The Commutativity Problem of the MapReduce Framework: A Transducer-Based Approach

Yu-Fang Chen[1(✉)], Lei Song[2], and Zhilin Wu[2]

[1] Institute of Information Science, Academia Sinica, Taipei, Taiwan
yfc@iis.sinica.edu.tw

[2] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China

Abstract. MapReduce is a popular programming model for data parallel computation. In MapReduce, the *reducer* produces an output from a list of inputs. Due to the scheduling policy of the platform, the inputs may arrive at the reducers in different order. The *commutativity problem* of reducers asks if the output of a reducer is independent of the order of its inputs. Although the problem is undecidable in general, the MapReduce programs in practice are usually used for data analytics and thus require very simple control flow. By exploiting the simplicity, we propose a programming language for reducers where the commutativity problem is decidable. The main idea of the reducer language is to separate the control and data flow of programs and disallow arithmetic operations in the control flow. The decision procedure for the commutativity problem is obtained through a reduction to the equivalence problem of *streaming numerical transducers* (SNTs), a novel automata model over infinite alphabets introduced in this paper. The design of SNTs is inspired by streaming transducers (Alur and Cerny, POPL 2011). Nevertheless, the two models are intrinsically different since the outputs of SNTs are integers while those of streaming transducers are data words. The decidability of the equivalence of SNTs is achieved with an involved combinatorial analysis of the evolvement of the values of the integer variables during the runs of SNTs.

1 Introduction

MapReduce is a popular framework for data parallel computation. It has been adopted in various cloud computing platforms including Hadoop [8] and Spark [16]. In a typical MapReduce program, a *mapper* reads from data sources and outputs a list of key-value pairs. The scheduler of the MapReduce framework reorganizes the pairs $(k, v_1), (k, v_2) \dots (k, v_n)$ with the same key k to a pair (k, l), where l is a list of values $v_1, v_2, \dots, v_n$, and sends (k, l) to a *reducer*. The reducer then iterates through the list and outputs a key-value pair[1]. More specifically, taking the "word-counting" program as an example. It counts the occurrences

[1] We focus on the Hadoop style reducer in this work.

S. Chaudhuri and A. Farzan (Eds.): CAV 2016, Part II, LNCS 9780, pp. 91–111, 2016.
DOI: 10.1007/978-3-319-41540-6_6

of each word in a set of documents. The mappers read the documents and output for each document a list in the form of $(word_1, count_1)$, $(word_2, count_2)$, ..., $(word_n, count_n)$, where $count_k$ is the number of occurrences of $word_k$ in the document being processed. These lists will be reorganized into the form of $(word_1, list_1), (word_2, list_2), \ldots, (word_n, list_n)$ and sent to the reducers, where $list_k$ is a list of integers recording the number of occurrences of $word_k$. Note that the *order* of the integers in the lists can differ in different executions due to the scheduling policy. This results in the *commutativity problem*.

A reducer is said to be *commutative* if its output is independent of the order of its inputs. The commutativity problem asks if a reducer is commutative. A study from Microsoft [18] reports that 58 % of the 507 reducers submitted to their MapReduce platform are non-commutative, which may lead to very tricky and hard-to-find bugs. As an evidence, those reducers already went through serious code review, testing, and experiments with real data for months. Still, among them 5 reducers containing very subtle bugs caused by non-commutativity (confirmed by the programmers).

The reducer commutativity problem in general is undecidable. However, in practice, MapReduce programs are usually used for data analytics and have very simple control structures. Many of them just iterate through the input list and compute the output with very simple operations. We want to study if the commutativity problem of real-world reducers is decidable. It has been shown in [3] that even with a simple programming language where the only loop structure allowed is to go over the input list once, the commutativity problem is already undecidable. Under scrutiny, we found that the language is still too expressive for typical data analytics programs. For example, it allows arbitrary multiplications of variables, which is a key element in the undecidability proof.

Contributions. By observing the behavioral patterns of reducer programs for data analytics, we first design a programming language for reducers to characterize the essential features of them. We found that the commutativity problem becomes decidable if we partition variables into *control variables* and *data variables*. Control variables can occur in transition guards, but can only store values directly from the input list (e.g., it is not allowed to store the sum of two input values in a control variable). On the other hand, data variables are used to aggregate some information for outputs (e.g. sum of the values from the input list), but cannot be used in transition guards. This distinction is inspired by the streaming transducer model [1], which, we believe, provides good insights for reducer programming language design in the MapReduce framework. Moreover, we assume that there are no nested loops in the language for reducers, which is a typical situation for MapReduce programs in practice.

We then introduce a formalism called *streaming numerical transducers (SNT)* and obtain a decision procedure for the commutativity problem of the aforementioned language for reducers. Similar to the language for reducers, SNTs distinguish between control variables and data variables. Although conceptually SNTs are similar to streaming transducers over data words introduced in [1], they are intrinsically different in the following sense: The outputs of SNTs are

integers and the integer variables therein are manipulated by linear arithmetic operations. On the other hand, the outputs of streaming transducers are data words, and the data word variables are manipulated by concatenation operations. SNTs in this paper are assumed to be *generalized flat*, which generalizes the "flat" automata (c.f. [11]) in the sense that each nontrivial strongly connected component (SCC) of the transition graph is a collection of cycles, instead of one single cycle. Generalized flat transition graphs are sufficient to capture the transition structures of the programs in the aforementioned language for reducers.

The decision procedure for the commutativity problem is obtained by reducing to the equivalence problem of SNTs, which is further reduced to the non-zero output problem. The non-zero output problem asks whether given an SNT, there exists some input data word w and initial valuation of variables such that the output of the SNT on w is defined and non-zero. For the non-zero output problem of SNTs, we apply a nontrivial combinatorial analysis of the evolvement of the integer variables during the runs of SNTs (Sect. 5.1). The key idea of the decision procedure is that, generally speaking, if only the non-zero output problem is concerned, the different cycles in the SCCs can be dealt with *independently* (Sects. 5.2 and 5.3). As a further evidence of the usefulness of SNTs for MapReduce programs, we demonstrate that SNTs can be composed to model and analyze the reducer programs that read the input list multiple times (Sect. 6).

