

# 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
1 if  $l.content = b.content$  then
2   |  $o.content \leftarrow r.content$ ;
3 else if  $b.content = r.content \vee l.content = r.content$  then
4   |  $o.content \leftarrow l.content$ ;
5 else
6   |  $L \leftarrow fileToTree(l)$ ;
7   |  $B \leftarrow fileToTree(b)$ ;
8   |  $R \leftarrow fileToTree(r)$ ;
9   |  $M \leftarrow mergeTrees(L, B, R)$ ;
10  |  $H \leftarrow getActiveHandlers()$ ;
11  | foreach  $h \in H$  do
12  |   |  $h.handle(M)$ ;
13  | end
14  |  $o.content \leftarrow treeToText(M)$ ;
15 end
```

### Algorithm 2: Merge Trees

```
Input: L, B, R
Output: result of merging left, base and right trees
1  $L.origin = LEFT$ ;
2  $B.origin = BASE$ ;
3  $R.origin = RIGHT$ ;
4  $LB \leftarrow mergeNodes(L, B)$ ;
5  $M \leftarrow mergeNodes(LB, R)$ ;
6  $D_B \leftarrow \{b \in B \mid (\neg \exists l \in L)(b.id = l.id) \wedge (\neg \exists r \in R)(b.id = r.id)\}$ ;
7 foreach  $d \in D_B$  do
8   |  $removeNode(d, M)$ ;
9 end
10  $runTextualMergeOnLeaves(M)$ ;
11 return  $M$ ;
```

### Algorithm 3: Run Textual Merge On Leaves

```
Input: T
1 foreach  $t \in T.children$  do
2   |  $runTextualMergeOnLeaves(t)$ ;
3 end
4 if  $T.children = \emptyset \wedge SEPARATOR \in T.body$  then
5   |  $l, b, r \leftarrow split(T.body, SEPARATOR)$ ;
6   |  $l \leftarrow l - MARKER$ ;
7   |  $T.body \leftarrow textualMerge(l, b, r)$ ;
8 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  $A.type \neq B.type \vee A.id \neq B.id$  then return  $null$  ;
4  $M.id \leftarrow A.id$ ;
5  $M.type \leftarrow A.type$ ;
6  $M.origin \leftarrow B.origin$ ;
7  $M.children \leftarrow \emptyset$ ;
8 if  $A.children = \emptyset \wedge B.children = \emptyset$  then
9   if  $MARKER \in A.body$  then
10      $M.body \leftarrow A.body + B.body$ ;
11   else if  $A.origin = LEFT \wedge B.origin = BASE$  then
12      $M.body \leftarrow MARKER + A.body + SEPARATOR + B.body + SEPARATOR$ ;
13   else if  $A.origin = LEFT$  then
14      $M.body \leftarrow MARKER + A.body + SEPARATOR + SEPARATOR + B.body$ ;
15   else
16      $M.body \leftarrow MARKER + SEPARATOR + A.body + SEPARATOR + B.body$ ;
17   end
18   return  $M$ ;
19 else if  $A.children \neq \emptyset \wedge B.children \neq \emptyset$  then
20   foreach  $b \in B.children$  do
21      $a \leftarrow find(a \in A.children \rightarrow a.type = b.type \wedge a.id = b.id)$ ;
22     if  $a.origin = UNKNOWN$  then  $a.origin \leftarrow A.origin$  ;
23     if  $b.origin = UNKNOWN$  then  $b.origin \leftarrow B.origin$  ;
24      $M.children \leftarrow M.children \cup mergeNodes(a, b, step)$ ;
25   end
26   foreach  $a \in A.children$  do
27      $b \leftarrow find(b \in B.children \rightarrow a.type = b.type \wedge a.id = b.id)$ ;
28     if  $a.origin = UNKNOWN$  then  $a.origin \leftarrow A.origin$  ;
29     if  $b = null$  then  $M.children \leftarrow M.children \cup a$  ;
30   end
31   return  $M$ ;
32 end
33 return  $null$ ;

```

## 2 Renaming Handler

### 2.1 Early Concepts

#### 2.1.1 Possibly renamed without body changes nodes

$$R_{wobc}(T, B) = \{b \in B \mid (\neg \exists t \in T)(t.id = b.id) \wedge (\exists t \in T)(t.body = b.body)\}$$

#### 2.1.2 Possibly deleted or renamed with body changes nodes

$$DR_{wbc}(T, B) = \{b \in B \mid (\neg \exists t \in T)(t.id = b.id \vee t.body = b.body)\}$$

#### 2.1.3 Nodes IDs similarity

$$a.id \approx b.id \leftrightarrow a.id.name = b.id.name \vee a.id.params = b.id.params$$

## 2.2 Match Algorithm

### Algorithm 5: Match Algorithm

**Input:**  $L, B, R, M$   
**Output:** Set of quadruples  $(l, b, r, m)$  consisting of the base node  $b$  and its corresponding left node  $l$ , right node  $r$  and merge node  $m$

