# 1 Semistructured Merge

## 1.1 Early Concepts

1. Every node's origin is set to UNKNOWN beforehand

## 1.2 Merge Algorithms

**Algorithm 1:** Merge Files

**Input:** l, b, r, o

**if** $l.content = b.content$ **then**
  $o.content \leftarrow r.content$;
**else if** $b.content = r.content \vee l.content = r.content$ **then**
  $o.content \leftarrow l.content$;
**else**
  $L \leftarrow$ `fileToTree`$(l)$;
  $B \leftarrow$ `fileToTree`$(b)$;
  $R \leftarrow$ `fileToTree`$(r)$;
  $M \leftarrow$ `mergeTrees`$(L, B, R)$;
  $H \leftarrow$ `getActiveHandlers()`;
  **foreach** $h \in H$ **do**
    $h.handle(M)$;
  **end**
  $o.content \leftarrow$ `treeToText`$(M)$;
**end**

**Algorithm 2:** Merge Trees

**Input:** L, B, R
**Output:** result of merging left, base and right trees

$L.origin = LEFT$;
$B.origin = BASE$;
$R.origin = RIGHT$;
$LB \leftarrow$ `mergeNodes`$(L, B)$;
$M \leftarrow$ `mergeNodes`$(LB, R)$;
$D_L \leftarrow \{b \in B \mid (\neg\exists l \in L)(l.id = b.id)\}$;
$D_R \leftarrow \{b \in B \mid (\neg\exists r \in R)(r.id = b.id)\}$;
$D_B \leftarrow D_L \cap D_R$;
**foreach** $d \in D_B$ **do**
  `removeNode`$(d, M)$;
**end**
`updateLeafBodies`$(M)$;
**return** $M$;

**Algorithm 3:** Update Leaf Bodies

**Input:** T

**foreach** $t \in T.children$ **do**
  `updateLeafBodies`$(t)$;
**end**
**if** $T.children = \emptyset \wedge SEPARATOR \in T.body$ **then**
  $l, b, r \leftarrow split(T.body, SEPARATOR)$;
  $l \leftarrow l - MARKER$;
  $T.body \leftarrow$ `textualMerge`$(l, b, r)$;
**end**

**Algorithm 4:** Merge Nodes

**Input:** A, B
**Output:** result of merging nodes A and B

```
1  if A = null then return B ;
2  if B = null then return A ;
3  if A.type ≠ B.type ∨ A.id ≠ B.id then return null ;
4  M.id ← A.id;
5  M.type ← A.type;
6  M.origin ← B.origin;
7  M.children ← ∅;
8  if A.children = ∅ ∧ B.children = ∅ then
9  |  if MARKER ∈ A.body then
10 |  |  M.body ← A.body + B.body;
11 |  else if A.origin = LEFT ∧ B.origin = BASE then
12 |  |  M.body ← MARKER + A.body + SEPARATOR + B.body + SEPARATOR;
13 |  else if A.origin = LEFT then
14 |  |  M.body ← MARKER + A.body + SEPARATOR + SEPARATOR + B.body;
15 |  else
16 |  |  M.body ← MARKER + SEPARATOR + A.body + SEPARATOR + B.body;
17 |  end
18 |  return M;
19 else if A.children ≠ ∅ ∧ B.children ≠ ∅ then
20 |  foreach b ∈ B.children do
21 |  |  a ← find(a ∈ A.children → a.type = b.type ∧ a.id = b.id);
22 |  |  if a.origin = UNKNOWN then a.origin ← A.origin ;
23 |  |  if b.origin = UNKNOWN then b.origin ← B.origin ;
24 |  |  M.children ← M.children ∪ mergeNodes(a, b, step);
25 |  end
26 |  foreach a ∈ A.children do
27 |  |  b ← find(b ∈ B.children → a.type = b.type ∧ a.id = b.id);
28 |  |  if a.origin = UNKNOWN then a.origin ← A.origin ;
29 |  |  if b = null then M.children ← M.children ∪ a ;
30 |  end
31 |  return M;
32 end
33 return null;
```