As a novel formalism over infinite alphabets, the model of SNTs is interesting in its own right: On the one hand, SNTs are expressive in the sense that they include linear arithmetic operations on integer variables, while at the same time admit rather general transition graphs, that is, generalized flat transition graphs. On the other hand, despite this strong expressibility, it turns out that the commutativity problem, the equivalence problem, and the non-zero output problem of SNTs are still decidable.

Related Work. SNTs can be seen as generalizations of register automata [10,14] where registers correspond to the control variables in our terminology. Although register automata can have very general transition graphs beyond the generalized flat ones, they do not allow arithmetic operations on the variables. There have been many automata models that contain arithmetic operations. Counter automata contain counters whose values can be updated by arithmetic operations (see [5–7,9,11], to cite a few) in each transition. Intuitively, the major difference between SNTs and counter automata is that SNTs work on data words and can apply arithmetic operations to an unbounded number of independent integer values, whereas counter automata contain a bounded number of counters which involve only a bounded number of integer values in one configuration. Cost register automata (CRA) [2] also contain arithmetic operations, where the costs are stored into registers for which arithmetic operations can be applied. The equivalence of CRAs with the addition operation is decidable. SNTs are different from CRAs since the inputs of CRAs are words on finite alphabets, while those of SNTs are data words. Moreover, SNTs allow guards over variables ranging over an infinite domain but CRAs do not. There have been several transducer models

on data words: Streaming transducers [1] mentioned before and symbolic transducers [17]. Symbolic transducers have data words as both inputs and outputs. They can put guards on the input value in one position of data words, but are incapable of comparing and aggregating multiple input values in different positions. In [13], the authors considered a model for reducers in the MapReduce framework where the only comparison that can be performed between data values are equalities, and the reducers are essentially register automata/transducers. Their model can describe a system with multiple layers of mappers and reducers.

The rest of the paper is organized as follows. Section 2 defines the notations used in this paper. Section 3 describes our design of the programming language for reducers. Section 4 defines SNTs. Section 5 describes the decision procedure of SNTs. Section 6 discusses how to use our approach to analyze the commutativity property of more challenging data analytics programs. We conclude this work in Sect. 7. The missing technical details and proofs can be found in the full version of this paper [4].

2 Preliminaries

Let $\mathbb{Z}$, $\mathbb{Z}^{\neq 0}$ be the set of integers, non-zero integers, respectively. We assume that all variables range over $\mathbb{Z}$. For a function f, let $\mathsf{dom}(f)$ and $\mathsf{rng}(f)$ denote the *domain* and *range* of f, respectively.

An *expression* e over the set of variables Z is defined by the following rules, $e ::= c \mid cz \mid (e + e) \mid (e - e)$, where $z \in Z$ and $c \in \mathbb{Z}$. As a result of the commutativity and associativity of $+$, without loss of generality, we assume that all expressions e in this paper are of the form $c_0 + c_1 z_1 + \cdots + c_n z_n$, where $c_0, c_1, \ldots, c_n \in \mathbb{Z}$ and $z_1, \ldots, z_n \in Z$. For an expression $e = c_0 + c_1 z_1 + \cdots + c_n z_n$, let $\mathsf{vars}(e)$ denote the set of variables z_i such that $c_i \neq 0$. Let $\mathcal{E}_Z$ denote the set of all expressions over the set of variables Z. In this paper, it is assumed that all the constants in the expressions are encoded in binary.

A *valuation* ρ of Z is a function from Z to $\mathbb{Z}$. A *symbolic valuation* Ω of Z is a function that maps a variable in Z to an expression (possibly over a different set of variables). The value of e under a valuation ρ (resp. symbolic valuation Ω), denoted by $[\![e]\!]_\rho$ (resp. $[\![e]\!]_\Omega$), is defined recursively in the standard way. For example, let Ω be a symbolic valuation the maps z_1 to $z_1 + z_2$ and z_2 to $3z_2$, then $[\![2z_1 + z_2]\!]_\Omega = 2[\![z_1]\!]_\Omega + [\![z_2]\!]_\Omega = 2(z_1 + z_2) + 3z_2 = 2z_1 + 5z_2$. For a valuation ρ, a variable z, and $c \in \mathbb{Z}$, define the valuation $\rho[c/z]$ such that $\rho[c/z](z) = c$ and $\rho[c/z](z') = \rho(z')$ for $z' \neq z$.

In this paper, we use X and Y to denote the sets of *control variables* and *data variables*, respectively. We use the variable $\mathsf{cur} \notin X \cup Y$ to store the data value that is currently being processed in the input list and use X^+ to denote the set $X \cup \{\mathsf{cur}\}$. A *guard* is a formula either of type 1 defined by the rules $g ::= \mathsf{true} \mid \mathsf{cur} \leq x \mid \mathsf{cur} > x \mid g \wedge g$, or of type 2 defined by the rules $g ::= \mathsf{true} \mid \mathsf{cur} \geq x \mid \mathsf{cur} < x \mid g \wedge g$, where $x \in X$, and $c \in \mathbb{Z}$. Note that the guards defined here are *equality-free* in the sense that for each guard g, no equalities between the variables in X^+ can be inferred from g. Let ρ be a

valuation of X^+ and g be a guard. Then ρ satisfies g, denoted by $\rho \models g$, iff g is evaluated to true under ρ. Let $[n]$ denote the set $\{1, 2, \ldots, n\}$, and $[a, b]$ denote the set $\{a, a+1, \ldots, b\}$ when $b \geq a$ and $\emptyset$ otherwise. A *permutation* on $[n]$ is a bijection from $[n]$ to $[n]$. The set of permutations on $[n]$ is denoted by S_n.

A *data word* w is a sequence of integer values $d_1 \ldots d_n$ such that $d_i \in \mathbb{Z}$ for each i. We use $\mathsf{hd}(w)$, $\mathsf{tl}(w)$, and $|w|$ to denote the data value d_1, the tail $d_2 \ldots d_n$, and the length n, respectively. We use ϵ to denote an empty data word. As a convention, we let $\mathsf{hd}(\epsilon) = \bot$, $\mathsf{tl}(\epsilon) = \bot$, and $|\epsilon| = 0$. Given two data words w, w', we use $w.w'$ to denote their concatenation. Given $\sigma \in S_n$, we lift σ to data words by defining $\sigma(w) = d_{\sigma(1)} \ldots d_{\sigma(n)}$, for each data word $w = d_1 \ldots d_n$. We call $\sigma(w)$ as a permutation of w.

