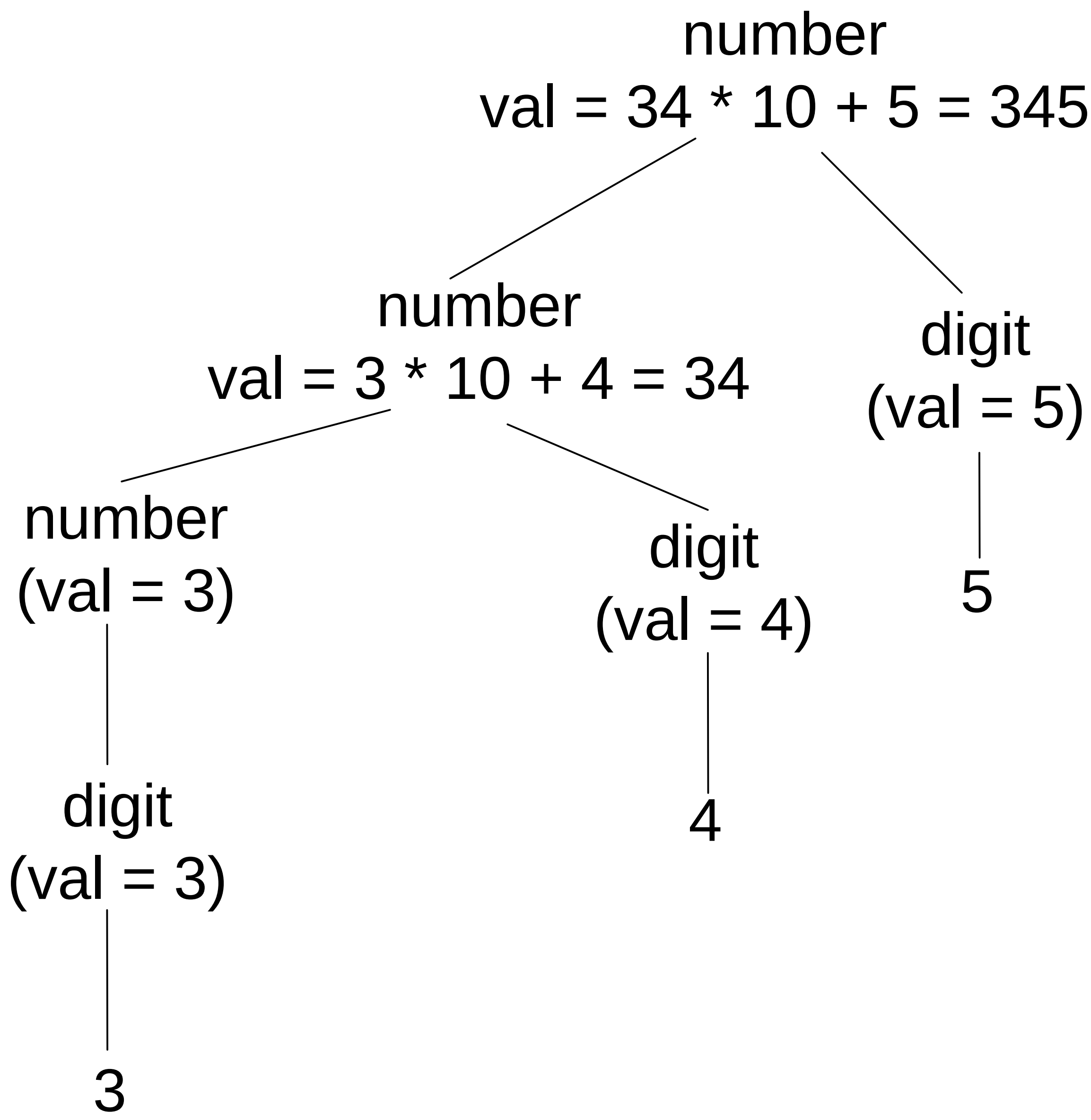
Semantic

<<Semantic.ppt>>

(* Attribute Grammar: synthetic/inherited attributes *)

