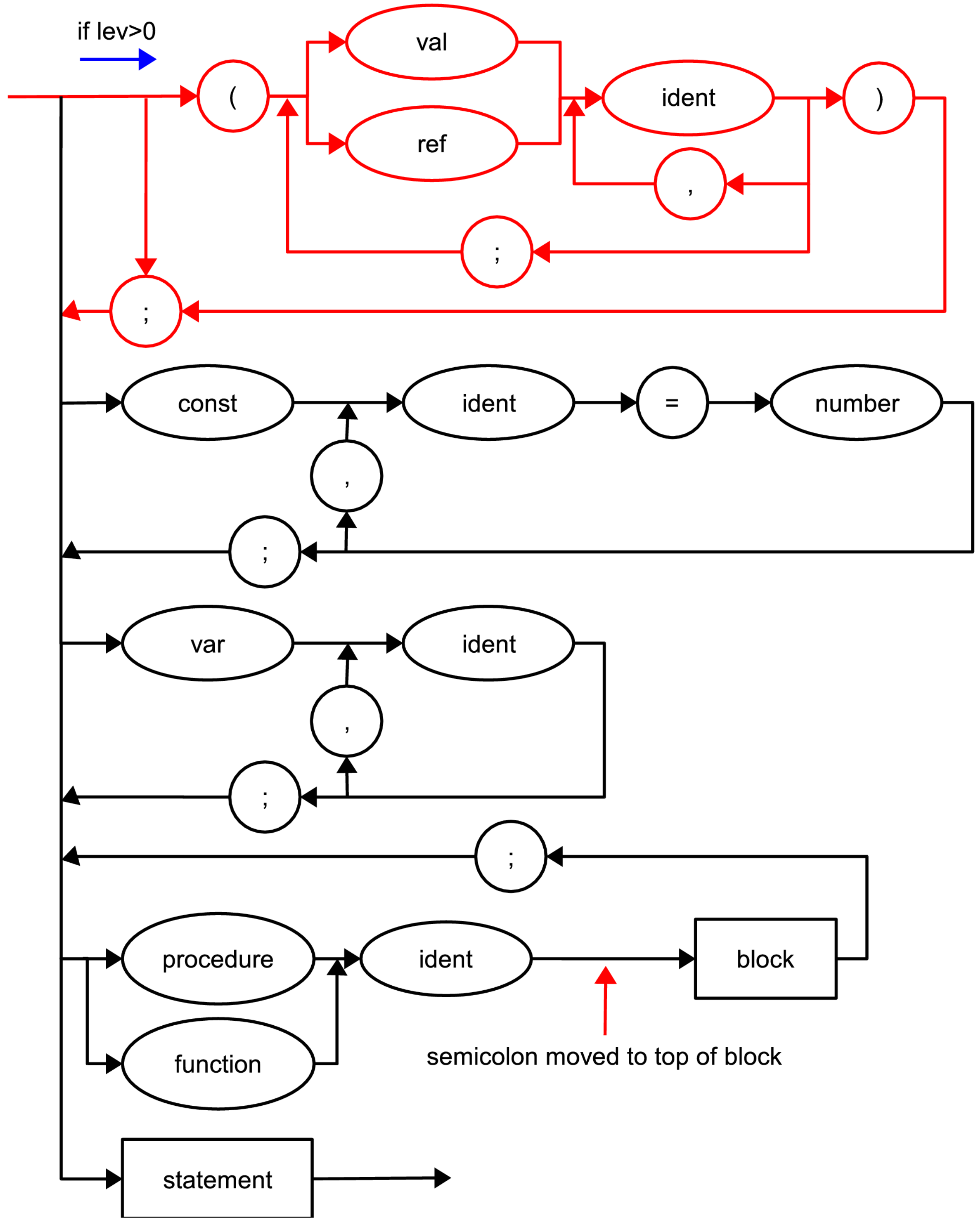
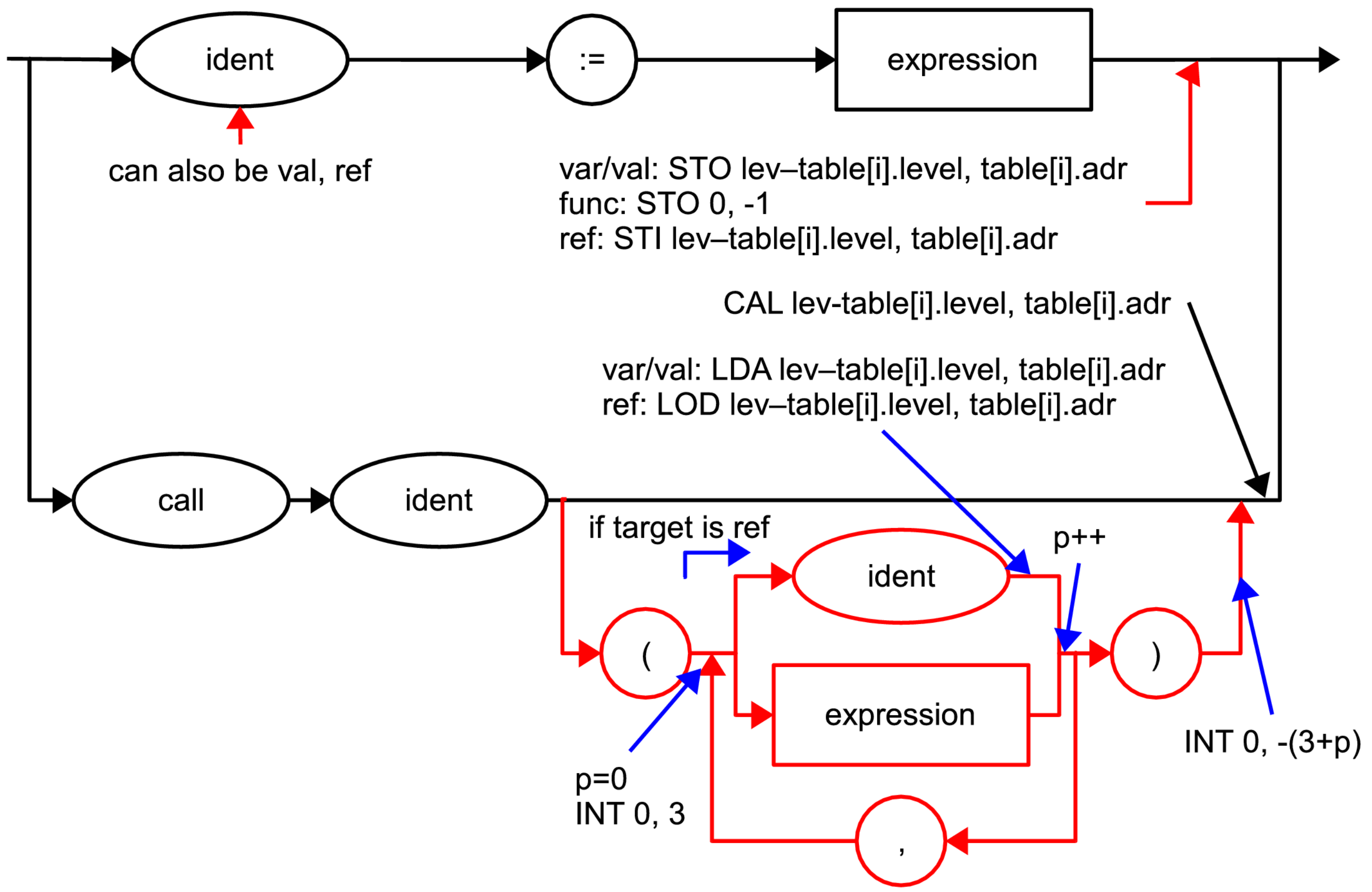
Syntax Diagrams (adding Value and Reference Parameters)