```

1  $matches \leftarrow \emptyset$ ;
2 foreach  $b \in DR_{wbc}(L, B) \cup DR_{wbc}(R, B) \cup R_{wobc}(L, B) \cup R_{wobc}(R, B)$  do
3    $l \leftarrow \text{correspondentNode}(b, L)$ ;
4    $r \leftarrow \text{correspondentNode}(b, R)$ ;
5    $m \leftarrow \text{mergeNode}(l, r, M)$ ;
6    $matches \leftarrow matches \cup (l, b, r, m)$ ;
7 end
8 return  $matches$ 

```

### Algorithm 6: Correspondent Node

**Input:**  $b, T$   
**Output:**  $b$ 's correspondent node on tree  $T$

```

1  $t \leftarrow \text{findFirst}(t \in T \rightarrow t.id = b.id)$ ;
2 if  $t = null$  then
3    $t \leftarrow \text{findFirst}(t \in T \rightarrow t.body = b.body)$ ;
4 end
5 if  $t = null$  then
6    $t \leftarrow \text{findFirst}(t \in T \rightarrow t.body \approx b.body \wedge t.id \approx b.id)$ ;
7 end
8 if  $t = null$  then
9    $t \leftarrow \text{findFirst}(t \in T \rightarrow t.body = \text{substring}(b.body) \vee b.body = \text{substring}(t.body))$ ;
10 end
11 return  $t$ ;

```

### Algorithm 7: Merge Node

**Input:**  $l, r, M$   
**Output:**  $l$  and  $r$ 's merge node on tree  $M$

```

1 if  $l \neq null$  then
2   return  $\text{find}(m \in M \rightarrow m.id = l.id)$ ;
3 end
4 if  $r \neq null$  then
5   return  $\text{find}(m \in M \rightarrow m.id = r.id)$ ;
6 end
7 return  $null$ ;

```

## 2.3 Handle Algorithms

### Algorithm 8: Check References and Merge Methods Variant

**Input:**  $(l, b, r, m), M$

```

1 if  $l.id = b.id \vee r.id = b.id$  then
2    $m.body = textualMerge(l, b, r)$ ;
3    $removeUnmatchedNode(l, r, m, M)$ ;
4 else if  $l.id \neq r.id$  then
5    $m.body = conflict(l, b, r)$ ;
6    $removeUnmatchedNode(l, r, m, M)$ ;
7 else if  $l.body \neq r.body$  then
8   if  $newReferenceTo(l) \vee newReferenceTo(r)$  then
9      $m.body = conflict(l, b, r)$ ;
10  else
11     $m.body = textualMerge(l, b, r)$ ;
12  end
13   $removeUnmatchedNode(l, r, m, M)$ ;
14 end

```

### Algorithm 9: Merge Methods Variant

**Input:**  $(l, b, r, m), M$

```

1  $m.body = textualMerge(l, b, r)$ ;
2  $removeUnmatchedNode(l, r, m, M)$ ;

```

### Algorithm 10: Check Textual and Keep Both Methods Variant

**Input:**  $(l, b, r, m), M$

```

1 if  $l.id = b.id \vee r.id = b.id$  then
2   if  $textualMergeHasConflictInvolvingSignature(b)$  then
3      $m.body = conflict(l, b, r)$ ;
4      $removeUnmatchedNode(l, r, m, M)$ ;
5   end
6 else if  $l.id \neq r.id \wedge l.body = r.body$  then
7    $m.body = conflict(l, b, r)$ ;
8    $removeUnmatchedNode(l, r, m, M)$ ;
9 end

```

### Algorithm 11: Keep Both Methods Variant

**Input:**  $(l, b, r, m), M$

```

1 if  $(l.id = b.id \vee r.id = b.id) \wedge hasConflict(m)$  then
2    $removeConflict(m)$ ;
3 end

```

### Algorithm 12: Remove Unmatched Node

**Input:**  $l, r, m, M$