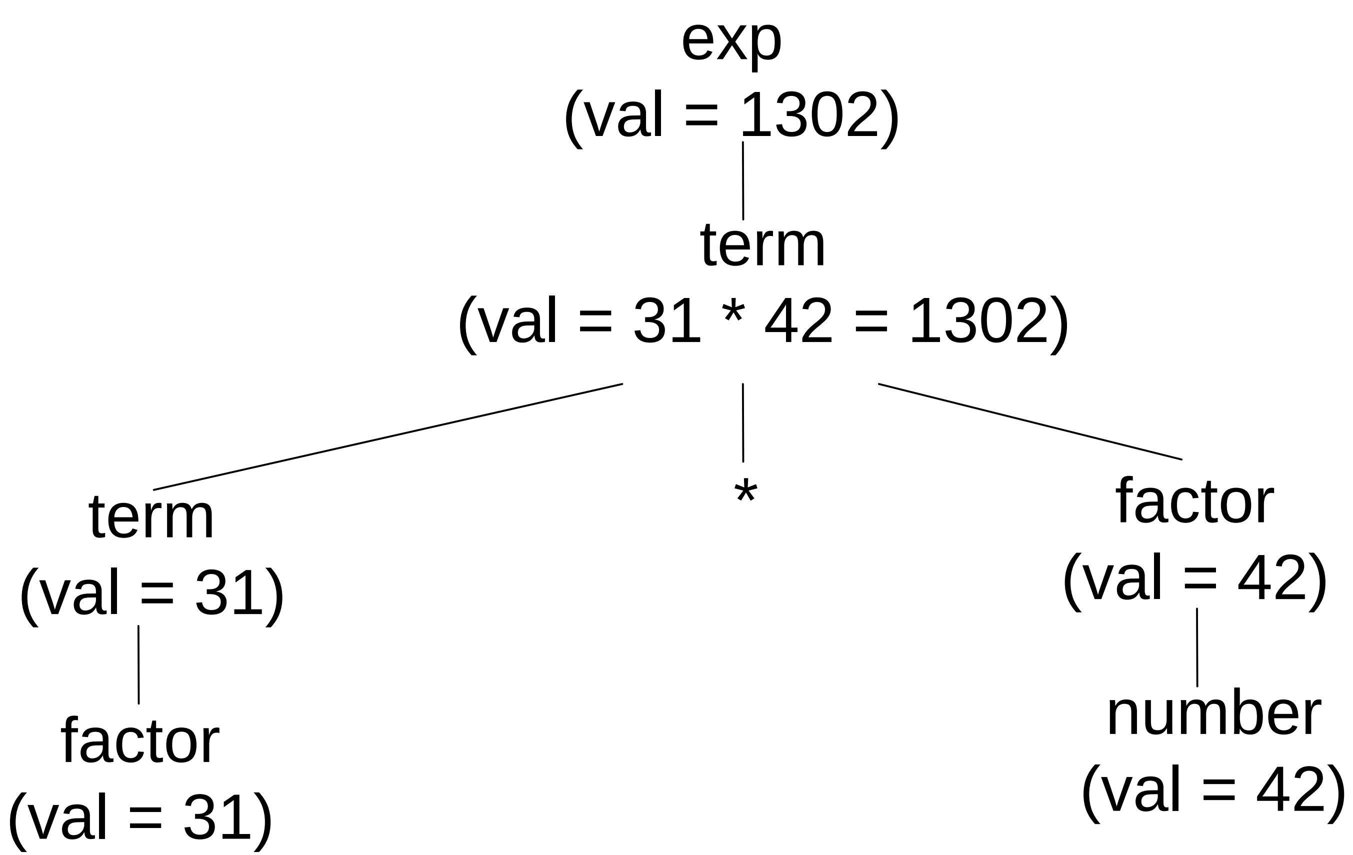
(* synthetic attribute *)

