

## 0.1 Renaming Handler

### 0.1.1 Early Concepts

- Possibly renamed without body changes nodes:

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

- Possibly deleted or renamed with body changes nodes:

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

- Nodes IDs similarity:

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

### 0.1.2 Match Algorithm

#### Algorithm 1: 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 ← ∅;
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 ← matches  $\cup (l, b, r, m)$ ;
7 end
8 return matches

```

#### Algorithm 2: 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  $t = \text{null}$  then
3    $t \leftarrow \text{findFirst}(t \in T \rightarrow t.body = b.body)$ ;
4 end
5 if  $t = \text{null}$  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  $t = \text{null}$  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 3: Merge Node

**Input:** l, r, M

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

```

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

```

### 0.1.3 Handler Algorithms

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

```

#### Algorithm 5: Merge Methods Variant

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

```

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

```

#### Algorithm 6: 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      $m.body = conflict(l.body, b.body, r.body)$ ;
4      $removeUnmatchedNode(l, r, m, M)$ ;
5   end
6 else if  $l.id \neq r.id \wedge l.body = r.body$  then
7    $m.body = conflict(l.body, b.body, r.body)$ ;
8    $removeUnmatchedNode(l, r, m, M)$ ;
9 end

```

#### Algorithm 7: 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 8: 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

```