

Automated Truncation of Differential Trails and Trail Clustering in ARX (Pseudocode)

Alex Biryukov¹, Luan Cardoso dos Santos¹, Daniel Feher¹,
Vesselin Velichkov², and Giuseppe Vitto¹

¹ SnT, University of Luxembourg
{alex.biryukov,luan.cardoso}@uni.lu
² University of Edinburgh
vvelichk@ed.ac.uk

Abstract. Accompanying pseudocode to publication “Automated Truncation of Differential Trails and Trail Clustering in ARX” from SAC’21 (ePrint Report 2021/1194), Sections 4 and 5.

Algorithm 1 Truncation of Differential Trails in ARX

Input:

i : $0 \leq i < n$: bit position; j : $1 \leq j < R$: round position
 τ : non-truncated trail on R rounds, where τ_i^j is the i -th bit at round j

Output:

$\{\tau\}$: all truncations of τ that follow Rule 1, Rule 2 and Rule 3

```
1: procedure truncate_trail
2: if  $j < R$  then
3:   for truncate = true, false do
4:     // truncate bit  $\tau_i^j$ 
5:     if truncate = true and  $\tau_i^j \neq *$  then
6:       Truncate bit  $\tau_i^j \leftarrow *$  and propagate to rounds  $j + 1, \dots, R - 1$ 
7:       if Rules 1,2 and 3 are not violated for any round then
8:         Update  $\tau$  with  $\tau^j, \tau^{j+1} \dots \tau^{R-1}$ 
9:         Call truncate_trail for next bit  $i + 1$  or next round  $j + 1$ 
10:      // do not truncate  $\tau_i^j$ : move to next bit
11:      if truncate = false then
12:        Call truncate_trail for next bit  $i + 1$  or next round  $j + 1$ 
13: else
14:   // Last round: return a truncated version of  $\tau$ 
15:   return  $\tau$ 
```

Algorithm 2 Absorb a new TD trail τ into existing set of trails \mathfrak{T}

Input: \mathfrak{T} : set of disjoint TD trails; τ : new TD trail (possibly $\tau \in \mathfrak{T}$)**Output:** \mathfrak{T}' : updated set of disjoint TD trails that contains all new (non-truncated) trails from τ (possibly $\mathfrak{T} = \mathfrak{T}'$)

```

1: procedure tdiff_absorb_new_trail( $\mathfrak{T}, \tau$ )
2: // initialize a set  $\mathbf{t}$  of TD trails with the input trail  $\tau$ 
3:  $\mathbf{t} \leftarrow \emptyset$ ; add  $\tau$  to  $\mathbf{t}$ 
4: for all  $\mathbf{T} \in \mathfrak{T}$  do
5:   if  $\mathbf{t} = \emptyset$  then
6:     // all trails in  $\mathbf{t}$  have been fully absorbed; return
7:     return  $\mathfrak{T}$ 
8:   // absorb  $\mathbf{t}$  into  $\mathbf{T}$  and store the remainder in  $\mathbf{t}'$ 
9:    $\mathbf{t}' \leftarrow \emptyset$ 
10:  for all  $\tau \in \mathbf{t}$  do
11:    // if  $\tau$  contains trails not already in  $\mathbf{T}$ , then split  $\tau$  into TD trail subsets
12:    to exclude duplicates using
13:    if  $(\mathbf{T} \subset \tau) \vee (\mathbf{T}, \tau : \text{PO})$  then
14:       $\mathbf{t}_{\text{temp}} \leftarrow \text{tdiff\_madd\_trails\_make\_disjoint}(\mathbf{T}, \tau)$ 
15:      add  $\mathbf{t}_{\text{temp}}$  to  $\mathbf{t}'$ 
16:    // if  $\tau, \mathbf{T}$ : disjoint, then all trails in  $\tau$  are new, so add it
17:    if  $(\tau, \mathbf{T})$ : disjoint then
18:      add  $\tau$  to  $\mathbf{t}'$ 
19:    // if  $\tau$  is a subset of  $\mathbf{T} \implies$  it contains no new trails, so do nothing
20:    if  $(\tau \subset \mathbf{T})$  then
21:      continue
22:    // overwrite  $\mathbf{t}$  with the part of it that was not absorbed i.e.  $\mathbf{t}'$ 
23:     $\mathbf{t} \leftarrow \mathbf{t}'$ 
24: //  $\mathbf{t}$  contains all trails not absorbed in  $\mathfrak{T}$  – add them to  $\mathfrak{T}$  and return
25:  $\mathfrak{T}' \leftarrow \mathfrak{T} \cup \mathbf{t}$ 
26: return  $\mathfrak{T}'$ 

