1 Semistructured Merge

1.1 Early Concepts

1. **Every node's origin is set to UNKNOWN beforehand**
2. **Nodes added by left**:
   $A_L \leftarrow \{l \in L \mid (\neg\exists b \in B)(l.id = b.id)\}$
3. **Nodes added by right**:
   $A_R \leftarrow \{r \in R \mid (\neg\exists b \in B)(r.id = b.id)\}$
4. **Nodes deleted from base**:
   $D_B \leftarrow \{b \in B \mid (\neg\exists l \in L)(b.id = l.id) \wedge (\neg\exists r \in R)(b.id = r.id)\}$

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($LB$, $R$);
foreach $d \in D_B$ do
    removeNode($d$, $M$);
end
runTextualMergeOnLeaves($M$);
return $M$;
```

Algorithm 3: Run Textual Merge On Leaves

```
Input: T
foreach $t \in T.children$ do
    runTextualMergeOnLeaves($t$);
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
if $A = null$ then return $B$;
if $B = null$ then return $A$;
if $A.type \neq B.type \vee A.id \neq B.id$ then return $null$;
$M.id \leftarrow B.id$;
$M.type \leftarrow B.type$;
$M.origin \leftarrow B.origin$;
$M.children \leftarrow \emptyset$;
if $A.children = \emptyset \wedge B.children = \emptyset$ then
    if $MARKER \in A.body$ then
        $M.body \leftarrow A.body + B.body$;
    else if $A.origin = LEFT \wedge B.origin = BASE$ then
        $M.body \leftarrow MARKER + A.body + SEPARATOR + B.body + SEPARATOR$;
    else if $A.origin = LEFT$ then
        $M.body \leftarrow MARKER + A.body + SEPARATOR + SEPARATOR + B.body$;
    else
        $M.body \leftarrow MARKER + SEPARATOR + A.body + SEPARATOR + B.body$;
    end
    return $M$;
else if $A.children \neq \emptyset \wedge B.children \neq \emptyset$ then
    foreach $b \in B.children$ do
        $a \leftarrow find(a \in A.children \rightarrow a.type = b.type \wedge a.id = b.id)$;
        if $a.origin = UNKNOWN$ then $a.origin \leftarrow A.origin$;
        if $b.origin = UNKNOWN$ then $b.origin \leftarrow B.origin$;
        $M.children \leftarrow M.children \cup$ mergeNodes($a$, $b$, $step$);
    end
    foreach $a \in A.children$ do
        $b \leftarrow find(b \in B.children \rightarrow a.type = b.type \wedge a.id = b.id)$;
        if $a.origin = UNKNOWN$ then $a.origin \leftarrow A.origin$;
        if $b = null$ then $M.children \leftarrow M.children \cup a$;
    end
    return $M$;
end
return $null$;
```