Grammar Rule	Semantic Rules
$number_1$ -> $number_2$ digit	$number_1$.val = $number_2$.val * 10 +digit.val
number -> digit	number.val = digit.val
digit -> 0	digit.val = 0
digit -> 1	digit.val = 1
digit -> 2	digit.val = 2
digit -> 3	digit.val = 3
digit -> 4	digit.val = 4
digit -> 5	digit.val = 5
digit -> 6	digit.val = 6
digit -> 7	digit.val = 7
digit -> 8	digit.val = 8
digit -> 9	digit.val = 9

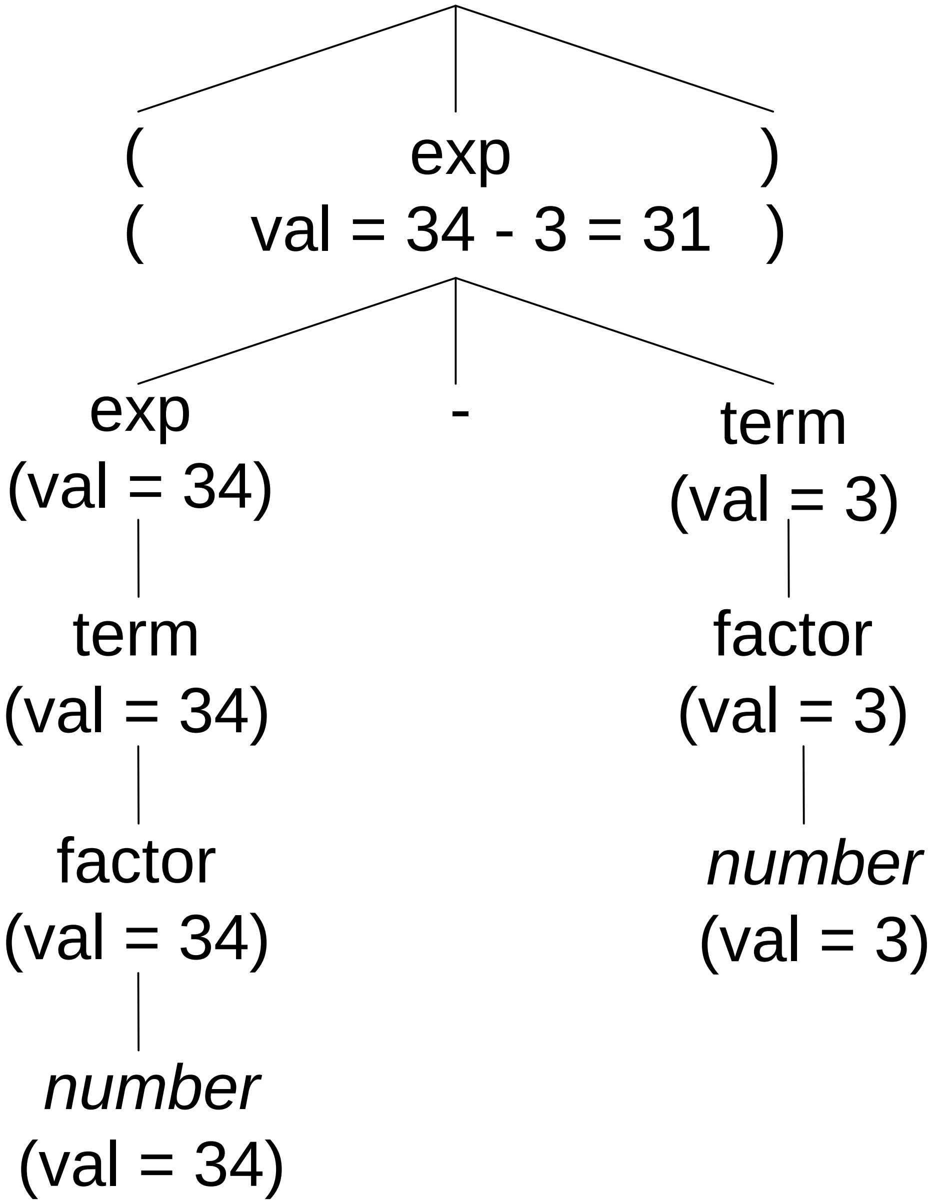

number
val = 34 * 10 + 5 = 345
number
val = 3 * 10 + 4 = 34
digit
(val = 5)
number
(val = 3)
digit
(val = 4)
5
digit
(val = 3)
4
3

