|  |  |
| --- | --- |
| Input: TW, artifacts set A, starting node of TW p | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58 | set CNA1, CNA2, CNA3, CNA4 are empty;  queue is empty;  put p into queue;  while (queue is not null) {  p = dequeue();  switch (p) {  case (starting node)  enqueue(p’s immediate successor);  p.K = all artifacts inside A;  break;  case (ending node):  break; *//Reaching end of TW*  case (activity node):  enqueue(p’s immediate successor);  let p.K = p’.K; p.W = p’.W; where p’ is the immediate predecessor of p;  let set S be the set of elements operated in p;  repeat {  artifact x = remove one element in S;  if (x has more than one operation in p)  o = x’s first operation in p;  switch (o) {  case (R):  if (x has a corresponding artifact q in K) {  add (x, q, p) into CNA1;  }  break;  case (K):  if (x has a corresponding artifact q in K) {  add (x, q, p) into CNA2;  }  break;  case (W):  if (x has a corresponding artifact q in K) {  add (x, q, p) into CNA3;  } else if (p has a corresponding artifact q in W){  add (x, q, p) into CNA4;  }  break;  } *//end switch*  p.K = p.K – {x}; p.W = p.W –{x};  if (o is last operation)  if (o is K or W) add (x, p) into p.K or p.W corresponding to o;  } until S is empty;  break;  case (split node):  enqueue(all p’s immediate successors);  break;  p.K = p’.K; p.W = p’.W; where p’ is the immediate predecessor of p;  case (join node):  if (p’s predecessors are all handled) {  enqueue(p’s immediate successor);  p.K = p’1.K∪…p’N.K;  p.W = p’1.W∪…p’N.K; where p’1…p’N are p’s immediate predecessors, N>1;  }  break;  } *//end switch*  } *//end while*  Return CNA1, CNA2, CNA3, CNA4; |

The input of this Algorithm is workflow TW, starting node of TW p, and artifacts set A. We assume that each artifact have 3 states: Read, Write, and Kill. Each node p will carry 2 sets, K, and W, which contains the record of kill, or write artifact sets for the specific node. These 2 set will be updated at each iteration of the Algorithm. A queue is used to traverse each node of TW.

If

In an activity node, continuous anomalies occur in 4 situations, we divide them into sets CNA1~CNA4, whose elements contain the tuples (x, q, p), x represent a variable, q and p represent the nodes:

CNA1: the set of elements which attempt to read a killed artifact.

CNA2: the set of elements which attempt to kill a killed artifact.

CNA3: the set of elements which attempt to write a killed artifact.

CNA4: the set of elements which attempt to write an artifact that is written before.

At each case, we check an artifact if corresponding artifact exist in previous node, and update the K, W sets. If an anomaly is found, the artifact, and its corresponding node pairs will be added into CAN. At the end of Algorithm, CNA1~CNA4 are returned.

The time complexity of this Algorithm will be O(n) (n is number of nodes in TW), while the computation time of each node is static, and we traverse each node once with queue.