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

No Author Given

No Institute Given

Abstract. Accompanying pseudocode to 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

procedure truncate_trail
if $j < R$ **then**
    for `truncate` = `true`, `false` **do**
        // truncate bit τ_i^j
        if `truncate` = `true` and $\tau_i^j \neq *$ **then**
            Truncate bit $\tau_i^j \leftarrow *$ and propagate to rounds $j+1, \ldots, R-1$
            if Rules 1,2 and 3 are not violated for any round **then**
                Update τ with $\tau^j, \tau^{j+1} \ldots \tau_j^{R-1}$
                Call **truncate_trail** for next bit $i+1$ or next round $j+1$
        // do not truncate τ_i^j: move to next bit
        if `truncate` = `false` **then**
            Call **truncate_trail** for next bit $i+1$ or next round $j+1$
else
    // Last round: return a truncated version of τ
    return τ

Algorithm 2 Absorb a new TD trail $\boldsymbol{\tau}$ into exisiting set of trails $\mathfrak{T}$

Input:

$\mathfrak{T}$: set of disjoint TD trails; $\boldsymbol{\tau}$: new TD trail (possibly $\boldsymbol{\tau} \in \mathfrak{T}$)

Output:

$\mathfrak{T}'$: updated set of disjoint TD trails that contains all new (non-truncated) trails from $\boldsymbol{\tau}$ (possibly $\mathfrak{T} = \mathfrak{T}'$)

procedure tdiff_absorb_new_trail($\mathfrak{T}, \boldsymbol{\tau}$)
// initialize a set $\mathbf{t}$ of TD trails with the input trail $\boldsymbol{\tau}$
$\mathbf{t} \leftarrow \emptyset$; add $\boldsymbol{\tau}$ to $\mathbf{t}$
for all $\mathbf{T} \in \mathfrak{T}$ **do**
    if $\mathbf{t} = \emptyset$ **then**
        // all trails in $\mathbf{t}$ have been fully absorbed; return
        return $\mathfrak{T}$
    // absorb $\mathbf{t}$ into $\mathbf{T}$ and store the remainder in $\mathbf{t}'$
    $\mathbf{t}' \leftarrow \emptyset$
    for all $\boldsymbol{\tau} \in \mathbf{t}$ **do**
        // if $\boldsymbol{\tau}$ contains trails not already in $\mathbf{T}$, then split $\boldsymbol{\tau}$ into TD trail subsets to exclude duplicates using
        if $(\mathbf{T} \subset \boldsymbol{\tau}) \vee (\mathbf{T}, \boldsymbol{\tau} : \mathsf{PO})$ **then**
            $\mathbf{t}_{\text{temp}} \leftarrow$ **tdiff_madd_trails_make_disjoint**($\mathbf{T}, \boldsymbol{\tau}$)
            add $\mathbf{t}_{\text{temp}}$ to $\mathbf{t}'$
        // if $\boldsymbol{\tau}, \mathbf{T}$: disjoint, then all trails in $\boldsymbol{\tau}$ are new, so add it
        if $(\boldsymbol{\tau}, \mathbf{T})$: disjoint **then**
            add $\boldsymbol{\tau}$ to $\mathbf{t}'$
        // if $\boldsymbol{\tau}$ is a subset of $\mathbf{T} \implies$ it contains no new trails, so do nothing
        if $(\boldsymbol{\tau} \subset \mathbf{T})$ **then**
            continue
    // overwrite $\mathbf{t}$ with the part of it that was not absorbed i.e. $\mathbf{t}'$
    $\mathbf{t} \leftarrow \mathbf{t}'$
// $\mathbf{t}$ contains all trails not absorbed in $\mathfrak{T}$ – add them to $\mathfrak{T}$ and return
$\mathfrak{T}' \leftarrow \mathfrak{T} \cup \mathbf{t}$
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

