

Compiler Construction

Theoretical Exercise 4: IR
Practical Exercise 4

Michael Engel

TE4.1 Rules for constructing a DAG

1. In a DAG,

- Interior nodes always represent the operators.
- Exterior nodes (leaf nodes) always represent the names, identifiers or constants.

2. While constructing a DAG,

- A check is made to find if there exists any node with the same value.
- A new node is created only when there does not exist any node with the same value.
- This action helps in detecting the common sub-expressions and avoiding the re-computation of the same.

3. The assignment instructions of the form $x:=y$ are not performed unless they are necessary.

TE4.1 Example for DAG construction

Expression: $(a + b) \times (a + b + c)$

Three Address Code for the given expression is:

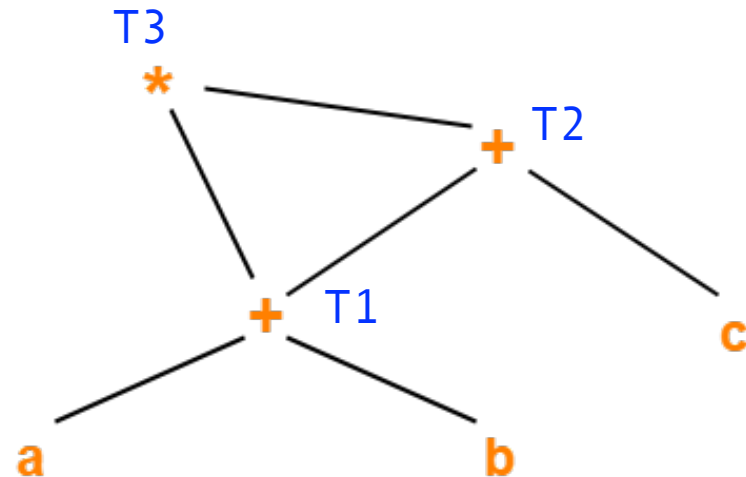
$T1 = a + b$

$T2 = T1 + c$

$T3 = T1 \times T2$

We observe:

- The common sub-expression $(a+b)$ has been expressed into a single node in the DAG.
- The computation is carried out only once and stored in the identifier T1 and reused later.

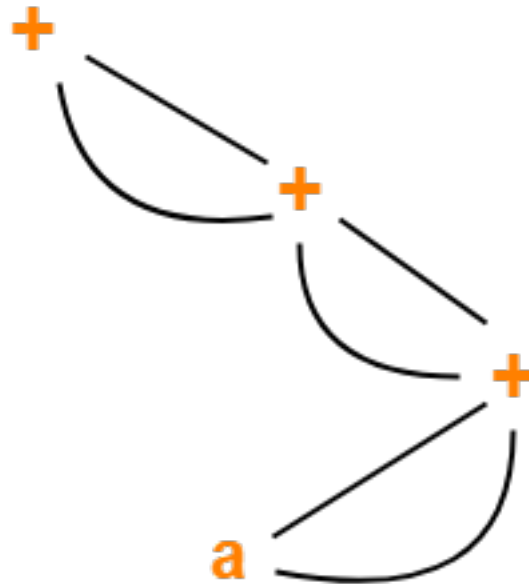


Directed Acyclic Graph

TE4.1 Example for DAG construction

Expression:

$((a + a) + (a + a)) + ((a + a) + (a + a))$



Directed Acyclic Graph

TE4.1 DAGs

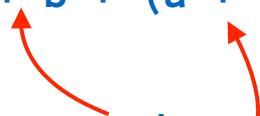
- a. Construct the DAG for the following expressions.
Assume that the + operator is left-associative:

$$a + b + (a + b)$$

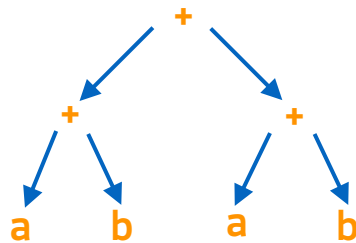
*Left-associative operators **of the same precedence** are evaluated in order from left to right*

TAC for
 $a + b + (a + b)$

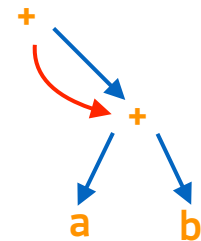
T1 = a + b
T2 = a + b
T3 = T1 + T2



AST for
 $a + b + (a + b)$



DAG for
 $a + b + (a + b)$



TE4.1 DAGs

- b. Construct the DAG for the following expressions.
Assume that the + operator is left-associative:

$a + b + a + b$

TAC for

$a + b + a + b$

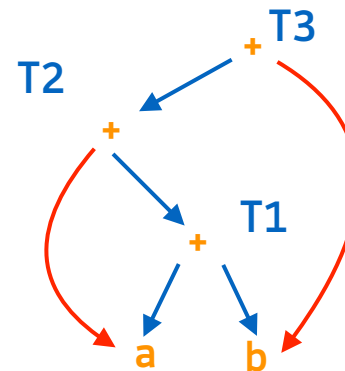
$T1 = a + b$

$T2 = T1 + a$

$T3 = T2 + b$

DAG for

$a + b + a + b$



TE4.1 DAGs

c. Construct the DAG for the following expressions.