(* synthetic attribute *)

Grammar Rule	Semantic Rules
exp_1 -> exp_2 + term	exp_1 .val = exp_2.val + term.val
exp_1 -> exp_2 - term	exp_1 .val = exp_2.val - term.val
exp -> term	exp.val = term.val
$term_1$ -> $term_2$ * factor	$term_1$.val = $term_2$.val * factor.val
term -> factor	term.val = factor.val
factor -> (exp)	factor.val = exp.val
factor -> ***number***	factor.val = ***number***.val


exp
(val = 1302)
term
(val = 31 * 42 = 1302)
term
(val = 31)
*
factor
(val = 42)
factor
(val = 31)
number
(val = 42)

(接下頁)


(
exp
)
(
val = 34 - 3 = 31
)
exp
(val = 34)
-
term
(val = 3)
term
(val = 34)
factor
(val = 3)
factor
(val = 34)
number
(val = 3)
number
(val = 34)

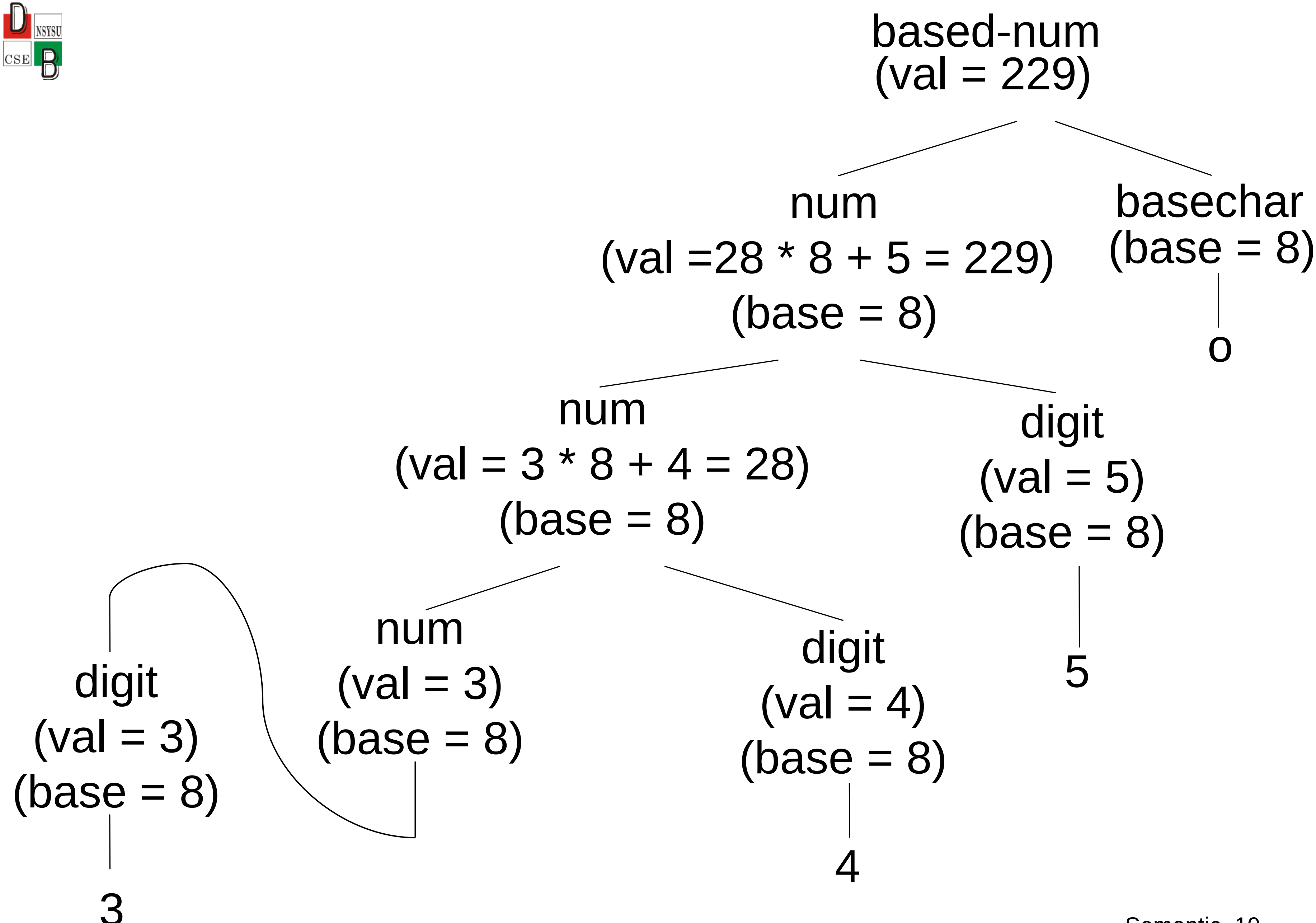
(* inherited attribute *)

