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;
```

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 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 ← ∅;
2  foreach b in possiblyRenamedOrDeletedBaseNodes(L, B, R) do
3  |  l ← getCorrespondentNode(b, L);
4  |  r ← getCorrespondentNode(b, R);
5  |  m ← getMergeNode(l, r, M);
6  |  matches ← matches ∪ (l, b, r, m);
7  end
8  return matches
9  function possiblyRenamedOrDeletedBaseNodes(L, B, R)
10 |  return DR_wbc(L, B) ∪ DR_wbc(R, B) ∪ R_wobc(L, B) ∪ R_wobc(R, B);
11 end
12 function getCorrespondentNode(b, T)
13 |  t ← findFirst(t ∈ T → t.id = b.id);
14 |  if t = null then
15 |  |  t ← findFirst(t ∈ T → t.body = b.body);
16 |  end
17 |  if t = null then
18 |  |  t ← findFirst(t ∈ T → t.body ≈ b.body ∧ ( t.id.name = b.id.name∨
   |  |    t.id.params = b.id.params));
19 |  end
20 |  if t = null then
21 |  |  t ← findFirst(t ∈ T → t.body = substring(b.body) ∨ b.body = substring(t.body));
22 |  end
23 |  return t;
24 end
25 function getMergeNode(l, r, M)
26 |  if l ≠ null then
27 |  |  return find(m ∈ M → m.id = l.id);
28 |  end
29 |  if r ≠ null then
30 |  |  return find(m ∈ M → m.id = r.id);
31 |  end
32 |  return null;
33 end
```

2.3 Handle Algorithms

Algorithm 6: Check References and Merge Methods Variant

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

```
1  if singleRenamingOrDeletion(l, b, r) then
2  |  m.body = textualMerge(l, b, r);
3  |  removeUnmatchedNode(l, r, m, M);
4  else if l.id ≠ r.id then
5  |  m.body = conflit(l, b, r);
6  |  removeUnmatchedNode(l, r, m, M);
7  else if l.body ≠ r.body then
8  |  if newReferenceTo(l) ∨ newReferenceTo(r) then
9  |  |  m.body = conflict(l, b, r);
10 |  |  removeUnmatchedNode(l, r, m, M);
11 |  else
12 |  |  m.body = textualMerge(l, b, r);
13 |  |  removeUnmatchedNode(l, r, m, M);
14 |  end
15 end
16 function singleRenamingOrDeletion(l, b, r)
17 |  return l.id = b.id ∨ r.id = b.id;
18 end
19 function removeUnmatchedNode(l, r, m, M)
20 |  if l.id = m.id ∧ r.id ≠ m.id then
21 |  |  removeNode(r, M);
22 |  end
23 end
```

Algorithm 7: Merge Methods Variant

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

```
1  m.body = textualMerge(l, b, r);
2  removeUnmatchedNode(l, r, m, M);
3  function removeUnmatchedNode(l, r, m, M)
4  |  if l.id = m.id ∧ r.id ≠ m.id then
5  |  |  removeNode(r, M);
6  |  end
7  end
```

Algorithm 8: 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 ≠ r.id ∧ 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 ∨ r.id = b.id;
12 end
13 function removeUnmatchedNode(l, r, m, M)
14 |  if l.id = m.id ∧ r.id ≠ m.id then
15 |  |  removeNode(r, M);
16 |  end
17 end
```

Algorithm 9: Keep Both Methods Variant

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

```
1  if singleRenamingOrDeletion(l, b, r) ∧ hasConflict(m) then
2  |  removeConflict(m);
3  end
4  function singleRenamingOrDeletion(l, b, r)
5  |  return l.id = b.id ∨ r.id = b.id;
6  end
```