```

Algorithm 3 The bitwise conditional truncated differential probability of ADD**Input:**

$(\alpha\beta\gamma)_i, (\alpha\beta\gamma)_{i-1}$: the values of the truncated differential $(\alpha\beta\gamma)$ at bit positions i and $(i-1)$ for $1 \leq i < n$

Output:

$p_{i-1} = \Pr((\alpha\beta\gamma)_i \mid (\alpha\beta\gamma)_{i-1})$ or -1 (invalid input) if the inputs do not comply with Rules 1, 2 and 3

```

1: procedure xdp_dset_add_i( $i, (\alpha\beta\gamma)_i, (\alpha\beta\gamma)_{i-1}$ )
2: if  $(\alpha\beta\gamma)_i, (\alpha\beta\gamma)_{i-1}$  contradict Rules 1, 2, 3 then
3:   return  $-1$ 
4:  $p_{i-1} \leftarrow 1$  // initialize Pr to 1
5: if  $1 \leq i \leq n-1$  then
6:   if  $(i-1) = 0$  then
7:     // By Rule 4:  $(\alpha_0 \neq *) \wedge (\beta_0 \neq *) \wedge (\gamma_0 \neq *)$ 
8:     if  $\alpha_0 \oplus \beta_0 \neq \gamma_0$  then  $p_{i-1} \leftarrow 0$  else  $p_{i-1} \leftarrow 1$  // Theorem 1
9:   // non-truncated case
10:  if  $(\alpha_{i-1} \neq *) \wedge (\beta_{i-1} \neq *) \wedge (\gamma_{i-1} \neq *)$  then
11:    if  $\alpha_{i-1} = \beta_{i-1} = \gamma_{i-1}$  then
12:      // By Rule 2:  $(\alpha_i \neq *) \wedge (\beta_i \neq *) \wedge (\gamma_i \neq *)$ 
13:      if  $\alpha_i \oplus \beta_i \oplus \gamma_i \neq \alpha_{i-1}$  then  $p_{i-1} \leftarrow 0$  else  $p_{i-1} \leftarrow 1$  // Theorem 1
14:    else
15:       $p_{i-1} \leftarrow 1/2$  // Theorem 1 for  $\neg(\alpha_{i-1} = \beta_{i-1} = \gamma_{i-1})$ 
16:  // truncated case
17:  if  $(\alpha_{i-1} = *) \vee (\beta_{i-1} = *) \vee (\gamma_{i-1} = *)$  then
18:    // w.l.o.g. assume  $(\alpha_{i-1} = *)$ : by Rule 1  $\implies (\beta_{i-1} \neq *) \wedge (\gamma_{i-1} \neq *)$ 
19:    if  $\beta_{i-1} = \gamma_{i-1}$  then
20:      // By Rule 2:  $(\alpha_i \neq *) \wedge (\beta_i \neq *) \wedge (\gamma_i \neq *)$ 
21:      if  $\alpha_i \oplus \beta_i \oplus \gamma_i = \beta_{i-1}$  then
22:        //  $(\alpha_{i-1} = *) \implies$  by Theorem 1 for  $\alpha_{i-1} = \beta_{i-1} \implies p_{i-1} = 1$ 
        // and for  $\alpha_{i-1} \neq \beta_{i-1} \implies p_{i-1} = 1/2$ , so in total  $p_{i-1} = 1 + 1/2 = 3/2$ 
23:         $p_{i-1} \leftarrow 3/2$ 
24:      else
25:         $p_{i-1} \leftarrow 0$  // Theorem 1
26:    if  $\beta_{i-1} \neq \gamma_{i-1}$  then
27:      //  $\neg(\alpha_{i-1} = \beta_{i-1} = \gamma_{i-1}) : \forall \alpha_{i-1}$ , so by Theorem 1 for  $\alpha_{i-1} = 0$  :  $p_{i-1} = 1/2$  and for  $\alpha_{i-1} = 1$  :  $p_{i-1} = 1/2$ , so in total  $p_{i-1} = 1/2 + 1/2 = 1$ 
28:       $p_{i-1} \leftarrow 1$ 
29:  // MSB
30: if  $i = n$  then
31:  // w.l.o.g. let  $(\alpha_{n-1} = *)$ : by Theorem 1 the Pr at the MSB is 1, so for  $\alpha_{n-1} = 0$  :  $p_{n-1} = 1$  and for  $\alpha_{n-1} = 1$  :  $p_{n-1} = 1$ , so in total  $p_{n-1} = 2$ 
32:  if  $(\alpha_{n-1} = *) \vee (\beta_{n-1} = *) \vee (\gamma_{n-1} = *)$  then
33:     $p_{n-1} \leftarrow 2$ 
34: return  $p_{i-1}$ 

