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** $b.content = r.content \lor l.content = r.content$ **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 $D_L \leftarrow \{b \in B \mid (\neg \exists l \in L)(l.id = b.id)\}$;
7 $D_R \leftarrow \{b \in B \mid (\neg \exists r \in R)(r.id = b.id)\}$;
8 $D \leftarrow D_L \cap D_R$;
9 **foreach** $d \in D$ **do**
10 | `removeNode`(d, M);
11 **end**
12 `runTextualMergeOnLeaves`(M);
13 **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** $T.children = \emptyset \land SEPARATOR \in T.body$ **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 $M.children \leftarrow M.children \cup \texttt{mergeNodes}(a, b, step)$;
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 $l \leftarrow$ `correspondentNode(`b`, L)`;
4 $r \leftarrow$ `correspondentNode(`b`, R)`;
5 $m \leftarrow$ `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 $t \leftarrow findFirst(t \in T \rightarrow t.id = b.id)$;

2 **if** $t = null$ **then**
3 $t \leftarrow findFirst(t \in T \rightarrow t.body = b.body)$;
4 **end**

5 **if** $t = null$ **then**
6 $t \leftarrow findFirst(t \in T \rightarrow t.body \approx b.body \wedge t.id \approx b.id)$;
7 **end**

8 **if** $t = null$ **then**
9 $t \leftarrow findFirst(t \in T \rightarrow t.body = substring(b.body) \vee b.body = substring(t.body))$;
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** $find(m \in M \rightarrow m.id = l.id)$;
3 **end**

4 **if** $r \neq null$ **then**
5   **return** $find(m \in M \rightarrow m.id = r.id)$;
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** $l.id = b.id \lor r.id = b.id$ **then**
2     $m.body = textualMerge(l, b, r)$;
3   `removeUnmatchedNode(`l, r, m, M`)` ;
4 **else if** $l.id \neq r.id$ **then**
5     $m.body = conflit(l, b, r)$;
6   `removeUnmatchedNode(`l, r, m, M`)` ;
7 **else if** $l.body \neq r.body$ **then**
8   **if** `newReferenceTo(`l`)` $\lor$ `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** $l.id = b.id \lor r.id = b.id$ **then**
2   **if** `textualMergeHasConflictInvolvingSignature(`b`)` **then**
3        $m.body = conflict(l, b, r)$;
4   `removeUnmatchedNode(`l, r, m, M`)` ;
5   **end**
6 **else if** $l.id \neq r.id \land l.body = r.body$ **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** $(l.id = b.id \lor r.id = b.id) \land$ `hasConflict(`m`)` **then**
2     $removeConflict(m)$;
3 **end**

Algorithm 12: Remove Unmatched Node

Input: l, r, m, M

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