procedure xdp_dset_add_i($i, (\alpha\beta\gamma)_i, (\alpha\beta\gamma)_{i-1}$)
if $(\alpha\beta\gamma)_i, (\alpha\beta\gamma)_{i-1}$ contradict Rules 1, 2, 3 **then**
    return -1
$p_{i-1} \leftarrow 1$ // initialize Pr to 1
if $1 \leq i \leq n-1$ **then**
    if $(i-1) = 0$ **then**
        // By *Rule 4*: $(\alpha_0 \neq *) \wedge (\beta_0 \neq *) \wedge (\gamma_0 \neq *)$
        if $\alpha_0 \oplus \beta_0 \neq \gamma_0$ **then** $p_{i-1} \leftarrow 0$ **else** $p_{i-1} \leftarrow 1$ // Theorem 1
    // non-truncated case
    if $(\alpha_{i-1} \neq *) \wedge (\beta_{i-1} \neq *) \wedge (\gamma_{i-1} \neq *)$ **then**
        if $\alpha_{i-1} = \beta_{i-1} = \gamma_{i-1}$ **then**
            // By *Rule 2*: $(\alpha_i \neq *) \wedge (\beta_i \neq *) \wedge (\gamma_i \neq *)$
            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
        else
            $p_{i-1} \leftarrow 1/2$ // Theorem 1 for $\neg(\alpha_{i-1} = \beta_{i-1} = \gamma_{i-1})$
    // truncated case
    if $(\alpha_{i-1} = *) \vee (\beta_{i-1} = *) \vee (\gamma_{i-1} = *)$ **then**
        // w.l.o.g. assume $(\alpha_{i-1} = *)$: by *Rule 1* $\implies (\beta_{i-1} \neq *) \wedge (\gamma_{i-1} \neq *)$
        if $\beta_{i-1} = \gamma_{i-1}$ **then**
            // By *Rule 2*: $(\alpha_i \neq *) \wedge (\beta_i \neq *) \wedge (\gamma_i \neq *)$
            if $\alpha_i \oplus \beta_i \oplus \gamma_i = \beta_{i-1}$ **then**
                // $(\alpha_{i-1} = *) \implies$ 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 $p_{i-1} = 1 + 1/2 = 3/2$
                $p_{i-1} \leftarrow 3/2$
            else
                $p_{i-1} \leftarrow 0$ // Theorem 1
        if $\beta_{i-1} \neq \gamma_{i-1}$ **then**
            // $\neg(\alpha_{i-1} = \beta_{i-1} = \gamma_{i-1})$: $\forall \alpha_{i-1}$, 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 $p_{i-1} = 1/2 + 1/2 = 1$
            $p_{i-1} \leftarrow 1$
// MSB
if $i = n$ **then**
    // 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 $p_{n-1} = 2$
    if $(\alpha_{n-1} = *) \vee (\beta_{n-1} = *) \vee (\gamma_{n-1} = *)$ **then**
        $p_{n-1} \leftarrow 2$
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:
$p_{i'}^{j'}, \overline{p}_{i'}^{j'}$: Pr of the beta chain for the cases resp. $\beta_{i'}^{j'} = 0$ and $\beta_{i'}^{j'} = 1$; updated V;

