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
3. 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 ← r.content;
3  else if b.content = r.content ∨ l.content = r.content then
4  |  o.content ← l.content;
5  else
6  |  L ← fileToTree(l);
7  |  B ← fileToTree(b);
8  |  R ← fileToTree(r);
9  |  M ← mergeTrees(L, B, R);
10 |  H ← getActiveHandlers();
11 |  foreach h ∈ H do
12 |  |  h.handle(M);
13 |  end
14 |  o.content ← 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 ← mergeNodes(L, B, 1);
5  M ← mergeNodes(LB, R, 2);
6  D ← D_L ∩ D_R;
7  removeDeletedNodes(M, D);
8  updateLeafBodies(M);
9  return M;
```

Algorithm 3: Remove Deleted Nodes

Input: T, D

```
1  if D = ∅ then return;
2  foreach d ∈ D do
3  |  if T = d then
4  |  |  P ← T.parent;
5  |  |  P.children ← P.children − T;
6  |  |  return;
7  |  end
8  end
9  foreach t ∈ T.children do
10 |  removeDeletedNodes(t, D);
11 end
```

Algorithm 4: Update Leaf Bodies

Input: T

```
1  foreach t ∈ T.children do
2  |  updateLeafBodies(t);
3  end
4  if T.children = ∅ then
5  |  if SEPARATOR ∈ T.body then
6  |  |  l, b, r ← split(T.body, SEPARATOR);
7  |  |  l ← l − MARKER;
8  |  |  T.body ← textualMerge(l, b, r);
9  |  end
10 end
```

Algorithm 5: Merge Nodes

Input: A, B, step
Output: result of merging node A and node B

```
1  if A.type ≠ B.type ∨ A.id ≠ B.id then
2  |  return null;
3  end
4  M ← A;
5  if A.children = ∅ ∧ B.children = ∅ then
6  |  if MARKER ∈ A.body then
7  |  |  M.body ← A.body + B.body;
8  |  else if step = 1 then
9  |  |  M.body ← MARKER + A.body + SEPARATOR + B.body + SEPARATOR;
10 |  else if A.origin = LEFT then
11 |  |  M.body ← MARKER + A.body + SEPARATOR + SEPARATOR + B.body;
12 |  else
13 |  |  M.body ← MARKER + SEPARATOR + A.body + SEPARATOR + B.body;
14 |  end
15 |  return M;
16 else if A.children ≠ ∅ ∧ B.children ≠ ∅ then
17 |  foreach b ∈ B.children do
18 |  |  a ← find(a ∈ A.children → a.type = b.type ∧ a.id = b.id);
19 |  |  if a = null then
20 |  |  |  M.children ← M.children ∪ b;
21 |  |  |  if step = 1 then D_L ← D_L ∪ b ;
22 |  |  |  else A_R ← A_R ∪ b ;
23 |  |  else
24 |  |  |  if a.origin = UNKNOWN then a.origin ← A.origin ;
25 |  |  |  if b.origin = UNKNOWN then b.origin ← B.origin ;
26 |  |  |  if step = 1 ∧ a ∈ A_L then A_R ← A_R ∪ b ;
27 |  |  |  m ← mergeNodes(a, b, step);
28 |  |  |  M.children ← M.children ∪ m;
29 |  |  end
30 |  end
31 |  foreach a ∈ A.children do
32 |  |  b ← find(b ∈ B.children → a.type = b.type ∧ a.id = b.id);
33 |  |  if b = null then
34 |  |  |  ls ← leftSiblings(a);
35 |  |  |  rs ← rightSiblings(a);
36 |  |  |  M.children ← ls ∪ a ∪ rs;
37 |  |  |  if step = 1 then A_L ← A_L ∪ a ;
38 |  |  |  else D_R ← D_R ∪ a ;
39 |  |  end
40 |  end
41 |  return M;
42 end
43 return null;
```