Assume that the + operator is left-associative:

$a + a + (a + a + a + (a + a + a + a))$

TAC for

$a + a + (a + a + a + (a + a + a + a))$

$T1 = a + a$

$T2 = a + a (= T1)$

$T3 = T2 + a$

$T4 = a + a (= T1)$

$T5 = T4 + a$

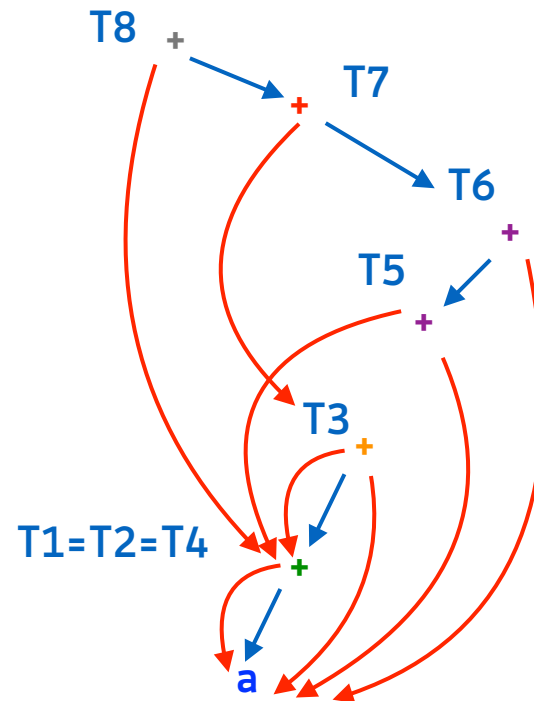
$T6 = T5 + a$

$T7 = T3 + T6$

$T8 = T1 + T7$

DAG for

$a + a + (a + a + a + (a + a + a + a))$



TE4.2 Intermediate representations

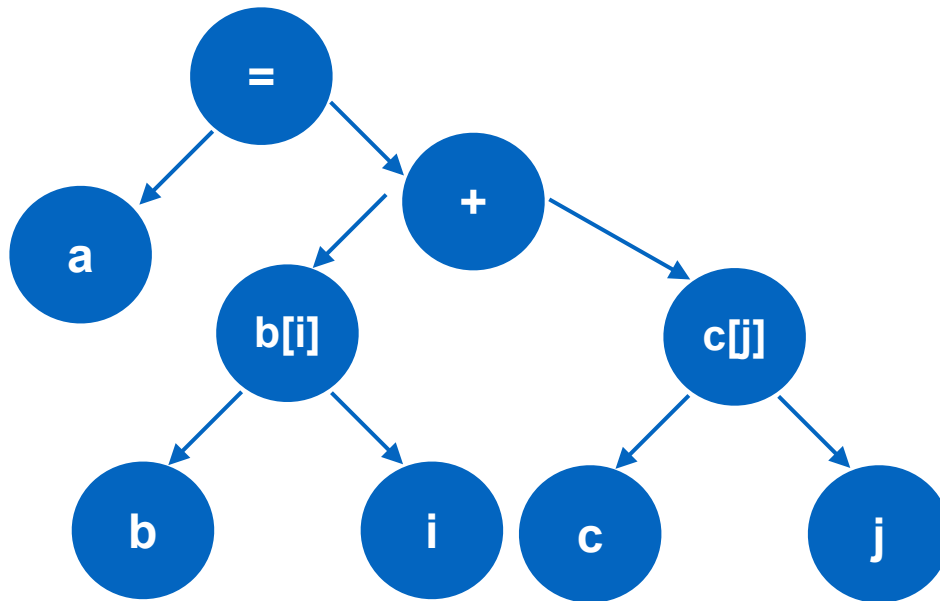
Translate each of the following expressions into

