

# 1 Semistructured Merge

## 1.1 Early Concepts

1.  $A_L$  and  $A_R$  are the sets of all the nodes added by left and right, respectively
2.  $D_L$  and  $D_R$  are the sets of all the nodes deleted by left and right, respectively

## 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  $L.index = 1$ ;
2  $B.index = 2$ ;
3  $R.index = 3$ ;
4  $LB \leftarrow mergeNodes(L, B, 1)$ ;
5  $M \leftarrow mergeNodes(LB, R, 2)$ ;
6  $D \leftarrow D_L \cap D_R$ ;
7  $removeRemainingBaseNodes(M, D)$ ;
8  $mergeMatchedContent(M)$ ;
9 return  $M$ ;
```

**Algorithm 3:** Remove Remaining Base Nodes

**Input:**  $T, D$

```

1 if  $D = \emptyset$  then return;
2 foreach  $d \in D$  do
3   if  $T = d$  then
4      $P \leftarrow T.parent$ ;
5      $P.children \leftarrow P.children - T$ ;
6     return;
7   end
8 end
9 foreach  $t \in T.children$  do
10   $\text{removeRemainingBaseNodes}(t, D)$ ;
11 end

```

**Algorithm 4:** Merge Matched Content

**Input:**  $T$

```

1 function  $\text{mergeMatchedContent}(T)$ 
2   foreach  $t \in T.children$  do
3      $\text{mergeMatchedContent}(t)$ ;
4   end
5   if  $T.children = \emptyset$  then
6     if  $SEPARATOR \in T.body$  then
7        $l, b, r \leftarrow \text{split}(T.body, SEPARATOR)$ ;
8        $l \leftarrow l - MARKER$ ;
9        $T.body \leftarrow \text{textualMerge}(l, b, r)$ ;
10    end
11  end
12 end

```

**Algorithm 5:** Merge Nodes**Input:** A, B, step**Output:** result of merging node A and node B

```

1 if  $A.type \neq B.type \vee A.id \neq B.id$  then
2   | return null;
3 end
4  $M \leftarrow A$ ;
5 if  $A.children = \emptyset \wedge B.children = \emptyset$  then
6   | if MARKER  $\in A.body$  then
7     |  $M.body \leftarrow A.body + B.body$ ;
8   | else if  $step = 1$  then
9     |  $M.body \leftarrow \text{MARKER} + A.body + \text{SEPARATOR} + B.body + \text{SEPARATOR}$ ;
10  | else if  $A.index = 1$  then
11    |  $M.body \leftarrow \text{MARKER} + A.body + \text{SEPARATOR} + \text{SEPARATOR} + B.body$ ;
12  | else
13    |  $M.body \leftarrow \text{MARKER} + \text{SEPARATOR} + A.body + \text{SEPARATOR} + B.body$ ;
14  | end
15  | return M;
16 else if  $A.children \neq \emptyset \wedge B.children \neq \emptyset$  then
17   | foreach  $b \in B.children$  do
18     |  $a \leftarrow \text{find}(a \in A.children \rightarrow a.type = b.type \wedge a.id = b.id)$ ;
19     | if  $a = \text{null}$  then
20       |  $M.children \leftarrow M.children \cup b$ ;
21       | if  $step = 1$  then  $D_L \leftarrow D_L \cup b$ ;
22       | else  $A_R \leftarrow A_R \cup b$ ;
23     | else
24       | if  $a.index = -1$  then  $a.index \leftarrow A.index$ ;
25       | if  $b.index = -1$  then  $b.index \leftarrow B.index$ ;
26       | if  $step = 1 \wedge a \in A_L$  then  $A_R \leftarrow A_R \cup b$ ;
27       |  $m \leftarrow \text{mergeNodes}(a, b, step)$ ;
28       |  $M.children \leftarrow M.children \cup m$ ;
29     | end
30   | end
31   | foreach  $a \in A.children$  do
32     |  $b \leftarrow \text{find}(b \in B.children \rightarrow a.type = b.type \wedge a.id = b.id)$ ;
33     | if  $b = \text{null}$  then
34       |  $ls \leftarrow \text{leftSiblings}(a)$ ;
35       |  $rs \leftarrow \text{rightSiblings}(a)$ ;
36       |  $M.children \leftarrow ls \cup a \cup rs$ ;
37       | if  $step = 1$  then  $A_L \leftarrow A_L \cup a$ ;
38       | else  $D_R \leftarrow D_R \cup a$ ;
39     | end
40   | end
41   | return M;
42 end
43 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