```

Algorithm 4 Differential Probability of SPECK DS Trail: Processing a Beta Chain

Input:

$\beta_{i'}^{j'} = *$: start of a beta chain – the value of β in an R -round SPECK trail at round j' and bit position i' that is $*$: $0 \leq j' \leq R-1$, $0 \leq i' \leq (n-1)$; $l \in \{2, 3\}$: left bit-rotation constant for SPECK; $r \in \{7, 8\}$: right bit-rotation constant for SPECK; V : array of visited bits in the beta chain indexed by j : round; i : bit position as V_{ji}

Output:

$p_{i'}^{j'}, \bar{p}_{i'}^{j'}$: Pr of the beta chain for the cases resp. $\beta_{i'}^{j'} = 0$ and $\beta_{i'}^{j'} = 1$; updated V ;

```

1: procedure speck_trail_beta_chain( $\beta_{i'}^{j'} = *, V$ )
2:  $\beta_{\text{prev}} \leftarrow \emptyset; \bar{\beta}_{\text{prev}} \leftarrow \emptyset$  // temporary variables to store  $\beta_{i-l}^{j-1}$ 
3:  $i \leftarrow i', p_{i'}^{j'} \leftarrow 1, \bar{p}_{i'}^{j'} \leftarrow 1$  // initialize the beta chain Pr and the start bit  $i$ 
4: for  $j = j' \dots R-1$  do
5:   if  $j = j'$  then
6:     // start of a beta chain: initialize the two possibilities for  $\beta_i^j = *$ 
7:      $\beta_i^j \leftarrow 0; \bar{\beta}_i^j \leftarrow 1$ 
8:   if  $j > j'$  then
9:     // middle of beta chain: compute  $\beta_i^j$  from previous rounds; by the
        // SPECK round function:  $\beta_i^j = \gamma_i^{j-1} \oplus \beta_{i-l}^{j-1}$ ; note that  $\beta_i^j = \beta_{i-l}^{j-1} = *$  as
        // they are part of a beta chain; then by the propagation conditions follows
        // that  $(\gamma_i^{j-1} \neq *)$ ; the value of  $\beta_{i-l}^{j-1}$  is stored in  $\beta_{\text{prev}}$ 
10:     $\beta_i^j \leftarrow \gamma_i^{j-1} \oplus \beta_{\text{prev}}; \bar{\beta}_i^j \leftarrow \gamma_i^{j-1} \oplus \bar{\beta}_{\text{prev}}$ 
11:    // the conditional Pr of bit  $i+1$ , round  $j$  for  $\beta_i^j$  and  $\bar{\beta}_i^j$  (Alg. (3))
12:     $p_i^j \leftarrow \mathbf{x dp\_dset\_add\_i}(i+1, (A, B, \Gamma)_{i+1}^j, (\alpha, \beta, \gamma)_i^j)$ 
13:     $\bar{p}_i^j \leftarrow \mathbf{x dp\_dset\_add\_i}(i+1, (A, B, \Gamma)_{i+1}^j, (\alpha, \bar{\beta}, \gamma)_i^j)$ 
14:    // accumulate the Pr to the total Pr of the beta chain
15:     $p_{i'}^{j'} \leftarrow (p_{i'}^{j'} \cdot p_i^j); \bar{p}_{i'}^{j'} \leftarrow (\bar{p}_{i'}^{j'} \cdot \bar{p}_i^j)$ 
16:     $\beta_{\text{prev}} \leftarrow \beta_i^j; \bar{\beta}_{\text{prev}} \leftarrow \bar{\beta}_i^j$  // store the values of  $\beta_i^j = *$ 
17:     $V_{ji} \leftarrow \mathbf{true}$  // mark the bit of the beta chain as visited
18:    // update the next bit position of the beta chain at the next round  $(j+1)$ : it
        // is  $i$  rotated by  $l$  to the left due to the left rotation in SPECK
19:     $i \leftarrow (i + l) \bmod n$ 
20: return  $p_{i'}^{j'}, \bar{p}_{i'}^{j'}$ 