- a syntax tree

- as well as quadruples

for the three-address code (TAC) IR as described in lecture 12:

a. $a = b[i] + c[j]$



0) $t1 = b[i]$
1) $t2 = c[j]$
2) $t3 = t1 + t2$
3) $a = t3$

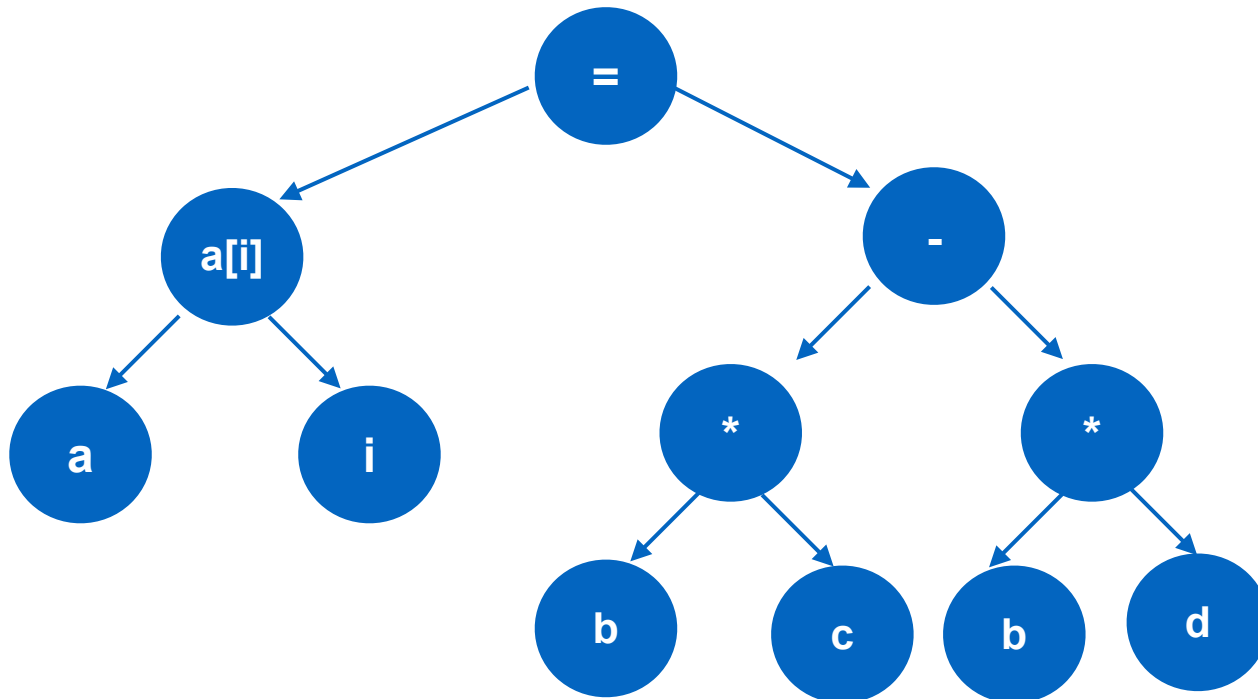
TE4.2 Intermediate representations

Translate each of the following expressions into

- a syntax tree
- as well as quadruples

for the three-address code (TAC) IR as described in lecture 12:

b. $a[i] = b * c - b * d$



```
0) t1 = b * c
1) t2 = b * d
2) t3 = t1 - t2
3) a[i] = t3
```

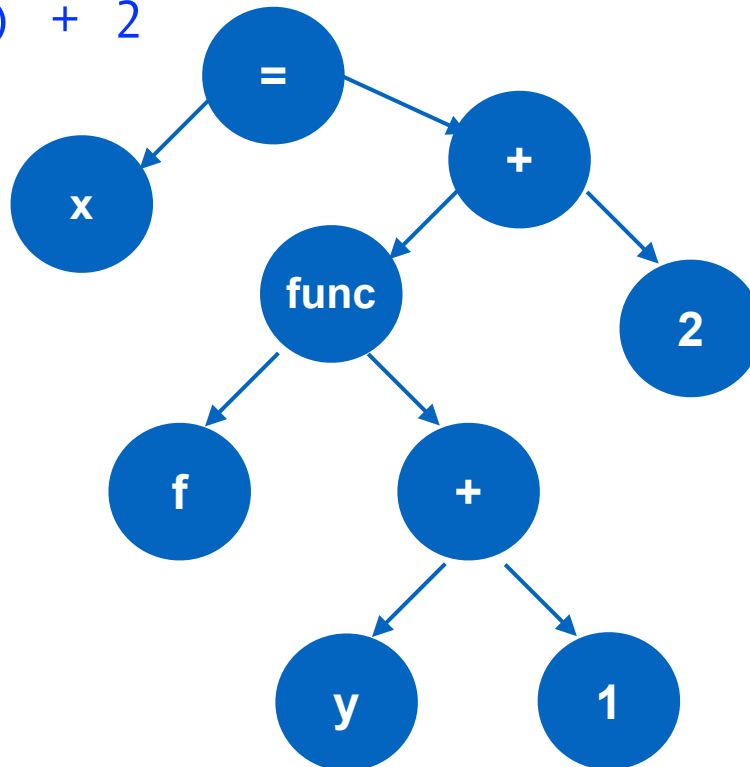
TE4.2 Intermediate representations

Translate each of the following expressions into

- a syntax tree
- as well as quadruples

for the three-address code (TAC) IR as described in lecture 12:

c. $x = f(y+1) + 2$



```
0) t1 = y + 1
1) param t1
2) t2 = call f
3) t3 = t2 + 2
4) x = t3
```