Grammar Rule	Semantic Rules
decl -> type var_list	var_list.dtype = type.dtype
type -> ***int***	type.dtype = integer
type -> ***float***	type.dtype = real
var_list_1 -> ***id***, var_list_2	***id***.dtype = var_list_1.dtype var_list_2.dtype = var_list_1.dtype
var_list -> ***id***	***id***.dtype = var_list.dtype

Grammar Rule	Semantic Rules
S -> exp	S.etype = ***if*** exp.isFloat ***then*** float ***else*** int S.val = exp.val
exp_1 -> exp_2 / exp_3	exp_1.isFloat= exp_2.isFloat ***or*** exp_3.isFloat exp_2.etype = exp_1.etype exp_3.etype = exp_1 .etype exp_1.val = ***if*** exp_1.etype = int ***then*** exp_2 .val ***div*** exp_3.val ***else*** exp_2.val / exp_3.val

(接下頁)

exp -> ***num***	exp.isFloat = ***false*** exp.val = ***if*** exp.etype = int ***then num***.val ***else*** Float(***num***.val)
exp -> ***num.num***	exp.isFloat = ***true*** exp.val = ***num.num***.val


based-num
(val = 229)
num
(val =28 * 8 + 5 = 229)
(base = 8)
basechar
(base = 8)
o
num
(val = 3 * 8 + 4 = 28)
(base = 8)
digit
(val = 5)
(base = 8)
5
num
(val = 3)
(base = 8)
digit
(val = 4)
(base = 8)
4
digit
(val = 3)
(base = 8)
3

Grammar Rule	Semantic Rules
based-num -> num.basechar	based-num.val = num.val num.base = basechar.base
basechar -> ***o***	basechar.base = 8
basechar -> ***d***	basechar.base = 10
num_1 -> num_2 digit	num_1.val = ***if*** digit.val = error ***or*** num_2.val = error ***then*** error ***else*** num_2.val * num_1.base + digit.val num_2.base = num_1.base digit.base = num_1.base

num -> digit	num.val = digit.val digit.base = num.base
digit -> ***0***	digit.val = 0
digit -> 1	digit.val = 1
………...	……………..
digit -> 7	digit.val = 7
digit -> 8	digit.val = ***if*** digit.base = 8 ***then*** error ***else*** 8
digit -> 9	digit.val = ***if*** digit.base = 8 ***then*** error ***else*** 9

(* inherited attribute *)

Grammar Rule	Semantic Rules
decl -> type var_list	var_list.dtype = type.dtype
type -> ***int***	type.dtype = integer
type -> ***float***	type.dtype = real
var_list_1 -> var_list_2 , ***id***	insert(***id***.name, var_list_1.dtype) var_list_2.dtype = var_list_1.dtype
var_list -> ***id***	insert(***id***.name, var_list.dtype)

(* the resulting data type; synthetic attribute *)

產生規則	語意規則
E -> num	E.type := integer
E-> num . num	E.type := real
E -> id	E.type := lookup(***id***.entry)
E-> E_1 ***op*** E_2	E.type := ***if*** E_1.type = integer ***and***
	E_2.type =integer ***then*** integer
	else if E_1.type = integer ***and***
	E_2.type = real ***then*** real
	else if E_1.type = real ***and***
	E_2.type = integer *then* real

(接下頁)

產生規則	語意規則
	else if E_1.type = real ***and*** E_2.type = real *then* real ***else*** type_error

(the resulting data type: synthetic attribute *)

E -> ***id***	{ E.type := lookup(id.entry) }
E -> E_1 ***mod*** E_2	{ E.type := ***if*** E_1.type = integer ***and*** E_2.type = integer ***then*** integer ***else*** type_error }
E -> E_1 [E_2]	{ E.type := ***if*** E_2.type = integer ***and*** E_1.type = array(s, t) ***then*** t ***else*** type_error }
E -> $E_1\uparrow$	{ E.type := ***if*** E_1.type = pointer(t) ***then*** t ***else*** type_error }
S -> ***id*** := E	{ S.type := ***if id***.type = E.type ***then*** void ***else*** type_error }

(接下頁)

S -> ***if*** *E* ***then*** S_1 { S.type := ***if*** *E*.type = boolean ***then*** S_1.type ***Else*** type_error }

S -> ***while*** *E* ***do*** S_1 { S.type := ***if*** *E.*type = boolean ***then*** S_1.type ***else*** type_error }

S -> S_1 ; S_2 { S.type := ***if*** S_1.type = void ***and*** S_2.type = void ***then*** void ***else*** type_error }

	Parsing Stack	Input	Parsing Action	Value Stack	Semantic Action
1	$	3*4+5 $	shift	$	
2	$ ***n***	*4+5 $	reduce $E \rightarrow$ ***n***	$ ***n***	E.val = ***n***.val
3	$ *E*	*4+5 $	shift	$ 3	
4	$ *E* *	4+5 $	shift	$ 3 *	
5	$ *E* * ***n***	+5 $	reduce $E \rightarrow$ ***n***	$ 3 * ***n***	E.val = ***n***.val
6	$ *E* * *E*	+5 $	reduce $E \rightarrow E * E$	$ 3 * 4	E_1.val = E_2.val * E_3.val

(接下頁)

7	$ *E*	+5 $	shift	$ 12	
8	$ *E*+	5 $	shift	$ 12 +	
9	$ *E* + ***n***	$	reduce $E \rightarrow \boldsymbol{n}$	$ 12 + ***n***	*E*.val = ***n***.val
10	$ *E* + *E*	$	reduce $E \rightarrow E + E$	$ 12 + 5	E_1.val = E_2.val + E_3.val
11	*$E*	$		$ 17	

(* structure information)

P -> D { offset := 0 }

D -> D ; D

D-> ***id*** : T { enter (***id***.name, T.type, offset);
offset := offset + T.width }