```

Algorithm 5 Differential Probability of SPECK DS Trail: Processing a Gamma Chain

Input:

$\gamma_{i'}^{j'} = *$: start of a gamma chain – the value of γ in an R -round SPECK trail at round j' and bit position i' that is $*$: $0 \leq j' \leq R-1$, $0 \leq i' \leq (n-1)$; $l \in \{2, 3\}$: left bit-rotation constant for SPECK; $r \in \{7, 8\}$: right bit-rotation constant for SPECK ; V : array of visited bits in the beta chain indexed by j : round; i : bit position as V_{ji})

Output:

$p_{i'}^{j'}, \bar{p}_{i'}^{j'}$: Pr of the gamma chain for the cases resp. $\gamma_{i'}^{j'} = 0$ and $\gamma_{i'}^{j'} = 1$; updated V ;

- 1: **procedure** `speck_trail_gamma_chain`($\gamma_{i'}^{j'} = *, V$)
- 2: $i \leftarrow i', j \leftarrow j', p_{i'}^{j'} \leftarrow 1, \bar{p}_{i'}^{j'} \leftarrow 1$ // initialize the gamma chain Pr and the start j, i
- 3: // start of a gamma chain: initialize the two possibilities for $\gamma_i^j = *$
- 4: $\gamma_i^j \leftarrow 0; \bar{\gamma}_i^j \leftarrow 1$
- 5: // the conditional Pr of bit $i+1$, round j for γ_i^j and $\bar{\gamma}_i^j$ (Alg. (3))
- 6: $p_i^j \leftarrow \text{xdp_dset_add_i}(i+1, (A, B, \Gamma)_{i+1}^j, (\alpha, \beta, \gamma)_i^j)$
- 7: $\bar{p}_i^j \leftarrow \text{xdp_dset_add_i}(i+1, (A, B, \Gamma)_{i+1}^j, (\alpha, \beta, \bar{\gamma})_i^j)$
- 8: // accumulate the Pr at (j, i) to the total Pr of the gamma chain
- 9: $p_{i'}^{j'} \leftarrow (p_{i'}^{j'} \cdot p_i^j); \bar{p}_{i'}^{j'} \leftarrow (\bar{p}_{i'}^{j'} \cdot \bar{p}_i^j)$
- 10: $V_{ji} \leftarrow \text{true}$ // mark the bit of the gamma chain as visited
- 11: // $\gamma_i^j = \alpha_{i-r}^{j+1}$ through the right rotation by r ; process the two possibilities for α_{i-r}^{j+1}
- 12: $p_{i-r}^{j+1} \leftarrow \text{xdp_dset_add_i}(i-r+1, (A, B, \Gamma)_{i-r+1}^{j+1}, (\alpha, \beta, \gamma)_{i-r}^{j+1})$
- 13: $\bar{p}_{i-r}^{j+1} \leftarrow \text{xdp_dset_add_i}(i-r+1, (A, B, \Gamma)_{i-r+1}^{j+1}, (\alpha, \beta, \bar{\gamma})_{i-r}^{j+1})$
- 14: // accumulate the Pr at $(j+1, i-r)$ to the total Pr of the gamma chain
- 15: $p_{i'}^{j'} \leftarrow (p_{i'}^{j'} \cdot p_{i-r}^{j+1}); \bar{p}_{i'}^{j'} \leftarrow (\bar{p}_{i'}^{j'} \cdot \bar{p}_{i-r}^{j+1})$
- 16: $V_{(j+1)(i-r)} \leftarrow \text{true}$ // mark the bit of the gamma chain as visited
- 17: // $\gamma_i^j = *$ further affects $\beta_i^{j+1} = *$ through $\beta_i^{j+1} = \gamma_i^j \oplus \beta_{i-3}^j$; note that $\gamma_i^j = *$ $\implies \beta_{i-3}^j \neq *$ due to the propagation conditions for SPECK64; $\beta_i^{j+1} = *$ gives rise to a beta chain, which is processed with Alg. (4)
- 18: $p_{i'}^{j'}, \bar{p}_{i'}^{j'} \leftarrow \text{speck_trail_beta_chain}(\beta_i^{j+1} = *, V)$
- 19: **return** $p_{i'}^{j'}, \bar{p}_{i'}^{j'}$