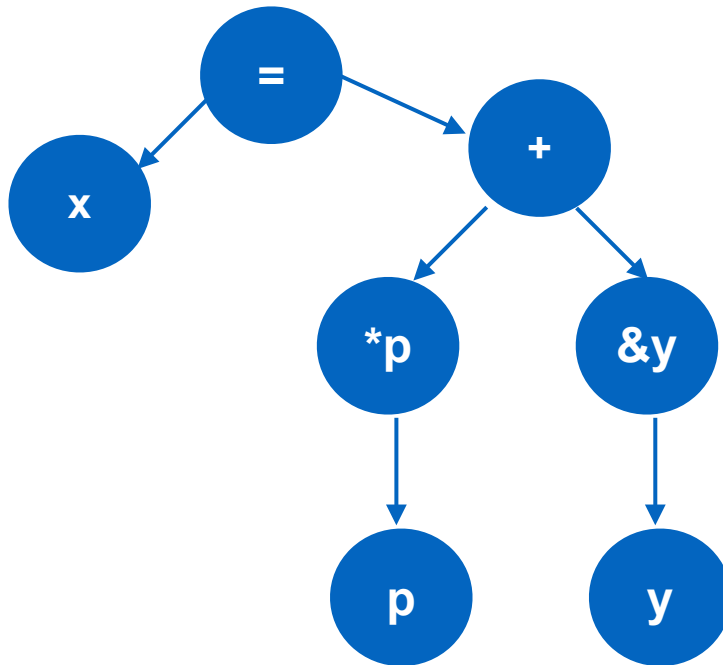
TE4.2 Intermediate representations

Translate each of the following expressions into

- a syntax tree
- as well as quadruples

for the three-address code (TAC) IR as described in lecture 12:

d. $x = *p + \&y$

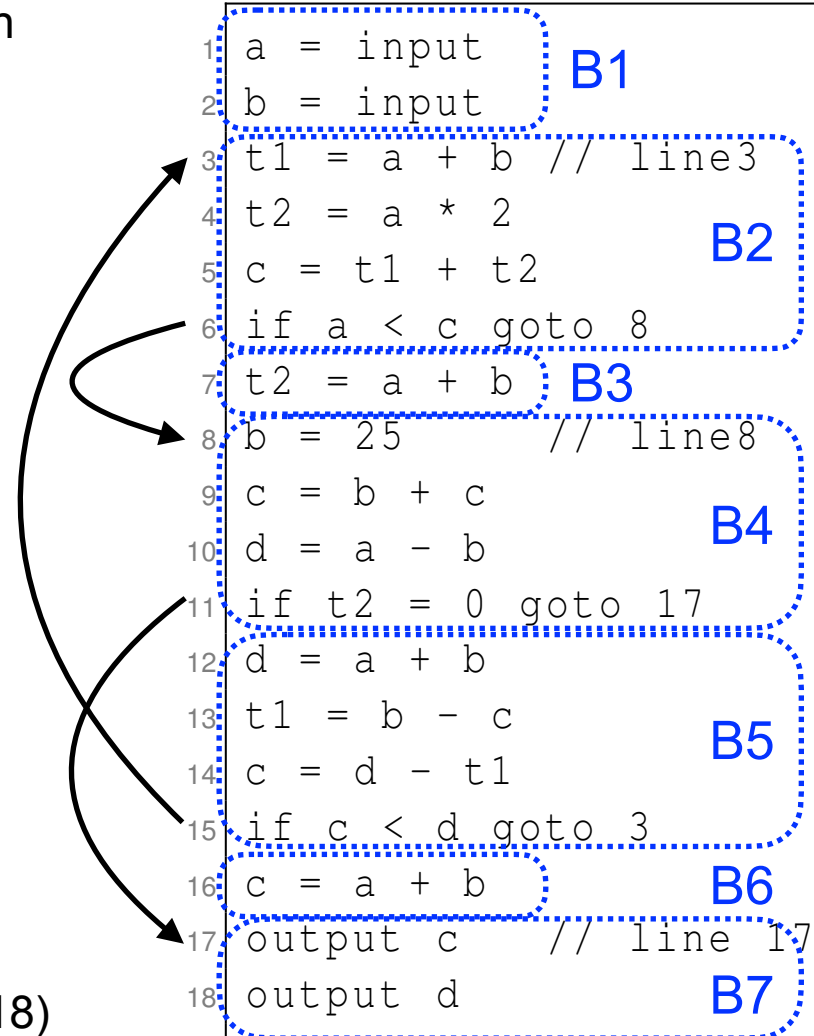


0) $t1 = *p$
1) $t2 = \&y$
2) $t3 = t1 + t2$
3) $x = t3$

TE4.3 Basic blocks and TAC analysis

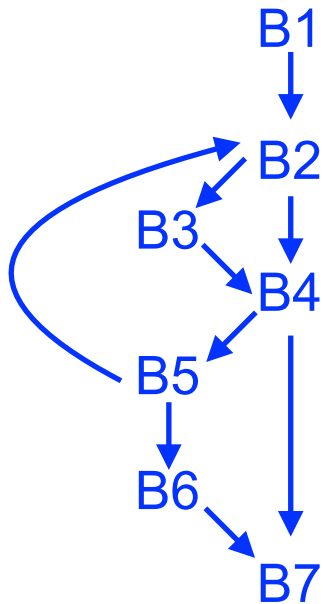
a. Indicate where new basic blocks start. For each basic block, give the line number such that the instruction in the line is the first one of that block.

- B1 starts at line 1
- B1 ends at line 2 (line 3 is branch target)
- B2 starts at line 3
- B2 ends at line 6 (conditional branch)
- B3 starts at line 7
- B3 ends at line 7 (line 8 is branch target)
- B4 starts at line 8
- B4 ends at line 11 (conditional branch)
- B5 starts at line 12
- B5 ends at line 15 (conditional branch)
- B6 starts at line 16
- B6 ends at line 16 (line 17 is branch target)
- B7 starts at line 17 and runs until the end (line 18)



TE4.3 Basic blocks and TAC analysis

b. Give names B1, B2, ... for the program's basic blocks in the order the blocks appear in the given listing. Draw the control flow graph making use of those names. Do not put in the code into the nodes of the flow graph, the labels Bi are sufficient.



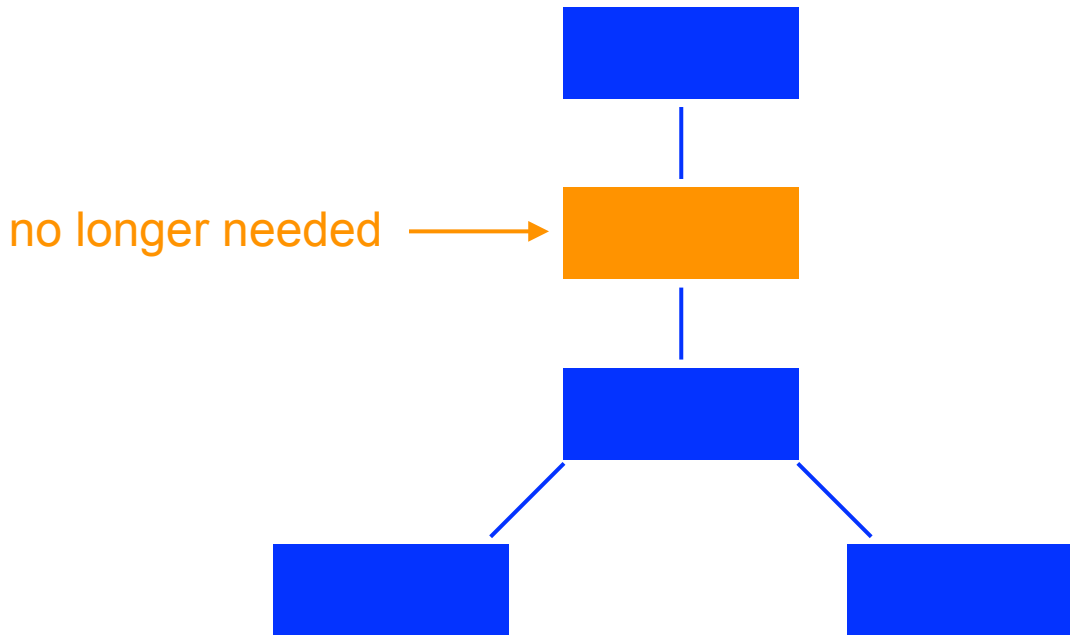
```
1 a = input          B1
2 b = input
3 t1 = a + b // line3
4 t2 = a * 2        B2
5 c = t1 + t2
6 if a < c goto 8
7 t2 = a + b        B3
8 b = 25 // line8
9 c = b + c
10 d = a - b        B4
11 if t2 = 0 goto 17
12 d = a + b
13 t1 = b - c       B5
14 c = d - t1
15 if c < d goto 3
16 c = a + b        B6
17 output c // line 17
18 output d        B7
```

