given, attribute grammar, when the source is the following fragment of a Simple program:

```
switch x
  case u, v, z : x:= y*w+x;
  default: x:=0;
```

Conditional Case: Switch
Syntactic Sugar and Abstract Syntax

Noting that the this form of switch is adeguate for the Abstract Syntax of switch of C-like languages

Complete the grammar with the attributes that are needed to checking for the correct presence of *exactly one default* statement in the command switch

Complete with actions that are Checking for one default

C::= S
S::= switch E Q
Q::= U Q
Q::= U
U::= case V L : C
U::= default : C
L::= V L
L::= ε
V::= Num | Char

Conditional Case: Switch
Syntactic Sugar and Abstract Syntax

Noting that the this form of switch is adeguate for the Abstract Syntax of switch of C-like languages

Complete the grammar with the attributes that are needed to generate an Abstract Syntax having the switch default statement as the last sub-tree in the tree depth-first visit.

Extending the Simple Grammar

C::= S
S::= switch E Q
Q::= U Q
Q::= U
U::= case V L : C
U::= default : C
L::= V L
L::= ε
V::= Num | Char

where mk("a")=["a"-<>]; mk("a",L)=["a"-L]; operazioni su liste L di tree: emptylist=[];append=+

Conditional Case: Switch
Syntactic Sugar and Abstract Syntax

Noting that the this form of switch is adeguate for the Abstract Syntax of switch of C-like languages

Complete the grammar with the attributes that are needed to generate a text transliteration of the switch command in a way that default statement is put at the end, whilst all the other ones stay unchanged

Extending the Simple Grammar

C::= S
S::= switch E Q
Q::= U Q
Q::= U
U::= case V L : C ;
U::= default : C ;
L::= , V L
L::= ε
V::= Num | Char

where || = *text concatenation*

A Strange Iteration Statement: iter

(Reduction Semantics of iter)

$$\frac{S \vdash e \rightarrow (S,i) \quad q \equiv c_1 \ldots c_i \ldots c_n \quad S \vdash c_i \rightarrow S1 \quad S1 \vdash \text{iter } e\ q \rightarrow S2}{S \vdash \text{iter } e\ q \rightarrow S2}$$

$$\frac{S \vdash e \rightarrow (S,i) \quad q \equiv c_1 \ldots c_i \ldots c_n \quad i \notin [1..n]}{S \vdash \text{iter } e\ q \rightarrow S}$$

Extend the Simple Grammar with the command *iter*

C::= R
R::= iter E Q
Q::= C Q
Q::= C

Extend into an attribute grammar for code translation

C::= R {C.next:= R.next;}
R::= iter {init:=quad;}
E {Q.iloc:= E.loc; Q.icount:=1; Q.iinit:=init; Q.inextc:=[]}
Q {R.next:= Q.next;}
Q_1::= {BK(Q_1.nextc,quad); Q_2.inextc:=[quad];
emit("if" Q_1.iloc [≠] Q_1.icount "goto --");}
C {BK(C.next,Q_1.iinit); emit("goto" Q_1.iinit);
Q_2.iloc:= Q_1.ilocc; Q_2.icount:=Q_1.icoun+1; Q_2.iinit:=Q_1.init;}
Q_2 {Q_1.next:= Q_2.next;}
Q::= {BK(Q.nextc,quad); Q.next:=[quad];
emit("if" Q_1.iloc [≠] Q_1.icount "goto --");}
C {BK(C.next,Q.iinit); emit("goto" Q.iinit);}

A Strange Iteration Statement: iter

Show the code generated, by using the given, attribute grammar, when the source is the following fragment of a Simple program:

```
iter x
   x:= x+1;
   x:= x -1;
   while x>1 do x:=x-1;
```

Procedure Invocation: Transmission by Value

(Reduction Semantics of call-by-value)

$$\frac{S_0 \vdash e_1 \rightarrow (S_1, v_1) \ \ldots \ S_{n-1} \vdash e_n \rightarrow (S_n, v_n) \qquad S_n \vdash \text{binding}((v_1,\ldots,v_n),(x_1,\ldots,x_n)) \rightarrow S_{inv} \qquad \text{body}(S_n,p) = C \qquad \text{pars}(S_n,p) = x_1,\ldots,x_n \qquad S_{inv} \vdash C \rightarrow S_{ret}}{S_0 \vdash p(e_1,\ldots,e_n) \rightarrow S_{ret}}$$

Extend into an attribute grammar

C::= P
P::= ide (E V)
V::= , E V
V::=ε

Procedure Invocation: Transmission by Reference

(Reduction Semantics of call-by-reference)

$$\frac{S_0 \vdash e_1 \rightarrow (S_1,l_1) \ \ldots \ S_{n-1} \vdash e_n \rightarrow (S_n,l_n) \quad S_n \vdash \text{binding}((l_1,\ldots,l_n),(x_1,\ldots,x_n)) \rightarrow S_{inv} \qquad \text{body}(S_n,p) = C \qquad \text{pars}(S_n,p) = x_1,\ldots,x_n \qquad S_{inv} \vdash C \rightarrow S_{ret}}{S_0 \vdash p(e_1,\ldots,e_n) \rightarrow S_{ret}}$$

Extend into an attribute grammar

C::= P
P::= ide (E V)
V::= , E V
V::=ε

Preprocessing Optimization
Expression Constant Reduction

Complete the grammar of integer expressions of Simple with the attributes and the actions that are needed to generate a text transliteration of expression in a way that constant sub-expressions are statically computed and replaced with the resulting value. For instance, the expression "3+2*x*(2+1)+10+y" must be reduced to: "x*6+y+13".

Extend into an attribute grammar

E::= E + F | F
F::= F * N | N
N::= - N | T
T::= num | ide | (E)