3 Language for Integer Reducers

We discuss the rationale behind the design of the programming language for reducers such that the commutativity problem is decidable. The language intends to support the following typical behavior pattern of reducers: A reducer program iterates through the input data word once, aggregates intermediate information into variables, and produces an output when it stops. Later in Sect. 6, we will show an extension that allows resetting the iterators so that an input data word can be traversed multiple times.

$$s \in \mathit{Statements} ::= y := e; \mid y \mathrel{+}= e; \mid x := x'; \mid s\ s \mid \mathsf{next}; \mid \text{if } (g)\{s\} \text{ [else } \{s\}]$$
$$p \in \mathit{Programs} ::= \mathsf{loop}\{s\ \mathsf{next};\}\mathsf{ret}\ r; \mid s\ \mathsf{next}; p$$

Fig. 1. A simple programming language for reducers. Here $x \in X$ are control variables, $y \in Y$ are data variables, $x' \in X^+$, $e \in \mathcal{E}_{X^+}$ are expressions, and r is an expression in $\mathcal{E}_{X \cup Y}$. The square brackets mean that the else branch is optional.

More concretely, we focus on the programming language in Fig. 1. The language includes the usual features of program languages, variable assignments, sequential compositions, and conditional branchings. It also includes a statement next; which is used to advance the data word iterator. The loop$s$ next; statement repeatedly executes the loop body s next; until reaching the end of input data word. The novel feature of the language is that we partition the variables into two sets: *control variables* X and *data variables* Y. The variables from X are used for guiding the control flow and the variables from Y are used for storing aggregated intermediate data values. The variables from X can store only either initial values of variables in X or values occurring in the input data word. They can occur both in guards g or arithmetic expressions e. On the other hand, the variables from Y can aggregate the results obtained from arithmetic expressions e, but cannot occur in guards g or arithmetic expressions e. The initial values of variables can be arbitrary. Given a program p, a data word w, and a valuation

ρ_0, we use $p_{\rho_0}(w)$ to denote the output of p on w, with the initial values of variables given by ρ_0. The formal semantics of the language can be found in the full version [4].

In this paper, we assume that the reducer programs p satisfy that *all the guards between two consecutive* `next` *statements are mono-typed*, more specifically, for each execution path in the control flow graph of p and each pair of consecutive `next` statements, either all the guards g of the branching statements between them are of type 1, or all the guards between them are of type 2 (cf. Sect. 2 for the definition of guards). In addition, to simplify the presentation, we assume that the reducer programs p are *transition-enabled* in the following sense, for each execution path in the control flow graph of p, there is an input w and initial valuations of variables ρ_0 so that the run of p over w and ρ_0 follows the execution path.

Note that we do not allow multiplications in the language, so the reduction from the Diophantine equations in [3] no longer works. Even though, if we do not distinguish the control and data variables, we can show easily that commutativity problem for this language is still undecidable, by a reduction from the reachability problem of Petri nets with inhibitor arcs [12,15]. The reachability problem of Petri nets with inhibitor arcs is reduced to the reachability problem of the reducer programs, which is in turn easily reduced to the commutativity problem of reducer programs.

Notice that in the programming language, we only allow additions (+=) or assignments (:=) of a new value computed from an expression over X^+ to data variables. In Fig. 2 we demonstrate a few examples performing data analytics operations. Observe that all of them follow the same behavioral pattern: The program iterates through the input data word and aggregates some intermediate information into some variables. The operations used for the aggregation are usually rather simple: either a new value is added to the variable (e.g. `sum` and `cnt` in Fig. 2) storing the aggregated information, or a new value is assigned to the variable (e.g. `max` and `2nd_largest` in Fig. 2). Actually, the similar behavioral pattern occurs in all programs we have investigated. Still, one may argue that allowing only additions and subtractions is too restrictive for data analytics.

```
max{
 max:=cur;
 next;
 loop{
  if (cur>max)
    {max:=cur;}
  next;
 }
 ret max;}
```

```
sum{
 sum:=cur;next;
 loop{sum+=cur;next;}
 ret sum;}
```

```
cnt{
 cnt:=0;next;
 loop{cnt+=1;next;}
 ret cnt;}
```

```
2nd_largest {
 a:=cur;b:=cur;next;
 if(cur>a){a:=cur;}
 else {b:=cur;}next;
 loop{
  if (cur>a)
    {b:=a;a:=cur;}
  else
    {if(cur>b){b:=cur;}}
  next;}
 ret b;}
```

Fig. 2. Examples of reducers performing data analytics operations

In Sect. 6, we will discuss the extensions of the language to support more challenging examples, such as *Mean Absolute Deviation* and *Standard Deviation*.

We focus on the following problems of reducer programs: (1) *Commutativity*: given a program p, decide whether for each data word w and its permutation w', it holds that $p_{\rho_0}(w) = p_{\rho_0}(w')$ for all initial valuations ρ_0. (2) *Equivalence*: given two programs p and p', decide whether for each data word w and each initial valuation ρ_0, it holds that $p_{\rho_0}(w) = p'_{\rho_0}(w)$.

4 Streaming Numerical Transducers

In this section, we introduce *streaming numerical transducers* (SNTs), whose inputs are data words and outputs are integer values. A SNT scans a data word from left to right, records and aggregates information using control and data variables, and outputs an integer value when it finishes reading the data word. We will use SNTs to decide the commutativity and equivalence problem of the reducer programs defined in Sect. 3.

A SNT $\mathcal{S}$ is a tuple $(Q, X, Y, \delta, q_0, O)$, where Q is a finite set of states, X is a finite set of control variables to store data values that have been met, Y is a finite set of data variables to aggregate information for the output, δ is the set of transitions, $q_0 \in Q$ is the initial state, O is the output function, which is a partial function from Q to $\mathcal{E}_{X \cup Y}$. The set of transitions δ comprises the tuples (q, g, η, q'), where $q, q' \in Q$, g is a guard over X^+ (defined in Sect. 2), and η is an assignment which is a partial function mapping $X \cup Y$ to $\mathcal{E}_{X^+ \cup Y}$ such that for each $x \in \mathsf{dom}(\eta) \cap X$, $\eta(x) = x'$ for some $x' \in X^+$. We write $q \xrightarrow{(g,\eta)} q'$ to denote $(q, g, \eta, q') \in \delta$ for convenience. We would like to remark that the guards in the transitions can be of both types, that is, of type 1 or type 2.