PE4: Simplifying trees

- What needs to be done this time is a recursive traversal of the constructed syntax tree
- Comparatively speaking, this should be simple
 - At least there is far less typing :)
 - There will be a bit more to do again on the remaining ones
- It is highly variable how comfortable everyone is with recursion
- We'll be traversing trees up, down, left, right and center for the rest, so this is a chance to grow comfortable with it

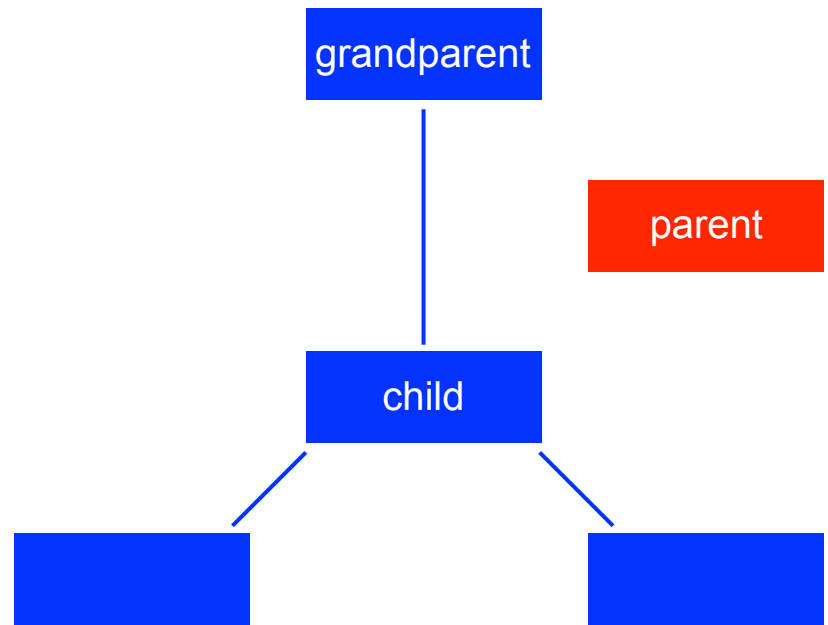
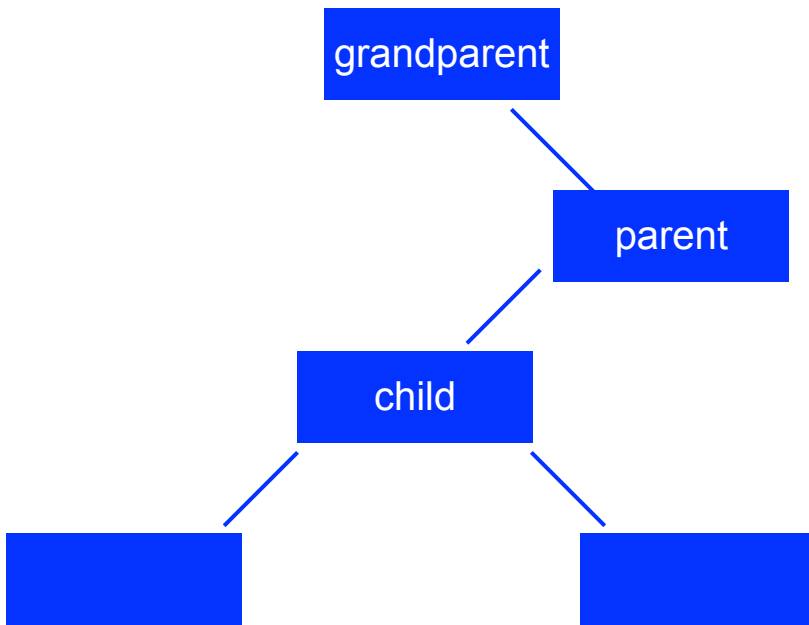
PE4: Single nodes (syntax artifacts)

- Some of our nodes are just there because it's easier to write the grammar that way
- Afterwards, this kind of structure doesn't tell us anything about the program



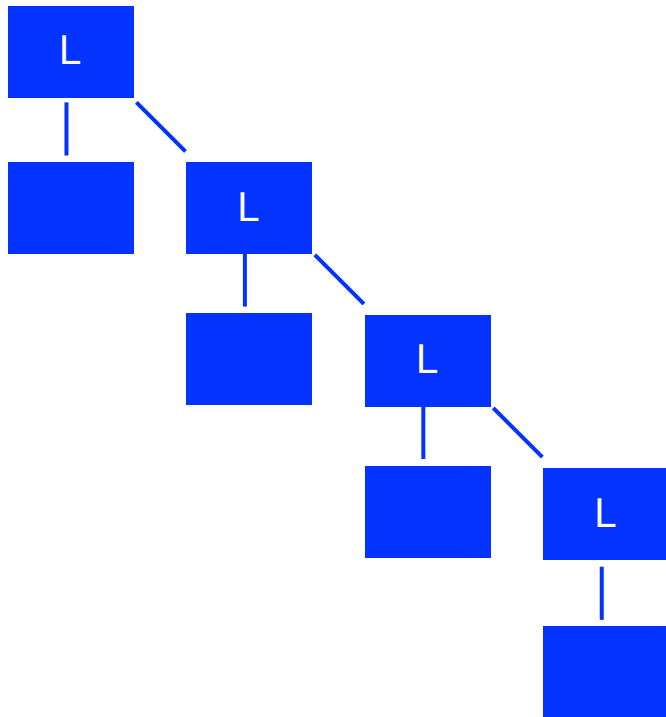
PE4: Single nodes (syntax artifacts)