procedure speck_trail_beta_chain($\beta_{i'}^{j'} = *, V$)
  $\beta_{\text{prev}} \leftarrow \emptyset$; $\overline{\beta}_{\text{prev}} \leftarrow \emptyset$ // temporary variables to store β_{i-l}^{j-1}
  $i \leftarrow i'$, $p_{i'}^{j'} \leftarrow 1$, $\overline{p}_{i'}^{j'} \leftarrow 1$ // initialize the beta chain Pr and the start bit i
  for $j = j' \dots R-1$ **do**
    if $j = j'$ **then**
      // start of a beta chain: initialize the two possibilities for $\beta_i^j = *$
      $\beta_i^j \leftarrow 0$; $\overline{\beta}_i^j \leftarrow 1$
    if $j > j'$ **then**
      // middle of beta chain: compute β_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 β_{i-l}^{j-1} is stored in β_{prev}
      $\beta_i^j \leftarrow \gamma_i^{j-1} \oplus \beta_{\text{prev}}$; $\overline{\beta}_i^j \leftarrow \gamma_i^{j-1} \oplus \overline{\beta}_{\text{prev}}$
    // the conditional Pr of bit $i+1$, round j for β_i^j and $\overline{\beta}_i^j$ (Alg. (3))
    $p_i^j \leftarrow$ **xdp_dset_add_i**$(i+1, (\mathrm{A}, \mathrm{B}, \mathit{\Gamma})_{i+1}^j, (\alpha, \beta, \gamma)_i^j)$
    $\overline{p}_i^j \leftarrow$ **xdp_dset_add_i**$(i+1, (\mathrm{A}, \mathrm{B}, \mathit{\Gamma})_{i+1}^j, (\alpha, \overline{\beta}, \gamma)_i^j)$
    // accumulate the Pr to the total Pr of the beta chain
    $p_{i'}^{j'} \leftarrow (p_{i'}^{j'} \cdot p_i^j)$; $\overline{p}_{i'}^{j'} \leftarrow (\overline{p}_{i'}^{j'} \cdot \overline{p}_i^j)$
    $\beta_{\text{prev}} \leftarrow \beta_i^j$; $\overline{\beta}_{\text{prev}} \leftarrow \overline{\beta}_i^j$ // store the values of $\beta_i^j = *$
    $V_{ji} \leftarrow$ `true` // mark the bit of the beta chain as visited
    // 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
    $i \leftarrow (i + l) \mod n$
return $p_{i'}^{j'}, \overline{p}_{i'}^{j'}$

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:

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

procedure speck_trail_gamma_chain($\gamma_{i'}^{j'} = *, V$)
$i \leftarrow i'$, $j \leftarrow j'$, $p_{i'}^{j'} \leftarrow 1$, $\overline{p}_{i'}^{j'} \leftarrow 1$ // initialize the gamma chain Pr and the start j, i
// start of a gamma chain: initialize the two possibilities for $\gamma_i^j = *$
$\gamma_i^j \leftarrow 0$; $\overline{\gamma}_i^j \leftarrow 1$
// the conditional Pr of bit $i+1$, round j for γ_i^j and $\overline{\gamma}_i^j$ (Alg. (3))
$p_i^j \leftarrow$ **xdp_dset_add_i**$(i+1, (\mathrm{A}, \mathrm{B}, \mathit{\Gamma})_{i+1}^j, (\alpha, \beta, \gamma)_i^j)$
$\overline{p}_i^j \leftarrow$ **xdp_dset_add_i**$(i+1, (\mathrm{A}, \mathrm{B}, \mathit{\Gamma})_{i+1}^j, (\alpha, \beta, \overline{\gamma})_i^j)$
// accumulate the Pr at (j, i) to the total Pr of the gamma chain
$p_{i'}^{j'} \leftarrow (p_{i'}^{j'} \cdot p_i^j)$; $\overline{p}_{i'}^{j'} \leftarrow (\overline{p}_{i'}^{j'} \cdot \overline{p}_i^j)$
$V_{ji} \leftarrow$ `true` // mark the bit of the gamma chain as visited
// $\gamma_i^j = \alpha_{i-r}^{j+1}$ through the right rotation by r; process the two possibilities for α_{i-r}^{j+1}
$p_{i-r}^{j+1} \leftarrow$ **xdp_dset_add_i**$(i-r+1, (\mathrm{A}, \mathrm{B}, \mathit{\Gamma})_{i-r+1}^{j+1}, (\alpha, \beta, \gamma)_{i-r}^{j+1})$
$\overline{p}_{i-r}^{j+1} \leftarrow$ **xdp_dset_add_i**$(i-r+1, (\mathrm{A}, \mathrm{B}, \mathit{\Gamma})_{i-r+1}^{j+1}, (\alpha, \beta, \overline{\gamma})_{i-r}^{j+1})$
// accumulate the Pr at $(j+1, i-r)$ to the total Pr of the gamma chain
$p_{i'}^{j'} \leftarrow (p_{i'}^{j'} \cdot p_{i-r}^{j+1})$; $\overline{p}_{i'}^{j'} \leftarrow (\overline{p}_{i'}^{j'} \cdot \overline{p}_{i-r}^{j+1})$
$V_{(j+1)(i-r)} \leftarrow$ `true` // mark the bit of the gamma chain as visited
// $\gamma_i^j = *$ 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)
$p_{i'}^{j'}$, $\overline{p}_{i'}^{j'} \leftarrow$ **speck_trail_beta_chain**$(\beta_i^{j+1} = *, V)$
return $p_{i'}^{j'}$, $\overline{p}_{i'}^{j'}$