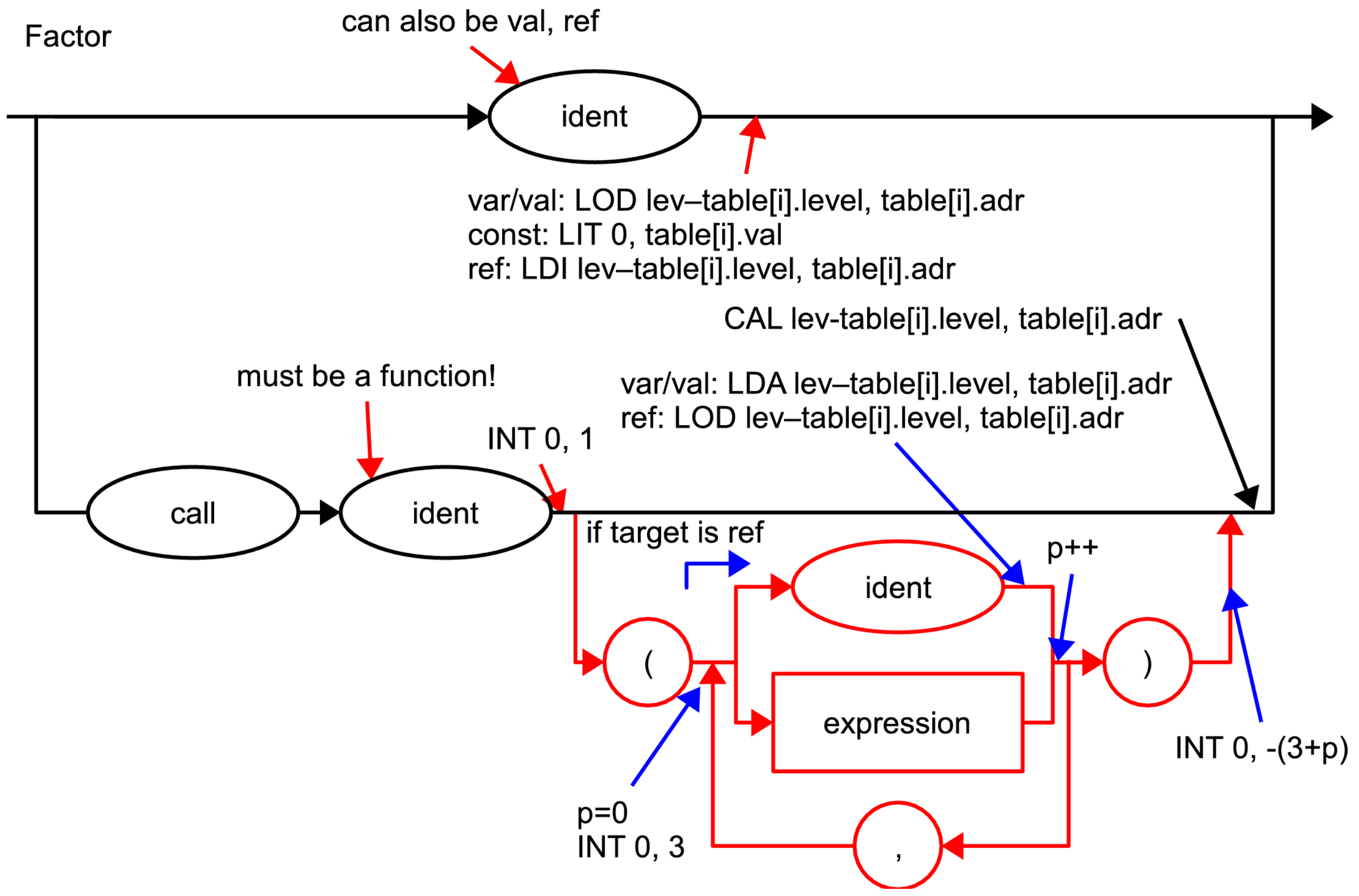
Block



Statement


ident
:=
expression
can also be val, ref
var/val: STO lev–table[i].level, table[i].adr
func: STO 0, -1
ref: STI lev–table[i].level, table[i].adr
CAL lev-table[i].level, table[i].adr
var/val: LDA lev–table[i].level, table[i].adr
ref: LOD lev–table[i].level, table[i].adr
call
ident
if target is ref
ident
p++
(
expression
)
INT 0, -(3+p)
p=0
INT 0, 3
,

Factor


can also be val, ref
ident
var/val: LOD lev–table[i].level, table[i].adr
const: LIT 0, table[i].val
ref: LDI lev–table[i].level, table[i].adr
CAL lev-table[i].level, table[i].adr
must be a function!
var/val: LDA lev–table[i].level, table[i].adr
ref: LOD lev–table[i].level, table[i].adr
INT 0, 1
call
ident
if target is ref
ident
p++
(
expression
)
INT 0, -(3+p)
p=0
INT 0, 3
,

we need to add two new types to the symbol table (VAL, REF)
also add symbols and reserved words

how do we know whether a parameter is VAL/REF in the ST?