Algorithm 6 Differential Probability of SPECK DS Trail

Input:**T**: DS SPECK trail on R rounds with n -bit words**Output:** p : the differential probability of **T** (the sum of the Pr of all non-truncated trails generated by **T**)

```

1: procedure xdp_dset_speck_trail(T)
2:   // Initialize array of visited bits in T indexed by  $j$ : round;  $i$ : bit position
3:    $\forall j, i: 0 \leq j \leq R-1, 0 \leq i \leq (n-1): V_{ji} \leftarrow \text{false}$ 
4:    $p \leftarrow 1$  // initialize Pr of trail T
5:   for  $j = 0 \dots R-1$  do
6:     for  $i = 0 \dots n-1$  do
7:       // start of gamma chain – process with Alg. (5)
8:       if  $(\gamma_i^j = *) \wedge (V_{ji} = \text{false})$  then
9:          $(p_i^j, \bar{p}_i^j) \leftarrow \text{speck\_trail\_gamma\_chain}(\gamma_i^j, V)$ 
10:         $p \leftarrow p (p_i^j + \bar{p}_i^j)$  // accumulate Pr
11:       // start of beta chain – process with Alg. (4)
12:       if  $(\beta_i^j = *) \wedge (V_{ji} = \text{false})$  then
13:          $(p_i^j, \bar{p}_i^j) \leftarrow \text{speck\_trail\_beta\_chain}(\beta_i^j, V)$ 
14:          $p \leftarrow p (p_i^j + \bar{p}_i^j)$  // accumulate Pr
15:       // if there are no dependency chains – compute the conditional Pr of the
       // single bit position using Alg. (3)
16:       if  $(\beta_i^j \neq *) \wedge (\gamma_i^j \neq *) \wedge (V_{ji} = \text{false})$  then
17:          $p_i^j \leftarrow \text{xdp\_dset\_add\_i}(i+1, (A, B, \Gamma)_{i+1}^j, (\alpha, \beta, \gamma)_i^j)$ 
18:          $V_{ji} \leftarrow \text{true}$  // mark the bit as visited
19:          $p \leftarrow p p_i^j$  // accumulate Pr
20:   return  $p$ 

```

Algorithm 7 Generate all differences in TD a that are not already in TD A

Input:

A, a : non-disjoint distinct TD i.e. $(a \subset A) \vee (A \subset a) \vee (A, a : \text{PO})$ (partially overlapping); A, a collectively generate set of differences \mathcal{D}

Output:

$\{a\}$: set of TD s.t. $\forall i, j : a_i, a_j \in \{a\}$: disjoint; $\forall i : a_i \in \{a\} : a_i, A$: disjoint; $A, \{a\}$ collectively generate \mathcal{D} with any duplicates removed

e : TD representing the differences generated by a that are also in A (i.e. the set of redundant differences in a with respect to A)

```

1: procedure tdiff_make_disjoint( $A, a$ )
2: if  $(A \subset a) \vee (A, a : \text{PO})$  then
3:    $m \leftarrow$  mask for the bits that are  $*$  in  $a$  and  $\cdot$  in  $A$ 
4:   // set the bits that are  $*$  in  $a$  and  $\cdot$  in  $A$  to the values in  $A$ 
5:   // the result TD  $e$  is the set of differences generated by  $a$  that are already in
    $A$ 
6:    $e \leftarrow a \text{ OR } (A \text{ AND } m)$ 
7:   // Split  $a$  into subsets that exclude the overlapping TD  $e$ 
8:    $\{a\} \leftarrow \emptyset$ 
9:   for all bit positions  $i$  where  $a_i = *$  do
10:     $a' \leftarrow a$ 
11:    // set all stars up to the  $(i - 2)$ -nd star to the value in  $e$ 
12:    set  $(a'_0, \dots, a'_{i-2})$  to  $(e_0, \dots, e_{i-2})$ 
13:    // set the  $(i-1)$ -st star to the negated value of  $e_{i-1}$  to ensure  $a', e$ : disjoint
14:     $a'_{i-1} \leftarrow 1 \oplus e_{i-1}$ 
15:    // leave all stars at positions  $\geq i$  unchanged i.e. as in  $a$ 
16:    add  $a'$  to  $\{a\}$ 
17: else
18:   //  $(a \subset A) \implies$  all diffs. in  $a$  are already in  $A$ 
19:    $e \leftarrow a$ ;  $\{a\} \leftarrow \emptyset$ 
20: return  $\{a\}, e$ 