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**)

procedure xdp_dset_speck_trail(T)
// Initialize array of visited bits in **T** indexed by j: round; i: bit position
$\forall j, i : \ 0 \leq j \leq R-1, \ 0 \leq i \leq (n-1)$: $V_{ji} \leftarrow$ `false`
$p \leftarrow 1$ // initialize Pr of trail **T**
for $j = 0 \ldots R-1$ **do**
    for $i = 0 \ldots n-1$ **do**
        // start of gamma chain – process with Alg. (5)
        if $(\gamma_i^j = *) \wedge (V_{ji} =$ `false`$)$ **then**
            $(p_i^j, \overline{p}_i^j) \leftarrow$ **speck_trail_gamma_chain**(γ_i^j, V)
            $p \leftarrow p \ (p_i^j + \overline{p}_i^j)$ // accumulate Pr
        // start of beta chain – process with Alg. (4)
        if $(\beta_i^j = *) \wedge (V_{ji} =$ `false`$)$ **then**
            $(p_i^j, \overline{p}_i^j) \leftarrow$ **speck_trail_beta_chain**(β_i^j, V)
            $p \leftarrow p \ (p_i^j + \overline{p}_i^j)$ // accumulate Pr
        // if there are no dependency chains – compute the conditional Pr of the single bit position using Alg. (3)
        if $(\beta_i^j \neq *) \wedge (\gamma_i^j \neq *) \wedge (V_{ji} =$ `false`$)$ **then**
            $p_i^j \leftarrow$ **xdp_dset_add_i**$(i+1, (\mathrm{A}, \mathrm{B}, \mathit{\Gamma})_{i+1}^j, (\alpha, \beta, \gamma)_i^j)$
            $V_{ji} \leftarrow$ `true` // mark the bit as visited
            $p \leftarrow p \ p_i^j$ // accumulate Pr
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. $(a \subset \mathrm{A}) \vee (\mathrm{A} \subset a) \vee (\mathrm{A}, a : \mathsf{PO})$ (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; $\forall i : a_i \in \{a\} : a_i, \mathrm{A}$: 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)

procedure tdiff_make_disjoint(A, a)
if $(\mathrm{A} \subset a) \vee (\mathrm{A}, a : \mathsf{PO})$ **then**
    $m \leftarrow$ mask for the bits that are $*$ in a and $\cdot$ in A
    // set the bits that are $*$ in a and $\cdot$ in A to the values in A
    // the result TD e is the set of differences generated by a that are already in A
    $e \leftarrow a$ `OR` (A `AND` m)
    // Split a into subsets that exclude the overalpping TD e
    $\{a\} \leftarrow \emptyset$
    for all bit positions i where $a_i = *$ **do**
        $a' \leftarrow a$
        // set all stars up to the $(i-2)$-nd star to the value in e
        set $(a'_0, \ldots, a'_{i-2})$ to $(e_0, \ldots, e_{i-2})$
        // set the $(i-1)$-st star to the negated value of e_{i-1} to ensure a', e: disjoint
        $a'_{i-1} \leftarrow 1 \oplus e_{i-1}$
        // leave all stars at positions $\geq i$ unchanged i.e. as in a
        add a' to $\{a\}$
else
    // $(a \subset \mathrm{A}) \implies$ all diffs. in a are already in A
    $e \leftarrow a$; $\{a\} \leftarrow \emptyset$
return $\{a\}, e$

Algorithm 8 Generate all ADD TD differentials in (a, b, c) that are not already in $(\mathrm{A}, \mathrm{B}, \mathit{\Gamma})$

Input:

$(\mathrm{A}, \mathrm{B}, \mathit{\Gamma}), (a, b, c)$: non-disjoint distinct ADD TD differentials i.e. $((a, b, c) \subset (\mathrm{A}, \mathrm{B}, \mathit{\Gamma})) \vee ((\mathrm{A}, \mathrm{B}, \mathit{\Gamma}) \subset (a, b, c)) \vee ((\mathrm{A}, \mathrm{B}, \mathit{\Gamma}), (a, b, c) : \mathsf{PO})$ (partially overlapping); $(\mathrm{A}, \mathrm{B}, \mathit{\Gamma}), (a, b, c)$ 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; $\forall i : (a, b, c)_i \in \{(a, b, c)\} : (a, b, c)_i, (\mathrm{A}, \mathrm{B}, \mathit{\Gamma})$: disjoint; $(\mathrm{A}, \mathrm{B}, \mathit{\Gamma}), \{(a.b.c)\}$ collectively generate the set $\mathcal{D}$ with any duplicates removed

$(e_{\mathrm{A}}, e_{\mathrm{B}}, e_{\mathit{\Gamma}})$: ADD TD differential representing the differences generated by (a, b, c) that are also in $(\mathrm{A}, \mathrm{B}, \mathit{\Gamma})$ (i.e. the set of redundant differences in (a, b, c) with respect to $(\mathrm{A}, \mathrm{B}, \mathit{\Gamma})$)