- have an array of bools for each procedure/function parameter:
    - 0/F->VAL; 1/T->REF
    - table[i].params[j] where i is the procedure/function ST index and j is the parameter index

we should not be able to pass constants or numbers to REF parameters

parameter loading

- value: LOD (normal load)
- reference: LDI (a new load indirect)

parameter storing

- value: STO (normal store)
- reference: STI (a new store indirect)

calling:

caller / callee	**VAL**	**REF**
VAL	LOD	LDA
REF	LDI	LOD
VAR	LOD	LDA
CONST	LIT	---
NUM	LIT	---

- LDA (a new load address)
- note how VAL and VAR are similar—we can simply treat them the same!

so we need 3 new instructions

- remember that base(x) finds the base, x levels down
- existing instructions:

```
LOD: t++;
     s[t] = s[base(table[i].level)+table[i].adr];
STO: s[base(table[i].level)+table[i].adr] = s[t];
     t--;
```

- new instructions:

```
STI: s[s[base(table[i].level)+table[i].adr]] = s[t];
     t--;
LDI: t++;
     s[t] = s[s[base(table[i].level)+table[i].adr]];
LDA: t++;
     s[t] = base(table[i].level)+table[i].adr;
```

we must allow declaration of value and reference parameters
they just behave like normally declared variables

- the first parameter is at an offset of 3 in the segment, the second at 4, etc

- additional declared variables in the block increment from there
- it's hard to parse for parameters in the normal block syntax diagram
    - because the level of parameters is incorrect
    - and also the variable address offset is only valid for the current block
- so let's just move that part to the top of block and allow “entrance” only if lev>0
    - which means we're in the current procedure or function
    - we also move the semicolon after the procedure/function ident to the top of block
    - now we can just use dx as normal to set the correct address!
    - one additional thing is to set the parameter boolean of the procedure/function properly
        - we do this when we enter the parameters in the ST

          ```
          if (sym == valsym)
          {
                enter(value);
                table[tabinx0].params[dx-4] = false;
          }
          ...
          ```
        - why dx-4?
            - dx starts at 3
            - bool params starts at 0 (we need to shift it back)
            - but dx increments right after we add a val/ref parameter
            - so if we modify the param array right after adding a parameter
                - we need dx-3-1 = dx-4
        - of course we could reverse the two instructions and it would be dx-3 (right?)
    - for the designer (geek), we put it where it is most efficient and easiest
    - for the user (ignoramus), we put it where s/he expects it to be

we then have to figure out how to handle storing to val/ref within procedures and functions

- this is done in statement
    - we modify ident := <expression> to also allow assigning to VAL/REF
    - if the ident is variable/value: STO lev-table[i].level, table[i].adr
    - if the ident is a function: STO 0, -1 (so long as we're in the proper function!)
    - if the ident is REF: STI lev-table[i].level, table[i].adr

we also need to figure out how to handle loading of val/ref within procedures and functions

- this is done in factor
    - we modify ident to allow loading VAL/REF
    - if the ident is variable/value: LOD lev-table[i].level, table[i].adr
    - if the ident is const: LIT 0, table[i].val
    - if the ident is ref: LDI lev-table[i].level, table[i].adr

we need to allow calling procedures and functions while passing parameters

- we do this in both statement (for procedures) and factor (functions)
    - in either case, we preliminarily INT 0, 3 to temporarily make space for SL, DL, RA
        - this is like a “fake call” just to load parameters
    - then we LOD, LIT, LDI, or LDA depending on type
    - this really depends on two things:
        - what type is the variable we're trying to pass
        - what type is the procedure/function parameter (receiver)
    - if the receiver is VAL then just call expression (we only need either LOD or LDI)
    - if the receiver is REF then we use LDA for VAL, VAR and LOD for REF
    - once we're done loading parameters, we decrement the stack to undo the fake call
        - INT 0, -(3+# parameters)