- If you reach nodes like this in any kind of traversal:
 - disassociate them
 - connect the child
 - and delete them

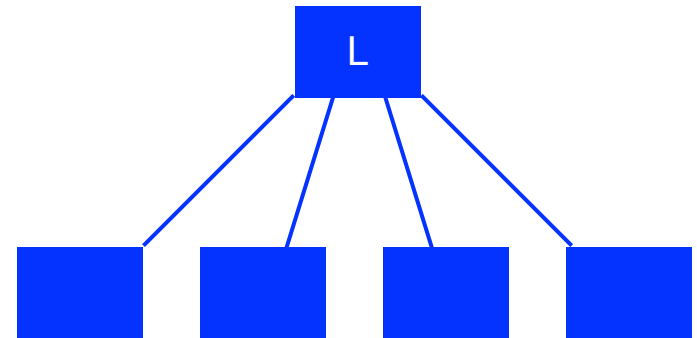


PE4: Lists

- Syntax results in structures like these:



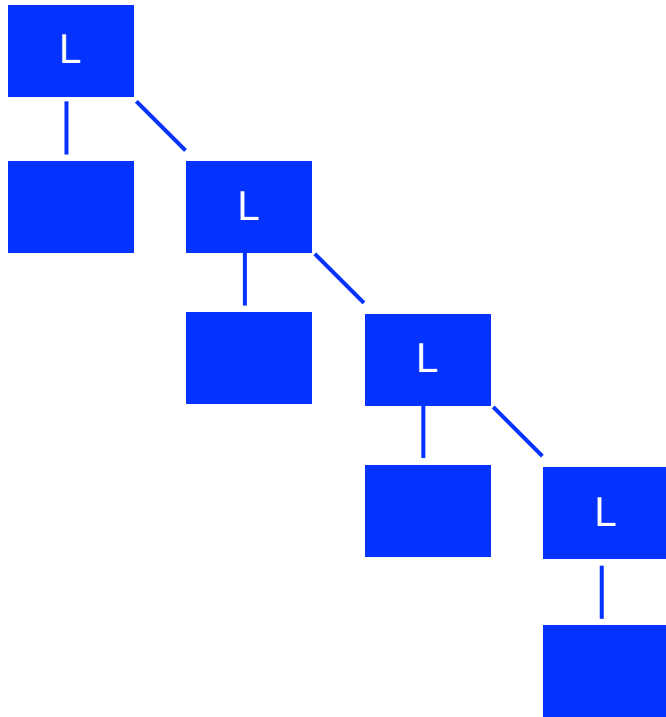
- We only need these:



Traversing the former is a lot of extra work if you only want the elements!