T -> ***integer*** { T.type := integer;
T.width := 4 }

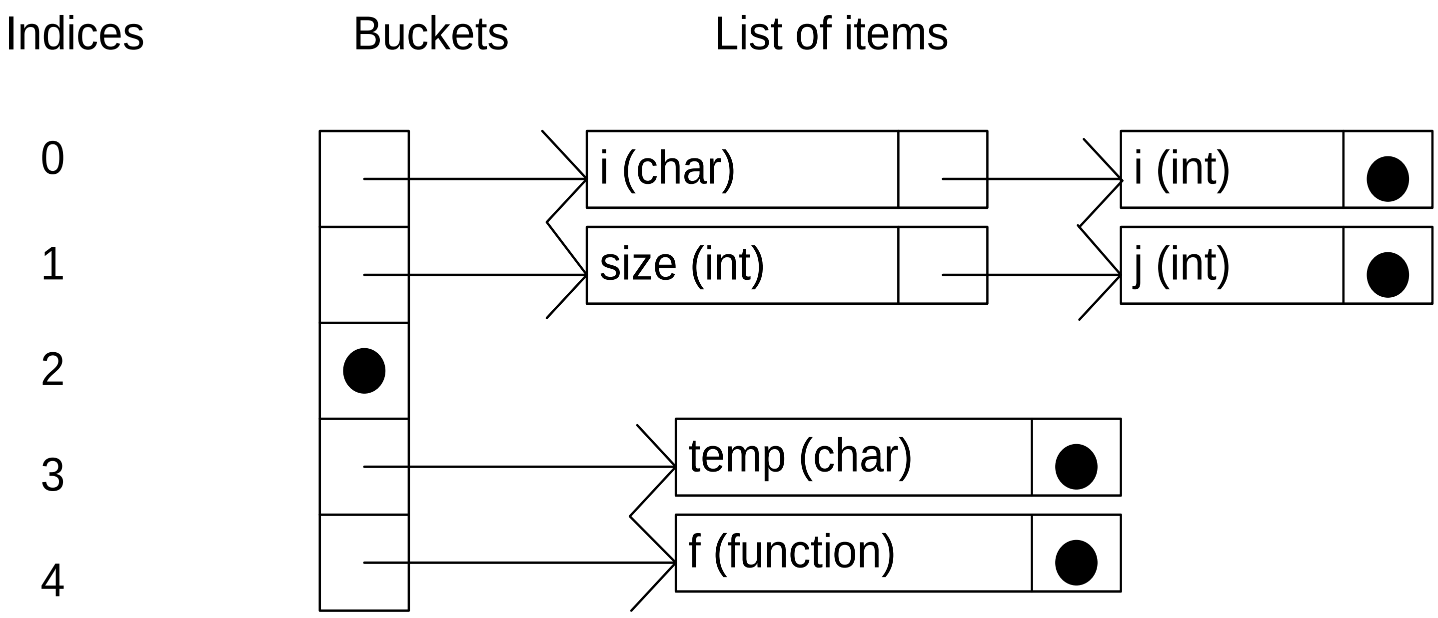
T -> ***real*** { T.type := ***real***;
T.width := 8 }

T -> ***array*** [***num***] ***of*** T1 { T.type := array(***num***.val, T_1.type);
T.width := ***num***.val * T_1.width }

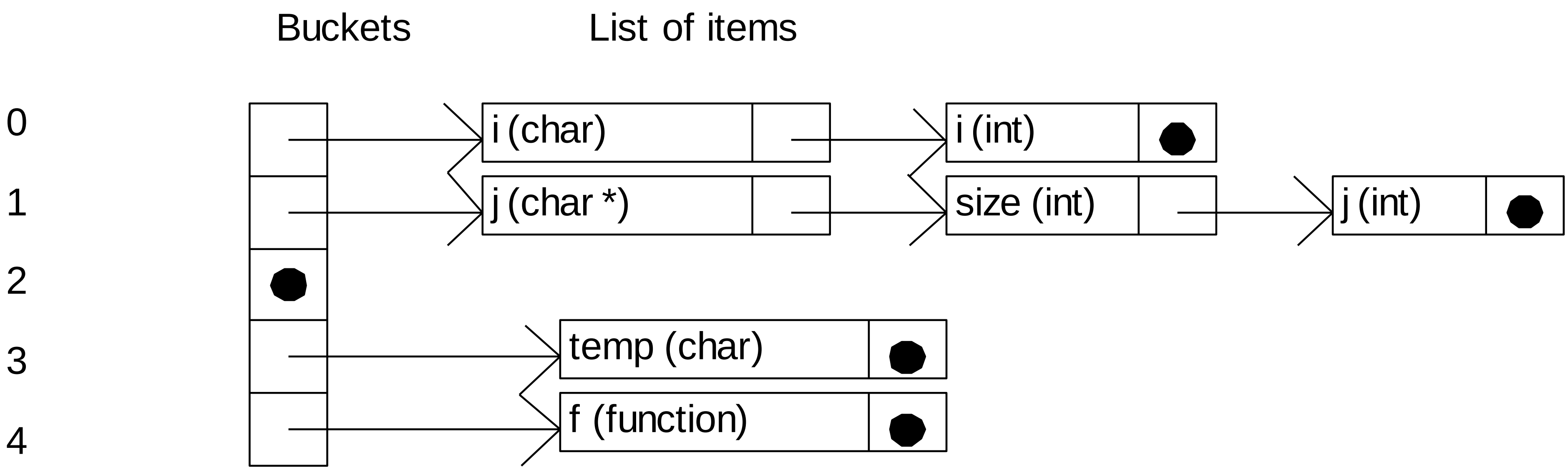
T -> $\uparrow T_1$ { T.type := pointer(T_1.type);
T.width := 4 }

