0.1 Multiple Initialization Blocks Handler

0.1.1 Handler Algorithm

Algorithm 1: Handle

Input: L, B, R, M

$IB_L \leftarrow \{n \in A_L \mid n.type = INITBLOCK\}$;
$IB_R \leftarrow \{n \in A_R \mid n.type = INITBLOCK\}$;
$IB_B \leftarrow \{n \in D_L \cap D_R \mid n.type = INITBLOCK\}$;

$eIB_L \leftarrow$ `editedNodes`(IB_L, IB_B);
$eIB_R \leftarrow$ `editedNodes`(IB_R, IB_B);

$dIB_L \leftarrow$ `deletedNodes`(IB_L, IB_B, eIB_L);
$dIB_R \leftarrow$ `deletedNodes`(IB_R, IB_B, eIB_R);

foreach $b \in IB_B$ **do**
    $l \leftarrow eIB_L[b]$;
    $r \leftarrow eIB_R[b]$;
    if $l \neq$ **null** $\wedge\, r \neq$ **null then**
        `updateMergeTree`(l, b, r, M);
    else if $l \neq$ **null** $\vee\, r \neq$ **null then**
        if $l \neq$ **null then**
            $r \leftarrow$ `find`$(r \in dIB_R \rightarrow r.body = b.body)$;
            if $r \neq$ **null then** `removeNode`(b, M);
        else
            $l \leftarrow$ `find`$(l \in dIB_L \rightarrow l.body = b.body)$;
            if $l \neq$ **null then** `removeNode`(b, M);
        end
        `updateMergeTree`(l, b, r, M);
    else
        $m \leftarrow$ `find`$(m \in M \rightarrow m.body = b.body)$;
        `removeNode`(m, M);
    end
end

$aIB_L \leftarrow$ `addedNodes`(IB_L, IB_B, eIB_L);
$aIB_R \leftarrow$ `addedNodes`(IB_R, IB_B, eIB_R);

$DEP \leftarrow$ `dependentNodes`(aIB_L, aIB_R);
foreach $(l, rs) \in DEP$ **do**
    $s \leftarrow \varepsilon$;
    foreach $r \in rs$ **do**
        $s \leftarrow s + r.body$;
        $m \leftarrow$ `find`$(m \in M \rightarrow m.body = r.body)$;
        `removeNode`(r, M);
    end
    $m \leftarrow$ `find`$(m \in M \rightarrow m.body = l.body)$;
    $m.body \leftarrow$ `conflict`$(l.body, \varepsilon, s)$;
end

foreach $l \in aIB_L$ **do**
    foreach $r \in aIB_R$ **do**
        if $l.body = r.body$ **then**
            $m \leftarrow$ `find`$(m \in M \rightarrow m.body = r.body)$;
            `removeNode`(m, M);
        end
    end
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

$D \leftarrow \{d \in IB_B \mid (\neg\exists a \in IB)(d.body = a.body)\}$;
$A \leftarrow \{a \in IB \mid (\neg\exists d \in IB_B)(a.body = d.body)\}$;
$matches \leftarrow \emptyset$;
foreach $a \in A$ **do**
  $S \leftarrow \{d \in D \mid a.body \approx d.body\}$;
  $b \leftarrow \underset{s \in S}{\text{argmax}}$ (`similarity(`$s.body,\ a.body$`)`);
  if $b \neq$ **null then** $matches \leftarrow matches \cup \{b : a\}$;
end
return $matches$

Algorithm 3: Added Nodes

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

$A \leftarrow \{n \in IB \mid (\neg\exists b \in IB_B)(n.body = b.body)\}$;
$A \leftarrow \{n \in A \mid (\neg\exists e \in eIB)(n.body = e.value.body)\}$;
return A;

Algorithm 4: Deleted Nodes

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

$D \leftarrow \{b \in IB_B \mid (\neg\exists n \in IB)(b.body = n.body)\}$;
$D \leftarrow \{n \in D \mid (\neg\exists e \in eIB)(n.body = e.key.body)\}$;
return D;

Algorithm 5: Update Merge Tree

Input: l, b, r, M

$m \leftarrow$ `find(`$m \in M \rightarrow m.body = l.body$`)`;
$m.body \leftarrow$ `textualMerge(`$l.body,\ b.body,\ r.body$`)`;
$m \leftarrow$ `find(`$m \in M \rightarrow m.body = r.body$`)`;
`removeNode(`$m,\ M$`)`;

Algorithm 6: Dependent Nodes

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

$DEP \leftarrow \emptyset$;
foreach $l \in aIB_L$ **do**
  $DEP \leftarrow DEP \cup \{l : \emptyset\}$;
  $V_L \leftarrow$ `globalVariables(`l`)`;
  foreach $r \in aIB_R$ **do**
    $V_R \leftarrow$ `globalVariables(`r`)`;
    if $V_L \cap V_R \neq \emptyset$ **then** $DEP[l] \leftarrow DEP[l] \cup r$;
  end
end
return DEP;