```

Algorithm 8 Generate all ADD TD differentials in (a, b, c) that are not already in (A, B, Γ)

Input:

$(A, B, \Gamma), (a, b, c)$: non-disjoint distinct ADD TD differentials i.e. $((a, b, c) \subset (A, B, \Gamma)) \vee ((A, B, \Gamma) \subset (a, b, c)) \vee ((A, B, \Gamma), (a, b, c) : \text{PO})$ (partially overlapping); $(A, B, \Gamma), (a, b, c)$ collectively generate set of ADD differentials \mathcal{D} (possibly with duplicates)

Output:

$\{(a, b, c)\}$: set of ADD TD differentials s.t. $\forall i, j : (a, b, c)_i, (a, b, c)_j \in \{(a, b, c)\}$: disjoint; $\forall i : (a, b, c)_i \in \{(a, b, c)\} : (a, b, c)_i, (A, B, \Gamma)$: disjoint; $(A, B, \Gamma), \{(a, b, c)\}$ collectively generate the set \mathcal{D} with any duplicates removed
 (e_A, e_B, e_Γ) : ADD TD differential representing the differences generated by (a, b, c) that are also in (A, B, Γ) (i.e. the set of redundant differences in (a, b, c) with respect to (A, B, Γ))

```

1: procedure tdiff_madd_make_disjoint $((A, B, \Gamma), (a, b, c))$ 
2:  $\{(a, b, c)\} \leftarrow \emptyset$ 
3: // Split  $a$  into  $\{a\}$  s.t.  $\{a\}, A$ : disjoint,  $a \cap A = e_A$  (Alg. (7))
4:  $(\{a\}, e_A) \leftarrow \text{tdiff\_make\_disjoint}(A, a)$ 
5: if  $(A \subset a) \vee (A, a : \text{PO})$  then
6:   // for each TD  $a_i \in \{a\}$  generate a new ADD TD differential  $(a_i, b, c)$ : as
    $a_i, A$ : disjoint  $\implies (a_i, b, c), (A, B, \Gamma)$ : disjoint
7:   for  $\forall a_i \in \{a\}$  do
8:     add  $(a_i, b, c)$  to  $\{(a, b, c)\}$ 
9: else
10:  //  $(a \subset A) \implies e_A \leftarrow a; \{a\} \leftarrow \emptyset$ : do nothing
11:  continue
12: // Split  $b$  into  $\{b\}$  s.t.  $\{b\}, B$ : disjoint,  $b \cap B = e_B$  (Alg. (7))
13:  $(\{b\}, e_B) \leftarrow \text{tdiff\_make\_disjoint}(B, b)$ 
14: if  $(B \subset b) \vee (B, b : \text{PO})$  then
15:  // for each TD  $b_i \in \{b\}$  generate a new ADD TD differential  $(e_A, b_i, c)$ : as
   $b_i, B$ : disjoint  $\implies (e_A, b_i, c), (A, B, \Gamma)$ : disjoint, where  $e_A = a \cap A$ 
16:  for  $\forall b_i \in \{b\}$  do
17:    add  $(e_A, b_i, c)$  to  $\{(a, b, c)\}$ 
18: else
19:  //  $(b \subset B) \implies e_B \leftarrow b; \{b\} \leftarrow \emptyset$ : do nothing
20:  continue
21: // Split  $c$  into  $\{c\}$  s.t.  $\{c\}, \Gamma$ : disjoint,  $c \cap \Gamma = e_\Gamma$  (Alg. (7))
22:  $(\{c\}, e_\Gamma) \leftarrow \text{tdiff\_make\_disjoint}(\Gamma, c)$ 
23: if  $(\Gamma \subset c) \vee (\Gamma, c : \text{PO})$  then
24:  // for each TD  $c_i \in \{c\}$  generate a new ADD TD differential  $(e_A, e_B, c_i)$ : as
   $c_i, \Gamma$ : disjoint  $\implies (e_A, e_B, c_i), (A, B, \Gamma)$ : disjoint, where  $e_A = a \cap A, e_B = b \cap B$ 
25:  for  $\forall c_i \in \{c\}$  do
26:    add  $(e_A, e_B, c_i)$  to  $\{(a, b, c)\}$ 
27: else
28:  //  $(c \subset \Gamma) \implies e_\Gamma \leftarrow c; \{c\} \leftarrow \emptyset$ : do nothing
29:  continue
30: return  $\{(a, b, c)\}, (e_A, e_B, e_\Gamma)$ 