(* symbol table structure for implementing scoping rule.)

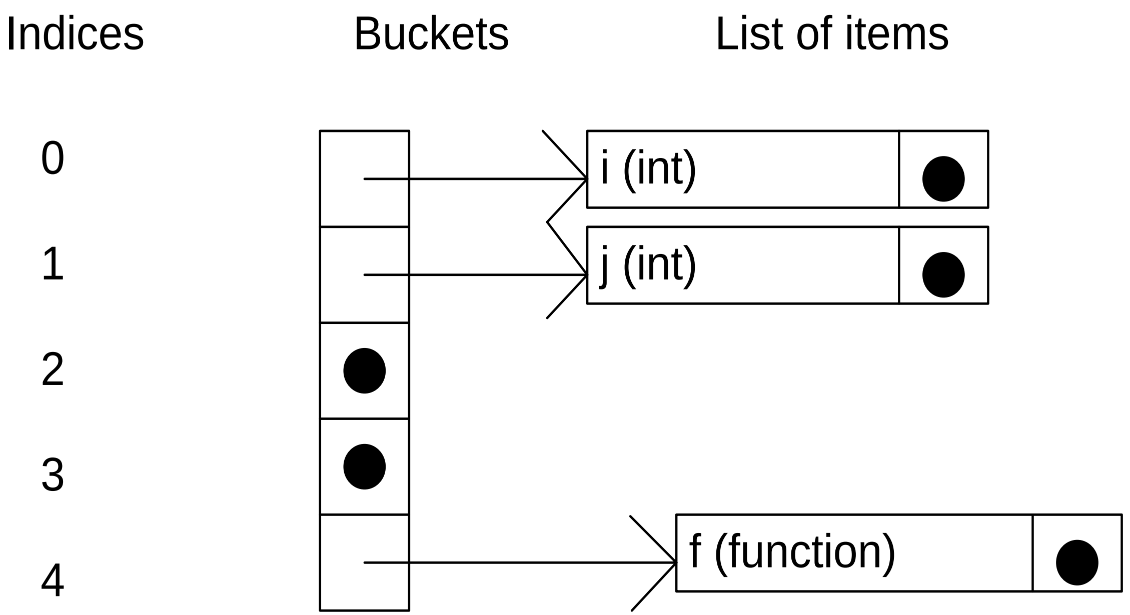
```
Program Ex;
var i , j : integer;
function  f (size: integer): integer;
var i , temp: char;
        procedure  g;
        var j: real;
        begin
                …
        end;
        procedure  h;
        var j: ^char;
        begin
                …
        end;
begin ( * f *)
        …
end;
begin ( * main program *)
        …
end.
```



(a) After processing the declarations of the body of f



(b) After processing the declarations of the second nested compound statement within the body of f



(c) After exiting the body of f (and deleting its declarations)