0.1 Multiple Initialization Blocks Handler

0.1.1 Handler Algorithm

Algorithm 1: Handle

Input: L, B, R, M

1 $IB_L \leftarrow \{n \in A_L \mid n.type = INITBLOCK\}$;
2 $IB_R \leftarrow \{n \in A_R \mid n.type = INITBLOCK\}$;
3 $IB_B \leftarrow \{n \in D_B \mid n.type = INITBLOCK\}$;

4 $E_L \leftarrow$ editedNodes(IB_L, IB_B);
5 $E_R \leftarrow$ editedNodes(IB_R, IB_B);

6 $DEL_L \leftarrow$ deletedNodes(IB_L, IB_B, E_L);
7 $DEL_R \leftarrow$ deletedNodes(IB_R, IB_B, E_R);

8 **foreach** $b \in IB_B$ **do**
9 $l \leftarrow E_L[b]$;
10 $r \leftarrow E_R[b]$;
11   **if** $l \neq null \wedge r \neq null$ **then**
12   updateMergeTree(l, b, r, M);
13   **else if** $l \neq null \vee r \neq null$ **then**
14   **if** $l \neq null$ **then**
15 $r \leftarrow find(r \in DEL_R \rightarrow r.body = b.body)$;
16   **if** $r \neq null$ **then** removeNode(b, M) ;
17   **else**
18 $l \leftarrow find(l \in DEL_L \rightarrow l.body = b.body)$;
19   **if** $l \neq null$ **then** removeNode(b, M) ;
20   **end**
21   updateMergeTree(l, b, r, M);
22   **else**
23 $m \leftarrow find(m \in M \rightarrow m.body = b.body)$;
24   removeNode(m, M);
25   **end**
26 **end**

27 $ADD_L \leftarrow$ addedNodes(IB_L, IB_B, E_L);
28 $ADD_R \leftarrow$ addedNodes(IB_R, IB_B, E_R);

29 $DEP \leftarrow$ dependentNodes(ADD_L, ADD_R);
30 **foreach** $(l, rs) \in DEP$ **do**
31 $s \leftarrow \varepsilon$;
32   **foreach** $r \in rs$ **do**
33 $s \leftarrow s + r.body$;
34 $m \leftarrow find(m \in M \rightarrow m.body = r.body)$;
35   removeNode(r, M);
36   **end**
37 $m \leftarrow find(m \in M \rightarrow m.body = l.body)$;
38 $m.body \leftarrow conflict(l.body, \varepsilon, s)$;
39 **end**

40 **foreach** $l \in ADD_L$ **do**
41   **foreach** $r \in ADD_R$ **do**
42   **if** $l.body = r.body$ **then**
43 $m \in find(m \in M \rightarrow m.body = r.body)$;
44   removeNode(m, M);
45   **end**
46   **end**
47 **end**

Algorithm 2: Edited Nodes

Input: IB, IB_B
Output: map associating a deleted base node b in IB_B and its correspondent added branch node a in IB

1   $D \leftarrow \{d \in IB_B \mid (\neg \exists a \in IB)(d.body = a.body)\}$;
2   $A \leftarrow \{a \in IB \mid (\neg \exists d \in IB_B)(a.body = d.body)\}$;

3   $matches \leftarrow \emptyset$;
4   **foreach** $a \in A$ **do**
5     $S \leftarrow \{d \in D \mid a.body \approx d.body\}$;
6     $b \leftarrow \underset{s \in S}{\mathrm{argmax}}\ (\texttt{similarity}(s.body,\ a.body))$;
7   **if** $b \neq null$ **then** $matches \leftarrow matches \cup \{b : a\}$;
8   **end**

9   **return** $matches$

Algorithm 3: Added Nodes

Input: IB, IB_B, E
Output: set of initilization block nodes added by branch

1   $A \leftarrow \{n \in IB \mid (\neg \exists b \in IB_B)(n.body = b.body)\}$;
2   $A \leftarrow \{n \in A \mid (\neg \exists e \in E)(n.body = e.value.body)\}$;
3   **return** A;

Algorithm 4: Deleted Nodes

Input: IB, IB_B, E
Output: set of initialization block nodes deleted by branch

1   $D \leftarrow \{b \in IB_B \mid (\neg \exists n \in IB)(b.body = n.body)\}$;
2   $D \leftarrow \{n \in D \mid (\neg \exists e \in E)(n.body = e.key.body)\}$;
3   **return** D;

Algorithm 5: Update Merge Tree

Input: l, b, r, M

1   $m \leftarrow find(m \in M \rightarrow m.body = l.body)$;
2   $m.body \leftarrow \texttt{textualMerge}(l.body,\ b.body,\ r.body)$;

3   $m \leftarrow find(m \in M \rightarrow m.body = r.body)$;
4   $\texttt{removeNode}(m,\ M)$;

Algorithm 6: Dependent Nodes

Input: ADD_L, ADD_R
Output: map associating an added left node l in ADD_L and all added right nodes r in ADD_R with common global variables

1   $DEP \leftarrow \emptyset$;
2   **foreach** $l \in ADD_L$ **do**
3     $DEP \leftarrow DEP \cup \{l : \emptyset\}$;
4     $V_L \leftarrow \texttt{globalVariables}(l)$;

5   **foreach** $r \in ADD_R$ **do**
6       $V_R \leftarrow \texttt{globalVariables}(r)$;
7   **if** $V_L \cap V_R \neq \emptyset$ **then** $DEP[l] \leftarrow DEP[l] \cup r$;
8   **end**
9   **end**

10   **return** DEP;