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
if $l.content = b.content$ then
    $o.content \leftarrow r.content$;
else if $b.content = r.content \lor l.content = r.content$ 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($L, B, LB\text{-}STEP$);
$M \leftarrow$ mergeNodes($LB, R, LBR\text{-}STEP$);
$D_B \leftarrow D_L \cap D_R$;
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, step
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 $null$;
end
$M \leftarrow A$;
$M.origin \leftarrow B.origin$;
if $A.children = \emptyset \wedge B.children = \emptyset$ then
    if $MARKER \in A.body$ then
        $M.body \leftarrow A.body + B.body$;
    else if $step = LB\text{-}STEP$ 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 $M$;
end
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 $b.origin \leftarrow B.origin$ ;
        if $a = null$ then
            if $step = LB\text{-}STEP$ then
                $D_L \leftarrow D_L \cup b$;
            else
                $A_R \leftarrow A_R \cup b$;
            end
        else if $step = LB\text{-}STEP \wedge a \in A_L$ then
            $A_R \leftarrow A_R \cup b$;
        end
        $m \leftarrow$ mergeNodes($a$, $b$, $step$);
        $M.children \leftarrow M.children \cup m$;
    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 $b = null$ then
            $ls \leftarrow$ leftSiblings($a$);
            $rs \leftarrow$ rightSiblings($a$);
            $M.children \leftarrow ls \cup a \cup rs$;
            if $step = LB\text{-}STEP$ then $A_L \leftarrow A_L \cup a$ ;
            else if $a \notin A_L$ then $D_R \leftarrow D_R \cup a$ ;
        end
    end
    return $M$;
end
return $null$;
```