```

Algorithm 9 Generate all TD trails in τ that are not already in \mathbf{T}

Input:

\mathbf{T}, τ : non-disjoint distinct TD trails: $\mathbf{T} = (A, B, \Gamma)_0, (A, B, \Gamma)_1 \dots, (A, B, \Gamma)_{n-1}$,
 $\tau = (a, b, c)_0, (a, b, c)_1 \dots, (a, b, c)_{n-1}$; \mathbf{T}, τ collectively generate set of TD trails \mathcal{D}
 (possibly with duplicates)

Output:

$\{\tau\}$: set of TD trails s.t. $\forall i, j : \tau_i, \tau_j \in \{\tau\}$: disjoint; $\forall i : \tau_i \in \{\tau\} : \tau_i, \mathbf{T}$: disjoint;
 $\mathbf{T}, \{\tau\}$ collectively generate the set \mathcal{D} with any duplicates removed

- 1: **procedure** `tdiff_madd_trails_make_disjoint`(\mathbf{T}, τ)
- 2: *// initialize a temporary (running) trail τ' to τ*
- 3: $\{\tau\} \leftarrow \emptyset; \tau' \leftarrow \tau$
- 4: **for** $i = 0 \dots (n - 1)$ **do**
- 5: *// get the i -th ADD TD differentials of the trails τ, \mathbf{T}*
- 6: $(a, b, c)_i \leftarrow \tau_i; (A, B, \Gamma)_i \leftarrow \mathbf{T}_i$
- 7: *// if $(a, b, c)_i$ contains differentials $(\alpha\beta\gamma)_i$ that are not already in $(A, B, \Gamma)_i$*
 then split $(a, b, c)_i$ into subsets to remove the overlap using Alg. (8)
- 8: **if** $((A, B, \Gamma)_i \subset (a, b, c)_i) \vee ((A, B, \Gamma)_i, (a, b, c)_i : \text{PO}) \wedge ((A, B, \Gamma)_i \neq (a, b, c)_i)$
 then
- 9: $\{(a, b, c)_i\}, (e_A, e_B, e_\Gamma)_i \leftarrow \text{tdiff_madd_make_disjoint}((A, B, \Gamma)_i, (a, b, c)_i)$
- 10: *// overwrite the i -th ADD TD transition of the original trail τ with each*
 element of $\{(a, b, c)_i\}$: since $\{(a, b, c)_i\}, (A, B, \Gamma)_i$: disjoint $\implies \tau', \mathbf{T}$: dis-
 joint
- 11: **for each** $(a, b, c)_i \in \{(a, b, c)_i\}$ **do**
- 12: $\tau'_i \leftarrow (a, b, c)_i$; add τ' to $\{\tau\}$
- 13: **else**
- 14: *// $(a, b, c)_i \subset (A, B, \Gamma)_i \vee ((A, B, \Gamma)_i = (a, b, c)_i)$*
- 15: *// $(a, b, c)_i$ does not contain differentials that are not already in $(A, B, \Gamma)_i$*
- 16: $\{(a, b, c)_i\} \leftarrow \emptyset; (e_A, e_B, e_\Gamma)_i \leftarrow (a, b, c)_i$
- 17: *// set the i -th ADD TD transition of the running trail τ' to the overlapping*
 ADD TD differential $(e_A, e_B, e_\Gamma)_i$ before moving to the $(i + 1)$ -th transitions
- 18: $\tau'_i \leftarrow (e_A, e_B, e_\Gamma)_i$; add τ' to $\{\tau\}$
- 19: **return** $\{\tau\}$