```

1 if  $l.id = m.id \wedge r.id \neq m.id$  then
2    $removeNode(r, M)$ ;
3 end

```

### 3 Initialization Blocks Handler

#### 3.1 Handler Algorithm

Algorithm 13: Handle	
<b>Input:</b> L, B, R, M	
1	$A_L \leftarrow \{l \in L \mid (\neg \exists b \in B)(l.id = b.id)\};$
2	$A_R \leftarrow \{r \in R \mid (\neg \exists b \in B)(r.id = b.id)\};$
3	$D_B \leftarrow \{b \in B \mid (\neg \exists l \in L)(b.id = l.id) \wedge (\neg \exists r \in R)(b.id = r.id)\};$
4	$IB_L \leftarrow \{n \in A_L \mid n.type = INITBLOCK\};$
5	$IB_R \leftarrow \{n \in A_R \mid n.type = INITBLOCK\};$
6	$IB_B \leftarrow \{n \in D_B \mid n.type = INITBLOCK\};$
7	$matches \leftarrow \emptyset;$
8	<b>if</b> $ IB_L  = 1 \wedge  IB_B  = 1 \wedge  IB_R  = 1$ <b>then</b>
9	$matches \leftarrow matches \cup (IB_{L_1}, IB_{B_1}, IB_{R_1});$
10	<b>else</b>
11	<b>foreach</b> $b \in IB_B$ <b>do</b>
12	$l \leftarrow findFirst(l \in IB_L \rightarrow l.body \approx b.body);$
13	$r \leftarrow findFirst(r \in IB_R \rightarrow r.body \approx b.body);$
14	$IB_L \leftarrow IB_L - l;$
15	$IB_R \leftarrow IB_R - r;$
16	<b>if</b> $l \neq null \wedge r \neq null$ <b>then</b>
17	$matches \leftarrow matches \cup (l, b, r);$
18	<b>end</b>
19	<b>end</b>
20	<b>foreach</b> $l \in IB_L$ <b>do</b>
21	$r \leftarrow findFirst(r \in IB_R \rightarrow r.body \approx l.body);$
22	$IB_R \leftarrow IB_R - r;$
23	<b>if</b> $r \neq null$ <b>then</b>
24	$matches \leftarrow matches \cup (l, null, r);$
25	<b>end</b>
26	<b>end</b>
27	<b>end</b>
28	<b>foreach</b> $(l, b, r) \in matches$ <b>do</b>
29	$m \leftarrow find(m \in M \rightarrow m.body = l.body);$
30	$m.body \leftarrow textualMerge(l.body, b.body, r.body);$
31	$m \leftarrow find(m \in M \rightarrow m.body = r.body);$
32	$removeNode(m, M);$
33	<b>end</b>

## 4 Multiple Initialization Blocks Handler

### 4.1 Handler Algorithm

#### Algorithm 14: Handle

**Input:**  $L, B, R, M$

- 1  $A_L \leftarrow \{l \in L \mid (\neg \exists b \in B)(l.id = b.id)\};$
- 2  $A_R \leftarrow \{r \in R \mid (\neg \exists b \in B)(r.id = b.id)\};$
- 3  $D_B \leftarrow \{b \in B \mid (\neg \exists l \in L)(b.id = l.id) \wedge (\neg \exists r \in R)(b.id = r.id)\};$
- 4  $IB_L \leftarrow \{n \in A_L \mid n.type = INITBLOCK\};$
- 5  $IB_R \leftarrow \{n \in A_R \mid n.type = INITBLOCK\};$
- 6  $IB_B \leftarrow \{n \in D_B \mid n.type = INITBLOCK\};$
- 7  $E_L \leftarrow \text{editedNodes}(IB_L, IB_B);$
- 8  $E_R \leftarrow \text{editedNodes}(IB_R, IB_B);$

#### Algorithm 15: Edited Nodes

**Input:**  $IB, IB_B$   
**Output:** set of pairs  $(b, a)$  consisting of a deleted base node  $b$  in  $IB_B$  and its correspondent branch added node  $a$  in  $IB$

- 1  $D \leftarrow \{d \in IB_B \mid (\neg \exists a \in IB)(d.body = a.body)\};$
- 2  $A \leftarrow \{a \in IB \mid (\neg \exists d \in IB_B)(a.body = d.body)\};$
- 3  $matches \leftarrow \emptyset;$
- 4 **foreach**  $a \in A$  **do**
- 5      $S \leftarrow \{d \in D \mid a.body \approx d.body\};$
- 6      $b \leftarrow null;$
- 7     **foreach**  $s \in S$  **do**
- 8         **if**  $\text{similarity}(s.body, a.body) > \text{similarity}(b.body, a.body)$  **then**  $b \leftarrow s;$
- 9     **end**
- 10     $b \leftarrow \underset{s \in S}{\text{argmax}} a;$
- 11    **if**  $b \neq null$  **then**  $matches \leftarrow matches \cup (b, a);$
- 12 **end**
- 13 **return**  $matches$