Moreover, we assume that an SNT $\mathcal{S}$ satisfies the following constraints. (1) *Deterministic*: For each pair of distinct transitions originating from q, say (q, g_1, η_1, q'_1) and (q, g_2, η_2, q'_2), it holds that $g_1 \wedge g_2$ is unsatisfiable. (2) *Generalized flat*: Each SCC (strongly connected component) of the transition graph of $\mathcal{S}$ is either a single state or a set of simple cycles $\{C_1, \ldots, C_n\}$ which contains a state q such that for each $i, j : 1 \leq i < j \leq n$, q is the *only* state shared by C_i and C_j. (3) *Independently evolving and copyless*: For each $(q, g, \eta, q') \in \delta$ and for each $y \in \mathsf{dom}(\eta) \cap Y$, $\eta(y) = e$ or $\eta(y) = y + e$ for some expression e over X^+.

The semantics of an SNT $\mathcal{S}$ is defined as follows. A *configuration* of $\mathcal{S}$ is a pair (q, ρ), where $q \in Q$ and ρ is a valuation of $X \cup Y$. An *initial* configuration of $\mathcal{S}$ is (q_0, ρ_0), where ρ_0 assigns arbitrary values to the variables from $X \cup Y$. A sequence of configurations $(q_0, \rho_0)(q_1, \rho_1) \ldots (q_n, \rho_n)$ is a *run* of $\mathcal{S}$ over a data word $w = d_1 \ldots d_n$ iff there exists a path (sequence of transitions) $P = q_0 \xrightarrow{(g_1,\eta_1)} q_1 \xrightarrow{(g_2,\eta_2)} q_2 \ldots q_{n-1} \xrightarrow{(g_n,\eta_n)} q_n$ such that for each $i \in [n]$, $\rho_{i-1}[d_i/\mathsf{cur}] \models g_i$, and ρ_i is obtained from ρ_{i-1} as follows: (1) For each $x \in X$, if $\eta_i(x) = \mathsf{cur}$ then $\rho_i(x) = d_i$, otherwise, if $\eta_i(x) = x' \in X$ then $\rho_i(x) = \rho_{i-1}(x')$, otherwise $\rho_i(x) = \rho_{i-1}(x)$. (2) For each $y \in Y$, if $y \in \mathsf{dom}(\eta_i)$, then $\rho_i(y) = [\![\eta_i(y)]\!]_{\rho_{i-1}[d_i/\mathsf{cur}]}$, otherwise, $\rho_i(y) = \rho_{i-1}(y)$. We call (q_n, ρ_n) the *final configuration* of the run.

In this case, we also say that the run follows the path P. We say that a path P in $\mathcal{S}$ is *feasible* iff there exists a run of $\mathcal{S}$ following P. An SNT $\mathcal{S}$ is said to be *transition-enabled* if each path in $\mathcal{S}$ is feasible. We assume that all SNTs considered in this paper are transition enabled.

Given a data word $w = d_1 \dots d_n$ and an initial configuration (q_0, ρ_0), if there is a run of $\mathcal{S}$ over w starting from (q_0, ρ_0) and with the final configuration (q_n, ρ_n), then the output of $\mathcal{S}$ over w w.r.t. ρ_0, denoted by $\mathcal{S}_{\rho_0}(w)$, is $[\![O(q_n)]\!]_{\rho_n}$. Otherwise, $\mathcal{S}_{\rho_0}(w)$ is undefined, denoted by $\perp$.

Example 1 (SNT for Max). The SNT $\mathcal{S}_{\max}$ for computing the maximum value of an input data word is defined as $(\{q_0, q_1\}, \{\mathsf{max}\}, \emptyset, \delta, q_0, O)$, where the set of transitions δ and the output function O are illustrated in Fig. 3 (here $X = \{\mathsf{max}\}$, $Y = \emptyset$, and $\mathsf{max} := \mathsf{cur}$ denotes the assignment of cur to the variable max).

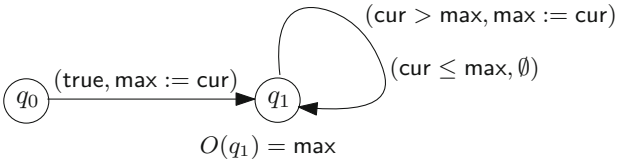

Fig. 3. The SNT $\mathcal{S}_{\max}$ for computing the maximum value

Proposition 1. *For each reducer program p, one can construct an equivalent SNT $\mathcal{S}$ where the number of states and the maximum number of simple cycles in an SCC of the transition graph are at most exponential in the number of branching statements in p.*

Intuitively, the exponential blow-up in the construction is due to the following difference between reducer programs p and SNTs $\mathcal{S}$: A reducer program moves to the next value of an input data word only when a next statement is executed, while an SNT advances the iterator in each transition. Therefore, a sequence of statements with k branching points between each pair of consecutive next statements in the control flow of p correspond to at most 2^k transitions of $\mathcal{S}$.

We focus on three decision problems of SNTs: (1) *Commutativity*: Given an SNT $\mathcal{S}$, decide whether $\mathcal{S}$ is commutative, that is, whether for each data word w and each permutation w' of w, $\mathcal{S}_{\rho_0}(w) = \mathcal{S}_{\rho_0}(w')$ for all initial valuations ρ_0. (2) *Equivalence*: Given two SNTs $\mathcal{S}, \mathcal{S}'$, decide whether $\mathcal{S}$ and $\mathcal{S}'$ are equivalent, that is, whether over each data word w, $\mathcal{S}_{\rho_0}(w) = \mathcal{S}'_{\rho_0}(w)$ for all initial valuations ρ_0. (3) *Non-zero output*: Given an SNT $\mathcal{S}$, decide whether $\mathcal{S}$ has a non-zero output, that is, whether there is a data word w and an initial valuation ρ_0 such that $\mathcal{S}_{\rho_0}(w) \notin \{\perp, 0\}$.

We first observe that the commutativity problem can be reduced to the equivalence problem of SNTs, which can be further reduced to the non-zero

output problem of SNTs. For analyzing the complexity of the decision procedure in the next section, we state the complexity of the reductions w.r.t. the following factors of SNTs: the number of states, the number of control variables (resp. data variables), and the maximum number of simple cycles in an SCC of the transition graph. We will adopt the convention that if after a reduction, some factor becomes exponential, then this fact will be stated explicitly, and on the other hand, if some factor is still polynomial after the reduction, then this fact will be made implicit and will not be stated explicitly.

Proposition 2. *The commutativity problem of SNTs is reduced to the equivalence problem of SNTs in polynomial time.*

We briefly describe the idea of the reduction in Proposition 2 here. Suppose that $\mathcal{S} = (Q, X, Y, \delta, q_0, O)$ is an SNT such that $X = \{x_1, \ldots, x_k\}$ and $Y = \{y_1, \ldots, y_l\}$. Without loss of generality, we assume that the output of $\mathcal{S}$ is defined only for data words of length at least two. We will construct two SNTs $\mathcal{S}_1$ and $\mathcal{S}_2$ so that $\mathcal{S}$ is commutative iff $\mathcal{S}$ is equivalent to both $\mathcal{S}_1$ and $\mathcal{S}_2$.

- Intuitively, over a data word $w = d_1 d_2 d_3 \ldots d_n$ with $n \geq 2$, $\mathcal{S}_1$ simulates the run of $\mathcal{S}$ over $d_2 d_1 d_3 \ldots d_n$, that is, the data word obtained from w by swapping the first two data values.
- Intuitively, over a data word $w = d_1 d_2 d_3 \ldots d_n$ with $n \geq 2$, $\mathcal{S}_2$ simulates the run of $\mathcal{S}$ over $d_2 d_3 \ldots d_n d_1$, that is, the data word obtained from w by moving the first data value to the end.

The correctness of this reduction follows from the fact that all the permutations of $d_1 \ldots d_n$ can be generated by composing the two aforementioned permutations corresponding to $\mathcal{S}_1$ and $\mathcal{S}_2$ respectively (cf. Proposition 1 in [3]). The construction of $\mathcal{S}_1$ (resp. $\mathcal{S}_2$) from $\mathcal{S}$ is in polynomial time w.r.t. the size of $\mathcal{S}$.

Proposition 3. *From SNTs $\mathcal{S}_1$ and $\mathcal{S}_2$, an SNT $\mathcal{S}_3$ can be constructed in polynomial time such that $(\mathcal{S}_1)_{\rho_0}(w) \neq (\mathcal{S}_2)_{\rho_0}(w)$ for some data word w and valuation ρ_0 iff $(\mathcal{S}_3)_{\rho_0}(w) \notin \{\bot, 0\}$ for some data word w and valuation ρ_0.*

Proposition 3 can be proved by a straightforward product construction.

The lemma below states a property of SNTs, due to the fact that the SNTs are assumed to be transition-enabled and the guards are equality-free (cf. the definition of guards in Sect. 2).

Proposition 4. *Let $\mathcal{S}$ be an SNT and P be a path in $\mathcal{S}$. There is a data word w such that (1) there is a run of $\mathcal{S}$ over w which follows P, (2) no data values occur twice in w.*

5 Decision Procedure for the Non-zero Output Problem

We prove our main result, Theorem 1, by presenting a decision procedure for the non-zero output problem of SNTs. We fix an SNT $\mathcal{S} = (Q, X, Y, \delta, q_0, O)$

such that $X = \{x_1, \ldots, x_k\}$ and $Y = \{y_1, \ldots, y_l\}$. We first define summaries of the computations of $\mathcal{S}$ on paths and cycles in Sect. 5.1, then present a decision procedure for the case that the transition graph of $\mathcal{S}$ is a *generalized lasso* in Sect. 5.2. The transition graph of $\mathcal{S}$ is said to be a generalized lasso if it comprises a handle $H = q_0 \xrightarrow{(g_1,\eta_1)} q_1 \ldots q_{m-1} \xrightarrow{(g_m,\eta_m)} q_m$ and a collection of simple cycles $C_1, \ldots, C_n$ such that q_m is the unique state shared by each pair of distinct cycles from $\{C_1, \ldots, C_n\}$. We extend the procedure to SNTs whose transition graphs are not necessarily generalized lassos in Sect. 5.3.

Theorem 1. *The non-zero output problem of SNTs can be decided in time exponential in the number of data variables and the maximum number of simple cycles in an SCC of transition graphs.*

Corollary 1. *The commutativity problem of reducer programs can be decided in time exponential in the number of data variables, and doubly exponential in the number of branching statements of reducer programs.*

Remark 1. Though the decision procedure for the commutativity problem of reducer programs has a complexity exponential in the number of data variables, and doubly exponential in the number of branching statements, we believe that the decision procedure could still be implemented to automatically analyze the programs in practice, in which these numbers are usually small.

5.1 Summarization of the Computations on Paths and Cycles

Suppose $P = p_0 \xrightarrow{(g_1,\eta_1)} p_1 \ldots p_{n-1} \xrightarrow{(g_n,\eta_n)} p_n$ is a path of $\mathcal{S}$. We assume that the initial values of the control and data variables are represented by a symbolic valuation Ω over $X \cup Y$. We use the variables $\mathfrak{d}_1^{\overline{P}}, \mathfrak{d}_2^{\overline{P}}, \ldots, \mathfrak{d}_{r^{\overline{P}}}^{\overline{P}}$ to denote the data values introduced while traversing P. Notice that according to Proposition 4, one can choose different values for different positions of P. Therefore, for each position of P, a fresh variable is introduced to represent the data value in that position. Thus we have $r^{\overline{P}} = n$. Here we use the superscript $\overline{P}$ to denote the fact that $r^{\overline{P}}$ (resp. $\mathfrak{d}_1^{\overline{P}}, \ldots$) is associated with the path P.

Proposition 5. *Suppose that P is a path and the initial values of $X \cup Y$ are represented by a symbolic valuation Ω. Then the values of $X \cup Y$ after traversing the path P are specified by a symbolic valuation $\Theta^{(P,\Omega)}$ satisfying the following conditions.*

- *The set of indices of X, i.e., $[k]$, is partitioned into $I_{pe}^{\overline{P}}$ and $I_{tr}^{\overline{P}}$, the indices of* persistent *and* transient *control variables, respectively. A control variable is persistent if its value has not been changed while traversing P, otherwise, it is transient.*
- *For each $x_j \in X$ such that $j \in I_{pe}^{\overline{P}}$, $\Theta^{(P,\Omega)}(x_j) = \Omega(x_j)$.*
- *For each $x_j \in X$ such that $j \in I_{tr}^{\overline{P}}$, $\Theta^{(P,\Omega)}(x_j) = \mathfrak{d}_{\pi^{\overline{P}}(j)}^{\overline{P}}$, where $\pi^{\overline{P}} : I_{tr}^{\overline{P}} \to [r^{\overline{P}}]$ is a mapping from the index of a transient control variable to the index of the data value assigned to it.*

– *For each* $y_j \in Y$, $\Theta^{(P,\Omega)}(y_j) = \varepsilon_j^{\overline{P}} + \lambda_j^{\overline{P}}\Omega(y_j) + \sum_{j' \in [k]} \alpha_{j,j'}^{\overline{P}}\Omega(x_{j'}) + \sum_{j'' \in [r^{\overline{P}}]} \beta_{j,j''}^{\overline{P}}\mathfrak{d}_{j''}^{\overline{P}}$, *where* $\varepsilon_j^{\overline{P}}, \lambda_j^{\overline{P}}, \alpha_{j,1}^{\overline{P}}, \ldots, \alpha_{j,k}^{\overline{P}}, \beta_{j,1}^{\overline{P}}, \ldots, \beta_{j,r^{\overline{P}}}^{\overline{P}}$ *are integer constants such that* $\lambda_j^{\overline{P}} \in \{0,1\}$ *(as a result of the "independently evolving and copyless" constraint). It can happen that* $\lambda_j^{\overline{P}} = 0$, *which means that* $\Omega(y_j)$ *is irrelevant to* $\Theta^{(P,\Omega)}(y_j)$. *Similarly for* $\alpha_{j,1}^{\overline{P}} = 0$, *and so on.*

In Proposition 5, the sets $I_{pe}^{\overline{P}}$, $I_{tr}^{\overline{P}}$, the mapping $\pi^{\overline{P}}$, and the constants $\varepsilon_j^{\overline{P}}$, $\lambda_j^{\overline{P}}, \ldots, \beta_{j,r^{\overline{P}}}^{\overline{P}}$ only depend on P and are independent of Ω. In addition, they can be computed in polynomial time from (the transitions in) P. We define $(\pi^{\overline{P}})^{-1}$ as the inverse function of $\pi^{\overline{P}}$, that is, for each $j' \in [r^{\overline{P}}]$, $(\pi^{\overline{P}})^{-1}(j') = \{j \in I_{tr}^{\overline{P}} \mid \pi^{\overline{P}}(j) = j'\}$.

As a corollary of Proposition 5, the following result demonstrates how to summarize the computations of $\mathcal{S}$ on the composition of two paths.

Corollary 2. *Suppose that* P_1 *and* P_2 *are two paths in* $\mathcal{S}$ *such that the last state of* P_1 *is the first state of* P_2. *Moreover, let* $\Theta^{(P_1,\Omega)}$ *(resp.* $\Theta^{(P_2,\Omega)}$*) be the symbolic valuation summarizing the computation of* $\mathcal{S}$ *on* P_1 *(resp.* P_2*). Then the symbolic valuation summarizing the computation of* $\mathcal{S}$ *on* P_1P_2 *is* $\Theta^{(P_2,\ \Theta^{(P_1,\Omega)})}$.

In order to get a better understanding of the relation between $\Theta^{(P_2,\ \Theta^{(P_1,\Omega)})}$ and $(\Theta^{(P_1,\Omega)}, \Theta^{(P_2,\Omega)})$, in the following, for each $y_j \in Y$, we obtain a more explicit form of the expression $\Theta^{(P_2,\ \Theta^{(P_1,\Omega)})}(y_j)$, by unfolding therein the expression $\Theta^{(P_1,\Omega)}$.

$$\Theta^{(P_2,\ \Theta^{(P_1,\Omega)})}(y_j) = \left(\varepsilon_j^{\overline{P_2}} + \lambda_j^{\overline{P_2}}\varepsilon_j^{\overline{P_1}}\right) + \left(\lambda_j^{\overline{P_2}}\lambda_j^{\overline{P_1}}\right)\Omega(y_j) + \sum_{j' \in I_{pe}^{\overline{P_1}}} \left(\alpha_{j,j'}^{\overline{P_2}} + \lambda_j^{\overline{P_2}}\alpha_{j,j'}^{\overline{P_1}}\right)\Omega(x_{j'}) + \sum_{j' \in I_{tr}^{\overline{P_1}}} \left(\lambda_j^{\overline{P_2}}\alpha_{j,j'}^{\overline{P_1}}\right)\Omega(x_{j'}) + \sum_{j' \in \mathsf{rng}(\pi^{\overline{P_1}})} \left(\lambda_j^{\overline{P_2}}\beta_{j,j'}^{\overline{P_1}} + \sum_{j'' \in (\pi^{\overline{P_1}})^{-1}(j')} \alpha_{j,j''}^{\overline{P_2}}\right)\mathfrak{d}_{j'}^{\overline{P_1}} + \sum_{j' \in [r^{\overline{P_1}}] \setminus \mathsf{rng}(\pi^{\overline{P_1}})} \left(\lambda_j^{\overline{P_2}}\beta_{j,j'}^{\overline{P_1}}\right)\mathfrak{d}_{j'}^{\overline{P_1}} + \sum_{j' \in [r^{\overline{P_2}}]} \beta_{j,j'}^{\overline{P_2}}\mathfrak{d}_{j'}^{\overline{P_2}}.$$

In the equation, $j' \in I_{pe}^{\overline{P_1}}$ implies that $x_{j'}$ remains unchanged when traversing P_1, which means the initial value of $x_{j'}$ before traversing P_2 is still $\Omega(x_{j'})$ and therefore we have the item $(\alpha_{j,j'}^{\overline{P_2}})\Omega(x_{j'})$. When $j' \in \mathsf{rng}(\pi^{\overline{P_1}})$, the initial value of $x_{j''}$ for each $j'' \in (\pi^{\overline{P_1}})^{-1}(j')$ before traversing P_2 is $\mathfrak{d}_{j'}^{\overline{P_1}}$ and therefore we have the item $\left(\sum_{j'' \in (\pi^{\overline{P_1}})^{-1}(j')} \alpha_{j,j''}^{\overline{P_2}}\right)\mathfrak{d}_{j'}^{\overline{P_1}}$. For all $j' \in [k] = I_{pe}^{\overline{P_1}} \cup I_{tr}^{\overline{P_1}}$, we have the item $(\lambda_j^{\overline{P_2}}\alpha_{j,j'}^{\overline{P_1}})\Omega(x_{j'})$, i.e. the coefficient of $\Omega(x_{j'})$ in $\Theta^{(P_1,\Omega)}$ multiplied by $\lambda_j^{\overline{P_2}}$. Moreover, for all $j' \in [r^{\overline{P_1}}] = \mathsf{rng}(\pi^{\overline{P_1}}) \cup ([r^{\overline{P_1}}] \setminus \mathsf{rng}(\pi^{\overline{P_1}}))$, we have the item $(\lambda_j^{\overline{P_2}}\beta_{j,j'}^{\overline{P_1}})\mathfrak{d}_{j'}^{\overline{P_1}}$, i.e. the coefficient of $\mathfrak{d}_{j'}^{\overline{P_1}}$ in $\Theta^{(P_1,\Omega)}$ multiplied by $\lambda_j^{\overline{P_2}}$.

In the following, by utilizing Proposition 5 and Corollary 2, for each path C^ℓ which is obtained by iterating a cycle C for ℓ times, we illustrate how $\Theta^{(C^\ell,\Omega)}$ is related to $\Theta^{(C,\Omega)}$ and ℓ. For convenience, we call ℓ a *cycle counter variable*.

Proposition 6. *Suppose that C is a cycle and $P = C^\ell$ such that $\ell \geq 2$. Then the symbolic valuation $\Theta^{(C^\ell,\Omega)}$ to summarize the computation of $\mathcal{S}$ on P is as follows,*

$$\begin{aligned}\Theta^{(C^\ell,\Omega)}(y_j) = {} & \left(1+\lambda_j^{\overline{C}}+\cdots+(\lambda_j^{\overline{C}})^{\ell-1}\right)\varepsilon_j^{\overline{C}} + (\lambda_j^{\overline{C}})^{\ell}\Omega(y_j) + \\ & \sum_{j'\in I_{pe}^{\overline{C}}}\left(1+\lambda_j^{\overline{C}}+\cdots+(\lambda_j^{\overline{C}})^{\ell-1}\right)\alpha_{j,j'}^{\overline{C}}\Omega(x_{j'}) + \sum_{j'\in I_{tr}^{\overline{C}}}(\lambda_j^{\overline{C}})^{\ell-1}\alpha_{j,j'}^{\overline{C}}\Omega(x_{j'}) + \\ & \sum_{j'\in \mathsf{rng}(\pi^{\overline{C}})}\sum_{s\in[\ell-1]}\left(\lambda_j^{\overline{C}}\beta_{j,j'}^{\overline{C}} + \sum_{j''\in(\pi^{\overline{C}})^{-1}(j')}\alpha_{j,j''}^{\overline{C}}\right)(\lambda_j^{\overline{C}})^{\ell-s-1}\mathfrak{d}_{j'}^{\overline{C,s}} + \\ & \sum_{j'\in[r^{\overline{C}}]\setminus\mathsf{rng}(\pi^{\overline{C}})}\sum_{s\in[\ell-1]}\left((\lambda_j^{\overline{C}})^{\ell-s}\beta_{j,j'}^{\overline{C}}\right)\mathfrak{d}_{j'}^{\overline{C,s}} + \sum_{j'\in[r^{\overline{C}}]}\beta_{j,j'}^{\overline{C}}\mathfrak{d}_{j'}^{\overline{C,\ell}},\end{aligned}$$

where the variables $\mathfrak{d}_1^{\overline{C,s}},\ldots,\mathfrak{d}_{r^{\overline{C}}}^{\overline{C,s}}$ for $s\in[\ell]$ represent the data values introduced when traversing C for the s-th time.

From Proposition 6 and the fact that $\lambda_j \in \{0,1\}$, we have the following observation.

– If $\lambda_j^{\overline{C}} = 0$, then

$$\begin{aligned}\Theta^{(C^\ell,\Omega)}(y_j) = {} & \varepsilon_j^{\overline{C}} + \sum_{j'\in I_{pe}^{\overline{C}}}\alpha_{j,j'}^{\overline{C}}\Omega(x_{j'}) + \sum_{j'\in\mathsf{rng}(\pi^{\overline{C}})}\left(\sum_{j''\in(\pi^{\overline{C}})^{-1}(j')}\alpha_{j,j''}^{\overline{C}}\right)\mathfrak{d}_{j'}^{\overline{C,\ell-1}} + \\ & \sum_{j'\in[r^{\overline{C}}]}\beta_{j,j'}^{\overline{C}}\mathfrak{d}_{j'}^{\overline{C,\ell}}.\end{aligned}$$

– If $\lambda_j^{\overline{C}} = 1$, then

$$\begin{aligned}\Theta^{(C^\ell,\Omega)}(y_j) = {} & \ell\varepsilon_j^{\overline{C}} + \Omega(y_j) + \sum_{j'\in I_{pe}^{\overline{C}}}\ell\alpha_{j,j'}^{\overline{C}}\Omega(x_{j'}) + \sum_{j'\in I_{tr}^{\overline{C}}}\alpha_{j,j'}^{\overline{C}}\Omega(x_{j'}) + \\ & \sum_{j'\in\mathsf{rng}(\pi^{\overline{C}})}\sum_{s\in[\ell-1]}\left(\beta_{j,j'}^{\overline{C}} + \sum_{j''\in(\pi^{\overline{C}})^{-1}(j')}\alpha_{j,j''}^{\overline{C}}\right)\mathfrak{d}_{j'}^{\overline{C,s}} + \\ & \sum_{j'\in[r^{\overline{C}}]\setminus\mathsf{rng}(\pi^{\overline{C}})}\sum_{s\in[\ell-1]}\beta_{j,j'}^{\overline{C}}\mathfrak{d}_{j'}^{\overline{C,s}} + \sum_{j'\in[r^{\overline{C}}]}\beta_{j,j'}^{\overline{C}}\mathfrak{d}_{j'}^{\overline{C,\ell}}.\end{aligned}$$

