

0.1 Deletions Handler

0.1.1 Handler Algorithm

Algorithm 1: Handle

```
Input: L, B, R, M
1  $T_L \leftarrow \text{treeToText}(L);$ 
2  $T_B \leftarrow \text{treeToText}(B);$ 
3  $T_R \leftarrow \text{treeToText}(R);$ 
4 foreach  $d_l \in D_L$  do
5   if  $d_l.\text{children} \neq \emptyset$  then
6      $r \leftarrow \text{find}(r \in R \rightarrow r.\text{id} = d_l.\text{id});$ 
7      $m \leftarrow \text{find}(m \in M \rightarrow m.\text{id} = d_l.\text{id});$ 
8     if  $\text{sameShape}(d_l, r) \wedge d_l.\text{body} = r.\text{body}$  then  $\text{removeNode}(m, M);$ 
9     else if  $\text{newReference}(d_l.\text{id}, T_B, T_R)$  then  $m.\text{parent.addChild}(r, m.\text{index});$ 
10    else
11       $\text{removeNode}(m, M);$ 
12       $a_l \leftarrow \text{renamingMatch}(A_L, d_l, T_B, T_L);$ 
13      if  $a_l \neq \text{null}$  then
14         $r.\text{id} \leftarrow a_l.\text{id};$ 
15         $\text{removeNode}(a_l, M);$ 
16         $m.\text{parent.addChild}(r, m.\text{index});$ 
17      else
18         $n.\text{id} \leftarrow r.\text{id};$ 
19         $n.\text{type} \leftarrow r.\text{type};$ 
20         $n.\text{body} \leftarrow \text{conflict}(\varepsilon, d_l, r);$ 
21         $m.\text{parent.addChild}(n, m.\text{index});$ 
22      end
23    end
24  end
25 end
26 foreach  $d_r \in D_R$  do
27   if  $d_r.\text{children} \neq \emptyset$  then
28      $l \leftarrow \text{find}(l \in L \rightarrow l.\text{id} = d_r.\text{id});$ 
29      $m \leftarrow \text{find}(m \in M \rightarrow m.\text{id} = d_r.\text{id});$ 
30     if  $\text{sameShape}(d_r, l) \wedge d_r.\text{body} = l.\text{body}$  then  $\text{removeNode}(m, M);$ 
31     else if  $\text{newReference}(d_r.\text{id}, T_B, T_L)$  then  $m.\text{parent.addChild}(l, m.\text{index});$ 
32     else
33        $\text{removeNode}(m, M);$ 
34        $a_r \leftarrow \text{renamingMatch}(A_R, d_r, T_B, T_R);$ 
35       if  $a_r \neq \text{null}$  then
36          $l.\text{id} \leftarrow a_r.\text{id};$ 
37          $\text{removeNode}(a_r, M);$ 
38          $m.\text{parent.addChild}(l, m.\text{index});$ 
39       else
40          $n.\text{id} \leftarrow l.\text{id};$ 
41          $n.\text{type} \leftarrow l.\text{type};$ 
42          $n.\text{body} \leftarrow \text{conflict}(l, d_r, \varepsilon);$ 
43          $m.\text{parent.addChild}(n, m.\text{index});$ 
44       end
45     end
46   end
47 end
```

Algorithm 2: Same Shape**Input:** A, B **Output:** whether nodes A and B have same shape

```

1 if  $A.children = \emptyset \wedge B.children = \emptyset$  then return  $A.type = B.type$ ;
2 if  $A.children = \emptyset \vee B.children = \emptyset$  then return  $FALSE$ ;
3 if  $|A.children| \neq |B.children|$  then return  $FALSE$ ;
4  $result \leftarrow TRUE$ ;
5 foreach  $(a, b) \in (A.children, B.children)$  do
6    $result \leftarrow result \wedge sameShape(a, b)$ ;
7 end
8 return  $result$ ;

```

Algorithm 3: New Reference**Input:** id, T_B, T **Output:** whether there is a new reference to id in T

```

1 return  $countReferences(id, T) > countReferences(id, T_B)$ ;

```

Algorithm 4: Renaming Match**Input:** A, d, T_B, T **Output:** added node a in A with the same shape and similar body as deleted node d , such that there are no new references to a 's id in T

```

1 return  $find(a \in A \rightarrow sameShape(a, d) \wedge a.body \approx d.body \wedge \neg newReference(a.id, T_B, T))$ ;

```