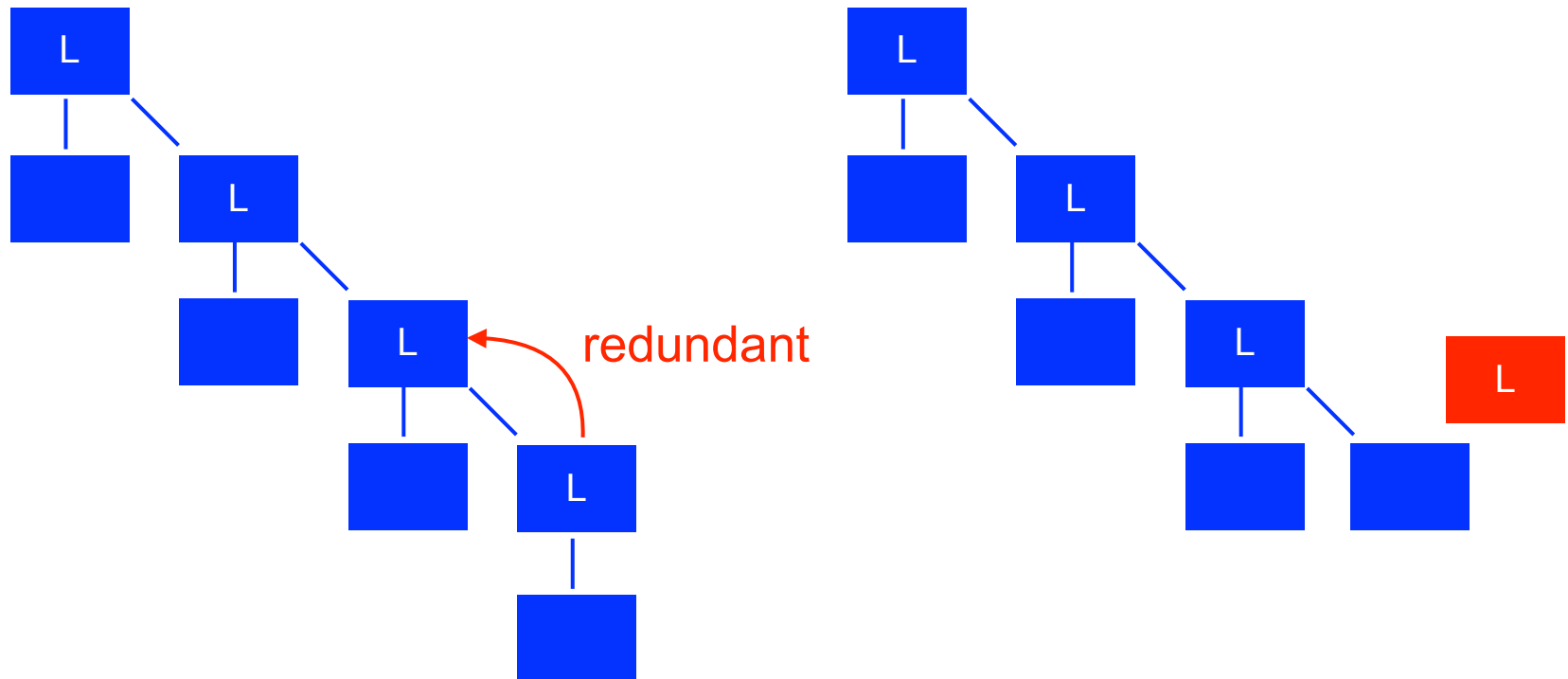
PE4: Lists step 1: recur to the bottom

- You've found it when the structure doesn't repeat



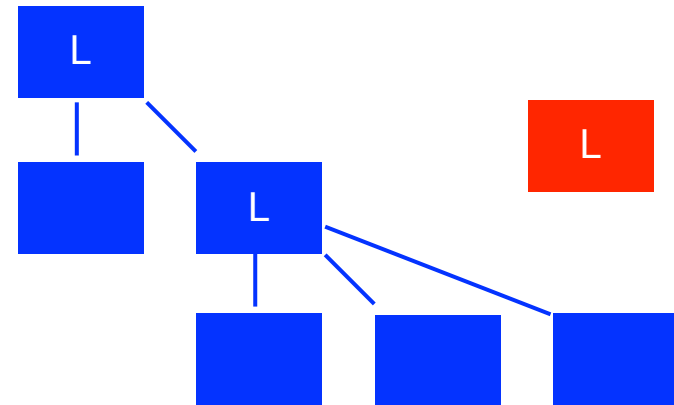
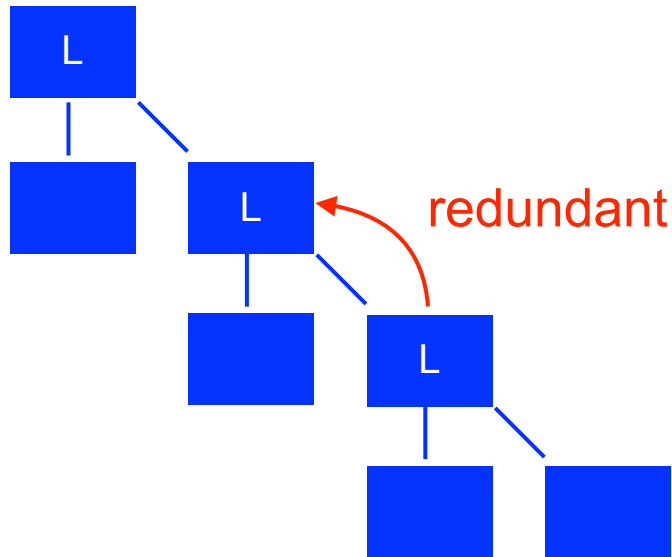
PE4: Lists step 2: on the way back

- When there are two or more children, remove redundant node and keep the rest:



PE4: Lists step 3: continue...

- When there are two or more children below, continue to remove redundant node and keep the rest:



PE4: Expressions

- Again, if you recur to the bottom first and handle the rewriting during the backtrack phase, the tree above reduces to the same basic cases as the lowest levels:

