An Algorithm for Reachability Problem of One-Counter Automata

XXXX

TODO: Add abstract here.

1 Introduction

2 Premilinaries

2.1 One-Counter Automata

Definition 1 (Counter Automata) *A k-counter automaton $\mathcal{A}$ is a tuple $\mathcal{A} = (Q, \Lambda, q_0, F, \Delta, \lambda, \epsilon)$ where*

- *Q is a finite set of states,*
- *Λ is a finite set of labels,*
- *$q_0 \in Q$ is the initial state,*
- *$F \subseteq Q$ is the set of final states,*
- *$\Delta \subseteq Q \times Q$ is the transition relation,*
- *$\lambda : Q \to 2^{\Lambda}$ is the state labelling function, and*
- *$\epsilon : \Delta \to Op$ is a transition labelling function where $Op = \{add_i(z) : i \in [k], z \in \mathbb{Z}\} \cup \{zero_i : i \in [k]\}$ is a set of operations.*

Definition 2 (One-Counter Automata) *An*

3 Algorithm

According to (*1*), which gave a theoretical proof of the decidability and NP-lower and -upper bound for the reachability problem of one-counter automata. Given a one-counter automata $\mathcal{A}$ and its corresponding transition system $T(\mathcal{A})$ defined in (*1*). The reachability problem of one-counter auotmata can be stated as follows.

Reachability Problem of One-Counter Automata

INPUT: An one-coutner automata $\mathcal{A}$ and configurations $C, C' \in C(\mathcal{A})$, where $C(\mathcal{A}) = Q \times \mathbb{N}$.

OUTPUT: Does $C \rightarrow_{\mathcal{A}}^{*} C'$?

More generally, we would like to use two variables to represent the counter values of the initial configuration and the ending configuration. Then we can convert the general problem into a QFPA(Quantifier-Free Presburger Arithmetic) Formula, such that the ending configuration is reachable from the initial configuration iff the QFPA formula is satisfiable.

General Reachability Problem of One-Counter Automata

INPUT: An one-coutner automata $\mathcal{A}$ and configurations $C, C' \in C(\mathcal{A})$, where $C(\mathcal{A}) = Q \times Var$ and Var is the set of variables in the scope of natural number.

OUTPUT: A QFPA formula ϕ that is satisfiable iff $C \rightarrow_{\mathcal{A}}^{*} C'$.

Due to the counter value and the possible infinity of the configurations of one-counter automata, finding a path in $T(\mathcal{A})$ is not realistic because it may cause exponential blow up (*1*). Thus we turn to consider the flow $f : E \to \mathbb{N}$ which assigns each edge in the automata a natural number. We call a flow *path flow* if the assignment of the number exactly corresponds to the number of a path π go through the edge in the automata.

The following lemma gives neccessary conditions for a flow to be a path flow. The original proof of the lemma in (*1*) is not complete.

Lemma 1 (Lemma 4.1.6 in (*1*)) *A flow f is a s-t path flow iff f satisfies the following conditions:*

- *If s = t then*
  1. $\Sigma_{w \in out(v)} f(v, w) = \Sigma_{w \in in(v)} f(w, v)$ *for all v*
  2. $F(f)$ *is a s-t support*
- *If $s \neq t$ then*
  1. $\Sigma_{w \in out(v)} f(v, w) = \Sigma_{w \in out(v)} f(w, v)$ *for all* $v \in V - \{s, t\}$

     $\Sigma_{w \in out(s)} f(s, w) = \Sigma_{w \in out(s)} f(w, s) + 1$

     $\Sigma_{w \in out(t)} = \Sigma_{w \in out(t)} f(w, t) - 1$
  2. *F(f) is connected.*

***Proof* 1** *Prove by induction on* $n = \Sigma_{e \in E} f(e)$

- $\Rightarrow$ *By definition direction of path flow the proof is obvious.*
- $\Leftarrow$

1. $s = t, F(f) is connected$, *we have the conclusion that* f *is a s-t path flow.*
   - *case 1 : there is a self loop from* s *to* s.
   - *case 2 : there is no self loop from* s *to* s. *In this case we delete an edge from* v *to* s *such that* $v \neq s$.
     - *subcase 2.1 the resulting graph is still connectedl let the new flow be* f'. *By the induction hypothesis* f' *is a path flow from* s *to* v.
     - *subcase 2.2 the resulting graph is not connected, in this case , there are exactly two connected components such in the resulting graph, we call them* C_1 *and* C_2 *and assume they contain* s *and* v *respectively.*
       - *sub-subcase 2.2.1:* C_2 *contains no edges.*
       - *sub-subcase 2.2.2:* C_1 *and* C_2 *both contain at least one edge.*
2. $s \neq t$ *is the same idea, we omit the proof here.*

By lemma 4.1.14 in (*1*), the problem whether (q', n') is reachable from (q, n) can be solve by enumerate all the type-1, type-3, and type-2 reachability and guess nondeterministically the possible path between them. We now present the algorithm in detail.

Definition 3 (Reachability Certificate) *Defined the same as in (1)*

By the definition of type-3 reachability certificate which require the positive cycle template. We put positive cycle template on the level of SCC of the weighted graph.

Definition 4 (Weighted Graph) *Let* $\mathcal{A}$ *be an one-counter automaton, the weighted graph of* $\mathcal{A}$ *is* $G = (V, E, \mu)$ *where* V *is a finite set of vertices according to the states of* $\mathcal{A}$, $E \subseteq V \times V$ *is a finite set of edges correspond to the edges of* $\mathcal{A}$ *and* $\mu : E \to \mathbb{Z}$ *is weight function correspond to the changes of the counter value of* $\mathcal{A}$.

We call $G' = (V', E', \mu')$ a *subgraph* of a weighted graph $G = (V, E, \mu)$ if $V' \subseteq V$, $E' \subseteq E$ and $\mu' = \mu$.

We call $G^{op} = (V, E^{op}, \mu^{op})$ the *skew transpose* of the graph.

Definition 5 (Strongly Connected Component(SCC)) *A strongly connected component of a weighted graph G is a subgraph $G' = (V', E', \mu)$ of G and for $\forall v, v' \in V'$ and $v \neq v'$, there exist a path π from v to v' and a path π' from v' to v.*

We have to add some structure on an SCC to make the algorithm works. We do some classification on thetype of vertices in an SCC:

Given an SCC $G' = (V', E', \mu)$ which is the subgraph of a weighted graph $G = (V, E, \mu)$, we call a vertice $v' \in V'$

- an *in-port vertex* if there exists $u \in V \wedge u \notin V'$ such that $(u, v) \in E$,
- an *out-port vertex* if there exists $u \in V \wedge u \notin V'$ such that $(v, u) \in E$,
- and an *internal vertex* otherwise.

Definition 6 (Drop-Weight Table(DWT)) *Given a graph $G = (V, E, \mu)$, we define its drop-weight table DWT_G and simply use a tuple $DWT_G(v_1, v_2, l, P)$ to represent a entry of the table, where*

- *$v_1 \in V$ is the starting node of paths and $v_2 \in V$ is the ending node of paths.*
- *$l \in \mathbb{N}$ denote the maximum length of paths.*
- *$P = \{(d, w) \mid w \in \mathbb{Z}, d \in \mathbb{Z}\}$ is a set remembering all the drop-weight pairs.*

Given a integer n, we use DWT_G^n to denote a drop-weight table of G that the length of paths we consider is at most n.

Algorithm 1 Compute DWT

function COMPUTEDWT(G, n)
Input: A graph $G = (V, E, \mu)$ and an integer $n \in \mathbb{N}$
Output: A DWT table DWT_G^n
▷ Initialize the table by the edges of the graph.
for each $v \in V$ **do**
for each $(v, w) \in E$ **do**
For entry $DWT_G^n(v, w, 1, P)$
construct (d, w) where
$d = min(0, \mu(v, w))$, $w = \mu(v, w)$
$P = P \cup \{(d, w)\}$
end for
end for

for $i := 2$ to n **do**
for $len := 1$ to $i - 1$ **do**
$preLen := len$
$sufLen := i - preLen$
$P_i := \bigcup_{DWT_G^n(s,t,i-1,P_k)} P_k$
for each $s \in V$ **do**
for each $t \in V$ **do**
for each $DWT_G^n(s, m, preLen, P_1)$ **do**
for each $DWT_G^n(m, t, sufLen, P_2)$ **do**
Let $(d_1, w_1) \in P_1$ and $(d_2, w_2) \in P_2$
Construct (d', w') from all possible tuple pairs:
$d' := min(d_1, w_1 + d_2)$
$w' := w_1 + w_2$
$P_i := P_i \cup \{(d', w')\}$
end for
end for
Add (s, t, i, P_i) to the table.
end for
end for
end for
end for
end function

With the definition of drop-weight table, we now need a algorithm to compute a table.

The basic idea of the algorithm is that we first put all the edge information into the table, then for paths length l larger than 1 we split the path into a prefix and a suffix whose lengths are both smaller than l. Since we have already compute the table for paths that have a length smaller than l. We can use the result to acquire the result we need.

In order to get a path $\Pi = \pi_1 \cdot \pi_2 \cdot \pi_3$ from s to t in $\mathcal{A}$ corresponding to a run in $T(\mathcal{A})$. Algorithm needs to guess the start node v_1 and end node v_2 of π_2. By lemma 4.1.17, if $|\pi_2| > 0$ there should be a positive cycle template at v_1 and a positive cycle template. Intuitively, a positive cycle template gives a overlook on the possible positive cycles in a SCC. That means if there is a simple positive cycle in an SCC, all other node can find a path to a point on that cycle and by circling the simple positive cycle many times. Then we can get a positive cycle on another point.

In the algorithm we first convert the one-counter automaton into weighted graph and then shrink the graph into a abstract graph defined on SCC. Now we define the shrinked weighted graph.

Definition 7 (Shrinked Weighted Graph) *Given a weighted graph $G = (V, E, \mu)$ we defined its corresponding shrinked weighted graph $SWG = (SCC(V), E', \mu')$, where*

- *$SCC(V) \subseteq V \times \mathbb{Z}^+$ is the set of all the SCCs and vertices in the same SCC have the same integer value.*
- *$E' \subseteq SCC(V) \times SCC(V)$ is a finite set of transitions between the SCCs. We also require for any $v, v' \in V$ and $a, b \in \mathbb{Z}^+$, if $((v, a), (v'b)) \in E'$ then $a \neq b$*
- *$\mu' : E' \to \mathbb{Z}$ is the weight function that assign each edge a weight. The value remain the same as μ in G.*

By the definition of SWG, it is obvious that it is a direct acycly graph. Now we need an algorithm to convert a weighted graph into a SWG by using Tarjan's algorithm [Add reference] to find the SCCs in a weighted graph.

Now by executing algorithm 2 we preprocess a given WG into a SWG.

Definition 8 (Abstract Shrinked Weighted Graph) *Given an $SWG = (SCC(V), E', \mu')$, we define its corresponding abstract shrinked weighted graph $ASWG = (S, E'', T, \mu', \eta)$ where*

- *S is a finite set of abstract state. A element $s \in S$ represents an SCC in SWG.*
- *$\eta : S \to SCC(V)$ is a function that maps a state in S to its corresponding SCC.*
- *$E'' \subseteq E'$ is a set of detailed edges of the form $((v_1, a), (v_2, b))$ where $(v_1, a), (v_2, b) \in SCC(V)$ and $a \neq b$,*
- *$T \subseteq S \times S$ is a finite set of abstract edges. Given $s_1, s_2 \in S$ and $(s_1, s_2) \in T$ iff there exists a edge $((v_1, t_1), (v_2, t_2)) \in E''$ such that $t_1 = \eta(s_1)$ and $t_2 = \eta(s_2)$.*
- *$\sigma : S \to \{\{+\}, \{-\}, \{+, -\}, \{0\}\}$ is a function denoting whether there is a positive cycle($\{+\}$), negative cycle($\{-\}$), both($\{+, -\}$), or none($\{0\}$).*

We call the path $\tau : s_1 \cdots s_n$ on an ASWG a $abstract\ path$ where $s_i \in S$ and $(s_i, s_{i+1}) \in T$

Given a weighted graph (V, E, μ) and its corresponding ASWG $(S < E'', T, \mu, \eta)$. We now define the operator $f_{SCC} : V \to S$ that maps a vertice to the SCC contains it.

Algorithm 3 converts an SWG to an ASWG.

The idea of the algorithm goes as follows. Given an one-counter automaton we first convert it into a weighted graph[Hasse]. Then use algorithm 2 and algorithm 3 we get corresponding SWG and ASWG. Then we need to finish the σ function in ASWG. The idea is using the algorithm 1 to compute the DWT of the SCCs and detect whether there is a positive, negative cycle in SCC. Here is the algorithm.

Algorithm 2 WG to SWG

$SCCIndex := 1$
function WG2SWG(G)
    Input: A weighted graph $G = (V, E, \mu)$
    Output: A shrinked weighted graph $SWG = (SCC(V), E', \mu')$

    $index$:= 1
    S := empty stack
    Initialize an empty shrinked weighted graph G':
    $SCC(V) := V \times 0, E := E, \mu' := \mu$

    for each $(v, mark) \in SCC(V)$ **do**
        if mark == 0 **then**
            STRONGCONNECT($(v, mark)$)
        end if
    end for
    return SWG
end function

function STRONGCONNECT($(v, mark)$)
    v.index := $index$
    v.lowlink := $index$
    $index$:= $index$ + 1
    S.push(v)
    v.onStack := **true**

▷ Consider successors of v
    for each $(v, w) \in E$ **do**
        if w.index == 0 **then**
            STRONGCONNECT(w)
            v.lowlink := $min(v$.lowlink, w.lowlink$)$
        else if w.onStack **then**
            v.lowlink := $min(v$.lowlink, w.index$)$
        end if
    end for

    if v.lowlink == v.index **then**
        repeat
            $w := S.pop()$
            w.onStack := **false**
            $(w, mark) := (w, SCCIndex)$
▷ Add w to current strongly connected component
        until $w == v$
        $SCCIndex := SCCIndex + 1$
▷ Output the current strongly connected component
    end if
end function

Algorithm 3 SWG to ASWG

function SWG2ASWG(SWG)
Input: $SWG = (SCC(V), E', \mu')$
Output: $ASWG = (S, E'', T, \mu', \eta, \sigma)$

$SCCNum := 0$
for each $(v, n) \in SCC(V)$ **do**
if $n > SCCNum$ **then**
$SCCNum := n$ ▷ Find the number of the SCCs in SWG.
end if
end for
Initialize ASWG:
$S := \{s_i \mid i \in [1, \ldots, SCCNum]\}$
$E'', T := \emptyset$
for each $((v_1, n_1), (v_2, n_2) \in E')$ **do**
if $n_1 \neq n_2$ **then**
Add $(v_1, n_1), (v_2, n_2)$ *to* E''.
if $(s_{n_1}, s_{n_2}) \notin T$ **then**
$T := T \cup \{(s_{n_1}, s_{n_2})\}$
end if
end if
end for

for each $s \in S$ **do**
$\sigma(s) :=$ TESTCYCLE(S)
end for
end function

Algorithm 4 Compute σ in $ASWG$

function TESTCYCLE(SCC)
  Input: $SCC = (V', E', \mu)$
  Output: $o \in \{\{+\}, \{-\}, \{+, -\}, \{0\}\}$.
  $DWT :=$ COMPUTEDWT(SCC, n)
  for each $v \in V'$ **do**
    for each $(v, v, n, P) \in DWT^n_{SCC}$ **do**
      for each $(d, w) \in P$ **do**
        if $w > 0$ **then**
          $isPos :=$ **true**
        else if $w < 0$ **then**
          $isNeg :=$ **true**
        end if
      end for
    end for
  end for

  if $isPos \wedge isNeg$ **then**
    return $\{+, -\}$
  else if $isPos$ **then**
    return $\{+\}$
  else if $isNeg$ **then**
    return $\{-\}$
  else
    return $\{0\}$
  end if
end function

By (*1*) the correctness of deciding whether an SCC contains a positive cycle template is given by the following lemma.

Lemma 2 *Given a strongly connected component $SCC = (V', E', \mu)$. There is a simple positive cycle in the SCC iff for any $v \in V'$ there is a positive v-cycle template respect to an $n \in \mathbb{N}$.*

Proof 2

- *$\Leftarrow$: By the definition of v-cycle template this direction is obvious.*
- *$\Rightarrow$: By the definition of v-cycle template in (1) and the definition of the strongly connected component in this paper. Assume there is a positive cycle π_+ in the SCC, there exists a path from v to any point one the v-cycle π_+. Here we only take a $p \in V'$ on π_+ and a v-p path $\pi_{v\text{-}p}$. Let $n = min(drop(\pi_+) + weight(\pi_{v\text{-}p}), drop(\pi_{v\text{-}p}))$. Then we have a positive v-cycle template $\pi = p_1 \cdot p_2 \cdot p_3$ respect to n where $p_1 = \pi_{v\text{-}p}$ and $p_2 = \pi_+$.*

After computing σ in $ASWG = (S, E'', T, \mu', \eta, \sigma)$ there are four kinds of situations to be considered:

- **Situation 1:** There exists $s_1, s_2 \in S$ (possibly $s_1 = s_2$) such that $+ \in \sigma(s_1)$ and $- \in \sigma(s_2)$
- **Situation 2:** For any $s \in S$, $\sigma(s) = \{+\}$ or $\sigma(s) = \{0\}$
- **Situation 3:** For any $s \in S$, $\sigma(s) = \{-\}$ or $\sigma(s) = \{0\}$
- **Situation 4:** For any $s \in S$, $\sigma(s) = \{0\}$.

Due to the requirement that a type-3 reachability certificate require a positive cycle template and a negative cycle, we only need to find type-3 reachability certificate in **Situation 1**.

How to find all the possible path that may cause a reachability certificate. The diffcult problem is how to settle all the nondeterminism in the algorithm. Recall the definition of strongly

connected componenet and its internal, inport and outport vertices before, we will use them to analyse how to generate a QFPA formula for reachability in the different situations above.

Assume the reachability problem ask given an-counter automaton $\mathcal{A}$ and configurations (s, n) an (t, n'), whether there is a run $(s, n) \to^* (t, n')$

Firstly convert the one-counter automaton to a weighted graph $G = (V, E, \mu)$, an SWG $G' = (SCC(V), E', \mu')$ and an ASWG $G'' = (S, E'', T, \mu', \eta)$.

Let $m = f_{SCC}(s) \in S, n = f_{SCC}(t) \in S$.

- **Situation 1:**

  Basic idea: Enumerate all the possible tuples (p, q) where $p, q \in S$, $+ \in \sigma(p)$ and $- \in \sigma(q)$.

  We first do DFS with a depth bound $|S|$ to find all the abstract path $\tau_1 : m \cdots p$ where $+ \in \sigma(p)$ and put these abstract path into set P_1.

  Likewise, do the same thing in G''^{op} and find all the abstract path $\tau_3^{op} : n \cdots q$ where $- \in \sigma(q)$. convert τ_3^{op} to τ_3 and put all these abstract paths into set P_3.

  Then, we need to find whether there is a abstract path between p and q, the idea is also finding path by DFS. For pair (p, q), put all the abstract path $\tau_2 = p \cdots q$ into a set $P_2^{(p,q)}$.

  With set $P_1^p, P_3^{(p,q)}, P_2^q$ where $p, q \in S$ we can now generate an abstract path that can be use to construct the reachability certificate by choosing abstract paths from $P_1^p, P_3^{(p,q)}, P_2^q$ and concatenante them. We summarize the process above in algorithm 4.

  Now we have solved the nondeterminism of choosing possible (p, q), the nodeterminism of choosing the concrete path between SCCs and the possible paths in an SCC from inports to outports.

  By the definition of type-1 reachability certificate in (*1*), which requries a support without positive cycles, weight and the correct sum of the weight along the path. It is obvious that

Algorithm 5 Find Type-3 Abstract Paths

function FINDTYPE3(WG, s, t)
Input: A weighted graph WG, starting vertex s and ending vertex t.
Output: An set of lists $\{[P_1^p, P_3^{(p,q)}, P_2^q]\}_{(p,q)}$ where p, q are states in corresponding $ASWG$
SWG := WG2SWG(WG)
$ASWG$:= SWG2ASWG(SWG)
$PosSet := \{s \mid s \in ASWG.S, + \in ASWG.\sigma(s)\}$
$NegSet := \{s \mid s \in ASWG.S, - \in ASWG.\sigma(s)\}$
$list$:= []
for each $p \in PosSet$ **do**
P_1^p := DFSFINDABSPATH($ASWG$, s, p)
for each $q \in NegSet$ **do**
$P_3^{(p,q)}$:= DFSFINDABSPATH($ASWG$, p, q)
P_2^q := DFSFINDABSPATH($ASWG$, q, t)
$list := list \cup \{[P_1^p, P_3^{(p,q)}, P_2^q]\}$
end for
end for
return $list$
end function

function DFSFINDABSPATH($ASWG$, p_1, p_2)
Input: An $ASWG = (S, E'', T, \mu, \eta, \sigma)$, starting state p_1 and target state p_2.
Output: a set of all the abstract paths P.
$current := p_1$
S := empty stack
$S.push(p_1)$
while $\exists(current, u) \in ASWG.T$ not visited **and** $current \neq$ **null do**
$S.push(u)$
$current := u$
if $current == p_2$ **then**
Output the stack as a path to P
$S.pop()$
$current := S.top()$
end if
if all $(current, u) \in ASWG.T$ are visited **then**
$S.pop()$
$current := S.top()$
end if
end while
end function

on a path the concrete edges between SCCs will at most be visited once. Hence we can split the type-1 reachability certificate into SCCs the type-1 abstract path visited.

Given a SCC $G' = (V', E', \mu)$ by the definitions above, there is a set of inports V_{in} and a set of outports V_{out}.

Let $\{(v_i, v_o) \mid v_i \in V_{in}, v_o \in V_{out}\}$ be all the inport-outport tuples. Due to the storng connectivity of SCC, there exists at least one path in G' starting from v_i and ending at v_o. Since for one tuple there are often many reachable paths in the graph, we would like to introduce a dynamic programming techniques which saves time we computing the drop and weight of a path whose length is less than a integer value. We define a table that store the intermediate result of the computation.

Having a DWT it is much more convienient to generate the QFPA formula that related to the properties of path flows and paths. We take the generation of the positive cycle template as an example.

By the definition of positive cycle template and our assumption that positive cycle appears in an SCC with a $+$ tag. In that SCC, assume the inport is v_1 and outport is t'. To find a vertex v with a simple positive cycle we only need to check $DWT^n_{SCC}(v, v, n, P_2)$ where $\exists(d_2, w_2) \in P \wedge w > 0$. A path from inport v_1 to v is also needed, hence we check $DWT^n_G(v_1, v, n, P_1)$. Then a positive cycle template can be formed as

$$\exists x \bigvee_{(d_1,w_1)\in P_1\ (d_2,w_2)\in P} x + d_1 \geq 0 \wedge x + w_1 + d_2 \geq 0 \wedge w_2 > 0$$

where x is the initial counter value and we treat it as a variable.

As shown by the example above, it is much easier to write the QFPA formula for the existence of type-2 reachability certificate π_2.

Before explaining the idea of the algorithm for type-1 reachability certificate, we first propose a lemma.

Lemma 3 *Given an* $SCC = (V', E', \mu)$ *and an inport* $v_i \in V'$ *and a outport* $v_o \in V'$. *Assume* $|V'| \geq 2$ *and there is a path* $\pi = v_i \cdots v_o$ *that contains cycles and there is no positive cycle along the path. If* $|\pi| \geq 3|V'|^2 + 1$, *then we can split* π *into* $\pi = p_1 \cdot p_2 \cdot p_3$ *where* $p_1 = v_i \cdots v'$, $p_2 = v' \cdots v'$ *and* v' *appears* $3|V'| + 1$ *times,* $p_3 = v' \cdots v_o$.

***Proof* 3** *Since we assume* $|V'| \geq 2$, *then* $3|V'|^2 > |V'|^2 + 1$. *By the pigeonhole principle, if the length of the path* $|\pi| \geq 3|V'|^2 + 1$ *then there exists a* $v' \in V'$ *such that* v' *appears at least* $\lceil \frac{3|V'|^2+1}{|V'|} \rceil = 3|V'| + 1$ *times along the path. take the first and the last time* v' *appears we got the desired path* p_2. *The construction of* p_1 *and* p_3 *is natural.*

We note that since there is a loop that repeats $3|V'|+1$ times in the path. Since we assume that there is no positive cycle and we can omit 0-cycle, the minimum decrement of the path is at least $3|V'|$. And by this conclusion we have the following lemma.

Lemma 4 *Given an* $SCC = (V', E', \mu)$ *where we assume* $|V'| > 2, \mu(E') = \{1, -1\}$ *and an inport* $v_i \in V'$ *and a outport* $v_o \in V'$. *Let* $\pi = v_i \cdots v_o$ *that* $|\pi| \geq 3|V'|^2 + 1$ *and contains no positive cycles, and by lemma 3 we can split* π *into* $\pi = p_1 \cdot p_2 \cdot p_3$ *where* $p_1 = v_i \cdots v'$, $p_2 = v' \cdots v'$ *and* $p_3 = v' \cdots v_o$. *Let* x, y_1, y_2 *and* z *be the counter variable of vertices* v_i, *first* v', *last* v' *and* v_o *respectively and let* $y_1 = m$. *If such* π *exists, then we have* $x \geq |V'|$.

***Proof* 4** *Since there is no positive loop in the graph and we assume each edge increase or decrease the counter value by 1, path* p_1 *can at most increase the counter value by* $|V'|$, *i.e.* $x - y_1 \leq |V'|$. *By the analysis above, we have* $y_2 - x \leq -3|V'|$. *Consider the* p_3

we have $z - y_2 \leq |n|$. *We can easily derive* $z - x \leq -2|V'|$ *from the inequalities. Hence* $x \geq z + 2|V'|$. *Since, we require the counter value* $z \geq 0$, *we have* $x \geq 2|V'| > |V'|$.

With lemma 3 and lemma 4, we can derive a special path certificate in an $SCC = (V', E', \mu)$ when the length of the path is larger than $3|V'|^2 + 1$.

Lemma 5 *Given an* $SCC = (V', E', \mu)$ *and inport* $v_i \in V'$ *and outport* $v_o \in V'$. *If there is a path* $\pi = v_i \cdots v_o$ *in the SCC where* $|\pi| \geq 3|V'|^2 + 1$, *then there exists an path* π' *and we can split* π' *into* $\pi = p_1 \cdot p_2 \cdot p_3$ *where* $p_1 = v_i \cdots v'$, $p_2 = v' \cdots v'$ *and* $p_3 = v' \cdots v_o$ *where each counter value on vertices of* π' *is greater than 0.*

***Proof* 5** *A simply application to the path and results of lemma 3 and 4.*

As for the type-1 reachability certificate π_1 we need to consider more because type-1 certificate does not allow the existence of positive cycle in the support of the path flow.

For the sake of this requirement, we need to guess whether the support of the path flow contains a cycle when the abstract path goes through an SCC. If the path in the SCC is only a simple path, we can guess the support and use DWT easily.

Otherwise, there is a cycle in the support of the path flow. We use the algorithm computing σ to test whether there is a positive cycle in the support of the flow. If not, we can make use of the lemma 5 and DWT to generate the formula. Due to the feature of DWT table, we only need to find all the positive simple cycles in the SCC and find all the possible support that destruct all the simple cycles.

- **Situation 2:** In this situation, we have no SCC tagged with negative cycle in the grpah. Hence, path only contains a type-2 certificate.

- **Situation 3** Similar to **Situation 2**, there is only a type-1 reachability certificate.

Algorithm 6 Situation 1 Formula Generation

function SIT1FORMULA(WG, s, t)
Input: An one-counter automata $\mathcal{A}$ and starting config (s, n) and ending config (t, n').
Output: A QFPA formula $\phi^1(G)$ that is satisfiable iff t is reachable from s in situation 1.
$WG :=$ ONECOUNTER2WG($\mathcal{A}$)
$SWG :=$ WG2SWG(WG)
$ASWG :=$ SWG2ASWG(SWG)
$s_a := f_{SCC}(s)$
$t_a := f_{SCC}(t)$
$WG^{op} :=$OP(WG)
$SWG^{op} :=$OP(SWG)
$ASWG^{op} :=$OP(ASWG)
$type3Lists :=$ FINDTYPE3(WG, s, t)
for each $[P_1^p, P_3^{(p,q)}, P_2^q] \in type3Lists$ **do**
for each $p_1 \in P_1^p$ **do**
for each $p_3 \in P_3^{(p,q)}$ **do**
for each $p_2^{op} \in P_2^q$ **do**
TODOs:
For each SCC on the absPath, guess the inport and outport.
Guess the path in the SCC is a simple path or with cycles.
Guess the all the maximum support of the path in the SCC and guarantee there is no positive cycles in it.
If we guess the path is with cycles and length larger than $3|SCC.V|^2 + 1$ use path flow to generate the former half of the path and use DWT to deal with the latter half.
Generate the formula and connecting it with the origin formula with conjunction.
end for
end for
end for
end for
end function

function OP($Graph$)
Input: A WG or an SWG or an ASWG
Output: the skew transpose of the input graph.
end function

For **Situation 3**, we only need a algorithm to compute all the possible type-1 certificates and form them into a QFPA formula. This algorithm can also be used in **Situation 2** if we make use of the skew transpose of the graph. Here is the algorithm.

Algorithm 7 Situation 2 Formula Generation

```
function SIT2FORMULA(WG, s, t)
    Input: A weighted graph WG, starting vertex s and ending vertex t.
    Output: A QFPA formula φ_2 that is satisfiable iff there is a deasible path from s to t.
    SWG := WG2SWG(WG)
    ASWG := SWG2ASWG(SWG)
    P := DFSFINDABSPATH(ASWG, s, t)
    for each π_1 ∈ P do
        for each SCC on π_1 do
            if − ∈TESTCYCLE(SCC) then
                TODO:
                Use path flow to construct the formula and wedge it.
            else
                TODO:
                Guess support S
                COMPUTEDWT(S, |S.V|)
                Use DWT table to generate the formula and wedge it.
            end if
        end for
    end for
end function

function SIT1FORMULA(WG, s, t)
    Input: A weighted graph WG, starting vertex s and ending vertex t.
    Output: A QFPA formula φ_3 that is satisfiable iff there is a deasible path from s to t.
    WG^op := OP(WG)
    return SIT2FORMULA(WG^op)
end function
```

- **Situation 4:** In this situation, we can omit all the 0-cycles because they do nothing "good" for the reachability problem. That means, there is no positve and negative cycle and 0-cycle does not change the counter value. Furthermore, it may introduce larger drop in the

graph. Hence, this situation can be solve by simply guessing all the simple paths.

Finally, the whole process of the algorithm goes as follows:

1. Firstly convert the one-counter automata $\mathcal{A}$ into a weighted graph WG and compute corresponding SWG and $ASWG$.

2. Then classify the $ASWG$ into the four situations and use the algorithms to generate target QFPA.

References and Notes

1. Dmitry Chistikov and Christoph Haase. On the complexity of quantified integer programming. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 94:1–94:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.