procedure tdiff_madd_make_disjoint$((\mathrm{A}, \mathrm{B}, \mathit{\Gamma}), (a, b, c))$
$\{(a, b, c)\} \leftarrow \emptyset$
// Split a into $\{a\}$ s.t. $\{a\}, \mathrm{A}$: disjoint, $a \cap \mathrm{A} = e_{\mathrm{A}}$ (Alg. (7))
$(\{a\}, e_{\mathrm{A}}) \leftarrow$**tdiff_make_disjoint**$(\mathrm{A}, a)$
if $(\mathrm{A} \subset a) \vee (\mathrm{A}, a : \mathsf{PO})$ **then**
  // for each TD $a_i \in \{a\}$ generate a new ADD TD differential (a_i, b, c): as a_i, A: disjoint $\implies (a_i, b, c), (\mathrm{A}, \mathrm{B}, \mathit{\Gamma})$: disjoint
  for $\forall a_i \in \{a\}$ **do**
    add (a_i, b, c) to $\{(a, b, c)\}$
else
  // $(a \subset \mathrm{A}) \implies e_{\mathrm{A}} \leftarrow a$; $\{a\} \leftarrow \emptyset$: do nothing
  continue
// Split b into $\{b\}$ s.t. $\{b\}, \mathrm{B}$: disjoint, $b \cap \mathrm{B} = e_{\mathrm{B}}$ (Alg. (7))
$(\{b\}, e_{\mathrm{B}}) \leftarrow$**tdiff_make_disjoint**$(\mathrm{B}, b)$
if $(\mathrm{B} \subset b) \vee (\mathrm{B}, b : \mathsf{PO})$ **then**
  // for each TD $b_i \in \{b\}$ generate a new ADD TD differential (e_{A}, b_i, c): as b_i, B: disjoint $\implies (e_{\mathrm{A}}, b_i, c), (\mathrm{A}, \mathrm{B}, \mathit{\Gamma})$: disjoint, where $e_{\mathrm{A}} = a \cap \mathrm{A}$
  for $\forall b_i \in \{b\}$ **do**
    add (e_{A}, b_i, c) to $\{(a, b, c)\}$
else
  // $(b \subset \mathrm{B}) \implies e_{\mathrm{B}} \leftarrow b$; $\{b\} \leftarrow \emptyset$: do nothing
  continue
// Split c into $\{c\}$ s.t. $\{c\}, \mathit{\Gamma}$: disjoint, $c \cap \mathit{\Gamma} = e_{\mathit{\Gamma}}$ (Alg. (7))
$(\{c\}, e_{\mathit{\Gamma}}) \leftarrow$**tdiff_make_disjoint**$(\mathit{\Gamma}, c)$
if $(\mathit{\Gamma} \subset c) \vee (\mathit{\Gamma}, c : \mathsf{PO})$ **then**
  // for each TD $c_i \in \{c\}$ generate a new ADD TD differential $(e_{\mathrm{A}}, e_{\mathrm{B}}, c_i)$: as $c_i, \mathit{\Gamma}$: disjoint $\implies (e_{\mathrm{A}}, e_{\mathrm{B}}, c_i), (\mathrm{A}, \mathrm{B}, \mathit{\Gamma})$: disjoint, where $e_{\mathrm{A}} = a \cap \mathrm{A}$, $e_{\mathrm{B}} = b \cap \mathrm{B}$
  for $\forall c_i \in \{c\}$ **do**
    add $(e_{\mathrm{A}}, e_{\mathrm{B}}, c_i)$ to $\{(a, b, c)\}$
else
  // $(c \subset \mathit{\Gamma}) \implies e_{\mathit{\Gamma}} \leftarrow c$; $\{c\} \leftarrow \emptyset$: do nothing
  continue
return $\{(a, b, c)\}$, $(e_{\mathrm{A}}, e_{\mathrm{B}}, e_{\mathit{\Gamma}})$

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

Input:

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

Output:

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

```
procedure tdiff_madd_trails_make_disjoint(T, τ)
// initialize a temporary (running) trail τ' to τ
{τ} ← ∅; τ' ← τ
for i = 0...(n − 1) do
    // get the i-th ADD TD differentials of the trails τ, T
    (a, b, c)_i ← τ_i; (A, B, Γ)_i ← T_i
    // if (a, b, c)_i contains differentails (αβγ)_i that are not already in (A, B, Γ)_i
    then split (a, b, c)_i into subsets to remove the overlap using Alg. (8)
    if ((A, B, Γ)_i ⊂ (a, b, c)_i) ∨ ((A, B, Γ)_i, (a, b, c)_i : PO) ∧ ((A, B, Γ)_i ≠ (a, b, c)_i)
    then
        {(a, b, c)_i}, (e_A, e_B, e_Γ)_i ← tdiff_madd_make_disjoint((A, B, Γ)_i, (a, b, c)_i)
        // 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, Γ)_i: disjoint ⟹ τ', T: dis-
        joint
        for each (a, b, c)_i ∈ {(a, b, c)_i} do
            τ'_i ← (a, b, c)_i; add τ' to {τ}
    else
        // ((a, b, c)_i ⊂ (A, B, Γ)_i) ∨ ((A, B, Γ)_i = (a, b, c)_i)
        // (a, b, c)_i does not contain differentials that are not already in (A, B, Γ)_i
        {(a, b, c)_i} ← ∅; (e_A, e_B, e_Γ)_i ← (a, b, c)_i
    // set the i-th ADD TDtransition of the running trail τ' to the overlapping
    ADD TD differential (e_A, e_B, e_Γ)_i before moving to the (i + 1)-th transitions
    τ'_i ← (e_A, e_B, e_Γ)_i; add τ' to {τ}
return {τ}
```