Assigning Types to Program Structures of a Strongly Typed Language

Types

Types=Basic-Types + N +Types X Types ->Types

Rules

$$\frac{\Gamma\text{|- i:t}\quad \Gamma\text{|- e:T}}{\Gamma\text{|- i:=e:N}} \qquad \frac{\Gamma\text{|-e:boolean}\quad \Gamma\text{|-Cs: N}}{\Gamma\text{|- while e do Cs :N}} \qquad \frac{\Gamma\text{|-C:N}\quad \Gamma\text{|-Cs:N}}{\Gamma\text{|-C;Cs : N}}$$

$$\frac{}{\Gamma_1,i:T,\Gamma_2\text{|- i:T}} \qquad \frac{\Gamma\text{|-e1:T1}\quad \Gamma\text{|-e2: T2}\quad \Gamma\text{|-op:T1xT2->T}}{\Gamma\text{|- op(e1,e2):T}} \qquad \frac{\Gamma,i:Ti,j:Tj\text{ |-e:T}}{\Gamma\text{|-fun(i:Ti,j:Tj)\{e\} : TixTj->T}}$$

Attributes:

r:- type of all the program structures but expressions
-synthesized of **P,C,Cs,A,W**

type:- type of expression
-synthesized of **E,F, E',F', T, ide**

in: -type expression of the left brother on the left
-inherited of **E', F'**

It is extending the first grammar in its last variant discussed in variant

Try to do it yourself + then compare your solution with the one in the next slide.

A type checker for Simple Commands

[0] P::= Ds Cs	P.r:= Cs.r
[1] P::= Cs	P.r:= Cs.r
[2] Ds::= Var Dts	
[3] Dts::= Dt Dts'	
[4] Dts'::= ; Dt Dts'	
[5] Dts'::= ε	
[6] Dt::= ide O	addtype(ide.entry,O.t)
[7] O_1::= , ide O_2	addtype(ide.entry,O_2.t); O_1.t:= O_2.t
[8] O::= : Y	O.t:=Y.t
[9] Cs_1::= ; C Cs_2	Cs_1.r:= if (C.r=N & Cs_1.r=N) then N else ⊥
[10]Cs::= ε	Cs.r::= N
[11]C::= A	C.r::= A.r
[12]C::= W	C.r::= W.r
[13]A::= ide:=E	A.r::= if (ide.type=E.type & ide.type≠⊥) then N else ⊥
[14]W::= while E do C Cs endw	W.r::=if(E.type=boolean & C.r=Cs.r) then C.r else ⊥
[24]Y::= boolean	Y.t:= boolean
[25]Y::= integer	Y.t:= integer
[26]Y::= file	Y.t:= file

A type checker for Simple Expressions

[15]E::= F E'	{E'.in = F.type; E.type := E'.type;

[16]E'_1::= op-l F E'_2	{match t1xt2->t = op-l.type;
	if (E'_1.in = t1 & F.type = t2)
	then {E'_2.in = t; E'_1.type := E'_2.type;}
	else {E'_2.in = ⊥; E'_1.type :=⊥;};
	}

[17]E'::= ε	{E'.type = E'.in}
[18]F::= T F'	{…}
[19]F'::= op-h T F'	{…}
[20]F'::= ε	{…}
[21]T::= num	{T.type = integer;}
[22]T::= ide	{T.type = ide.type;}
[23]T::= (E)	{T.type = E.type;}