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

### 2.2 Match Algorithm

#### Algorithm 6: 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 possiblyRenamedOrDeletedBaseNodes(L, B, R) do
3    $l \leftarrow \text{getCorrespondentNode}(b, L)$ ;
4    $r \leftarrow \text{getCorrespondentNode}(b, R)$ ;
5    $m \leftarrow \text{getMergeNode}(l, r, M)$ ;
6   matches  $\leftarrow \text{matches} \cup (l, b, r, m)$ ;
7 end
8 return matches
9 function possiblyRenamedOrDeletedBaseNodes(L, B, R)
10 | return  $DR_{wbc}(L, B) \cup DR_{wbc}(R, B) \cup R_{wobc}(L, B) \cup R_{wobc}(R, B)$ ;
11 end
12 function getCorrespondentNode(b, T)
13 |  $t \leftarrow \text{findFirst}(t \in T \rightarrow t.id = b.id)$ ;
14 | if  $t = \text{null}$  then
15 |    $t \leftarrow \text{findFirst}(t \in T \rightarrow t.body = b.body)$ ;
16 | end
17 | if  $t = \text{null}$  then
18 |    $t \leftarrow \text{findFirst}(t \in T \rightarrow t.body \approx b.body \wedge (t.id.name = b.id.name \vee$ 
19 |      $t.id.params = b.id.params))$ ;
19 | end
20 | if  $t = \text{null}$  then
21 |    $t \leftarrow \text{findFirst}(t \in T \rightarrow t.body = \text{substring}(b.body) \vee b.body = \text{substring}(t.body))$ ;
22 | end
23 | return  $t$ ;
24 end
25 function getMergeNode(l, r, M)
26 | if  $l \neq \text{null}$  then
27 |   return  $\text{find}(m \in M \rightarrow m.id = l.id)$ ;
28 | end
29 | if  $r \neq \text{null}$  then
30 |   return  $\text{find}(m \in M \rightarrow m.id = r.id)$ ;
31 | end
32 | return  $\text{null}$ ;
33 end

```

## 2.3 Handle Algorithms

**Algorithm 7:** Check References and Merge Methods Variant

```

Input:  $(l, b, r, m), M$ 
1 if  $\text{singleRenamingOrDeletion}(l, b, r)$  then
2    $m.body = \text{textualMerge}(l, b, r);$ 
3    $\text{removeUnmatchedNode}(l, r, m, M);$ 
4 else if  $l.id \neq r.id$  then
5    $m.body = \text{conflict}(l, b, r);$ 
6    $\text{removeUnmatchedNode}(l, r, m, M);$ 
7 else if  $l.body \neq r.body$  then
8   if  $\text{newReferenceTo}(l) \vee \text{newReferenceTo}(r)$  then
9      $m.body = \text{conflict}(l, b, r);$ 
10     $\text{removeUnmatchedNode}(l, r, m, M);$ 
11   else
12      $m.body = \text{textualMerge}(l, b, r);$ 
13      $\text{removeUnmatchedNode}(l, r, m, M);$ 
14   end
15 end
16 function  $\text{singleRenamingOrDeletion}(l, b, r)$ 
17   return  $l.id = b.id \vee r.id = b.id;$ 
18 end
19 function  $\text{removeUnmatchedNode}(l, r, m, M)$ 
20   if  $l.id = m.id \wedge r.id \neq m.id$  then
21      $\text{removeNode}(r, M);$ 
22   end
23 end

```

**Algorithm 8:** Merge Methods Variant

```

Input:  $(l, b, r, m), M$ 
1  $m.body = \text{textualMerge}(l, b, r);$ 
2  $\text{removeUnmatchedNode}(l, r, m, M);$ 
3 function  $\text{removeUnmatchedNode}(l, r, m, M)$ 
4   if  $l.id = m.id \wedge r.id \neq m.id$  then
5      $\text{removeNode}(r, M);$ 
6   end
7 end

```

**Algorithm 9:** Check Textual and Keep Both Methods Variant

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

```

1 if singleRenamingOrDeletion( $l, b, r$ ) 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
10 function singleRenamingOrDeletion( $l, b, r$ )
11   return  $l.id = b.id \vee r.id = b.id$ ;
12 end
13 function removeUnmatchedNode( $l, r, m, M$ )
14   if  $l.id = m.id \wedge r.id \neq m.id$  then
15      $removeNode(r, M)$ ;
16   end
17 end

```

**Algorithm 10:** Keep Both Methods Variant

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

```

1 if singleRenamingOrDeletion( $l, b, r$ )  $\wedge$  hasConflict( $m$ ) then
2    $removeConflict(m)$ ;
3 end
4 function singleRenamingOrDeletion( $l, b, r$ )
5   return  $l.id = b.id \vee r.id = b.id$ ;
6 end

```