5.2 Decision Procedure for Generalized Lassos

In this section, we present a decision procedure for SNTs whose transition graphs are generalized lassos. From Proposition 6, we know that the coefficients containing the cycle counter variable ℓ in $\Theta^{(C^\ell,\Omega)}(y_j)$ can be non-zero when $\lambda_j^{\overline{C}} = 1$. The non-zero coefficients may propagate to the output expression. In such a case,

because the SNTs are "transition-enabled" (i.e. for any sequence of transitions, a corresponding run exists), intuitively, one can pick a run corresponding to a very large ℓ so that it dominates the value of the output expression and makes the output non-zero. In the decision procedure we are going to present, we first check if the handle of the generalized lasso produces a non-zero output in Step I. We then check in Step II the coefficients containing ℓ in the output expression is non-zero. If this does not happen, then we show in Step III that the non-zero output problem of SNT can be reduced to a finite state reachability problem and thus can be easily decided.

Before presenting the decision procedure, we introduce some notations. Let e be an expression consisting of symbolic values $\Omega(z)$ for $z \in X \cup Y$ and variables $\mathfrak{d}_1, \ldots, \mathfrak{d}_s$ corresponding to the values of the input data word. More specifically, let $e := \mu_0 + \mu_1\Omega(z_1) + \cdots + \mu_{k+l}\Omega(z_{k+l}) + \xi_1\mathfrak{d}_1 + \cdots + \xi_s\mathfrak{d}_s$, such that $\mu_0, \mu_1, \ldots, \mu_{k+l}, \xi_1, \ldots, \xi_s$ are expressions containing only constants and cycle counter variables. Then we call μ_0 the *constant atom*, $\mu_i\Omega(z_i)$ the $\Omega(z_i)$-atom for $i \in [k+l]$, and $\xi_j\mathfrak{d}_j$ the $\mathfrak{d}_j$-atom for $j \in [s]$ of the expression e. Moreover, $\mu_1, \ldots, \mu_{k+l}, \xi_1, \ldots, \xi_s$ are called the *coefficients* and $\Omega(z_1), \ldots, \Omega(z_{k+l}), \mathfrak{d}_1 \ldots, \mathfrak{d}_s$ the *subjects* of these atoms. A non-constant atom is said to be *nontrivial* if its coefficient is *not* identical to zero.

In the rest of this subsection, we assume that the transition graph of $\mathcal{S}$ comprises a handle $H = q_0 \xrightarrow{(g_1,\eta_1)} q_1 \ldots q_{m-1} \xrightarrow{(g_m,\eta_m)} q_m$ and a collection of simple cycles $C_1, \ldots, C_n$ such that q_m is the unique state shared by each pair of distinct cycles from $\{C_1, \ldots, C_n\}$. Moreover, without loss of generality, we assume that $O(q_m) = a_0 + a_1x_1 + \cdots + a_kx_k + b_1y_1 + \cdots + b_ly_l$, and $O(q)$ is undefined for all the other states q.

A *cycle scheme* $\mathfrak{s}$ is a path $C_{i_1}^{\ell_1}C_{i_2}^{\ell_2}\ldots C_{i_t}^{\ell_t}$ such that $i_1, \ldots, i_t \in [n], \ell_1, \ldots, \ell_t \geq 1$, and for each $j \in [t-1]$, $i_j \neq i_{j+1}$. Intuitively, $\mathfrak{s}$ is a path obtained by first iterating C_{i_1} for ℓ_1 times, then C_{i_2} for ℓ_2 times, and so on. From Proposition 6 and Corollary 2, a symbolic valuation $\Theta^{(\mathfrak{s},\Omega)}$ can be constructed to summarize the computation of $\mathcal{S}$ on $\mathfrak{s}$.

Lemma 1. *Suppose* $\mathfrak{s} = C_{i_1}^{\ell_1}C_{i_2}^{\ell_2}\ldots C_{i_t}^{\ell_t}$ *is a cycle scheme, and* Ω *is a symbolic valuation representing the initial values of the control and data variables. For all* $j' \in I_{pe}^{\overline{C_{i_1}}}$, *let* $r_{j'}$ *be the largest number* $r \in [t]$ *such that* $j' \in \bigcap_{s\in[r]} I_{pe}^{\overline{C_{i_s}}}$, *i.e.,* $x_{j'}$ *remains persistent when traversing* $C_{i_1}^{\ell_1}C_{i_2}^{\ell_2}\ldots C_{i_{r_{j'}}}^{\ell_{r_{j'}}}$. *Then for each* $j \in [l]$ *and* $j' \in I_{pe}^{\overline{C_{i_1}}}$, *the coefficient of the* $\Omega(x_{j'})$*-atom in* $\Theta^{(\mathfrak{s},\Omega)}(y_j)$ *is*

$$e + \sum_{s_1\in[r_{j'}]} \left(1 + \lambda_j^{\overline{C_{i_{s_1}}}} + \cdots + (\lambda_j^{\overline{C_{i_{s_1}}}})^{\ell_{s_1}-1}\right) \alpha_{j,j'}^{\overline{C_{i_{s_1}}}} \prod_{s_2\in[s_1+1,t]} \left(\lambda_j^{\overline{C_{i_{s_2}}}}\right)^{\ell_{s_2}},$$

where (1) $e=0$ *when* $r_{j'}=t$ *and (2)* $e = (\lambda_j^{\overline{C_{i_s}}})^{\ell_s-1}\alpha_{j,j'}^{\overline{C_{i_s}}} \prod_{s'\in[s+1,t]} \left(\lambda_j^{\overline{C_{i_{s'}}}}\right)^{\ell_{s'}}$ *with* $s = r_{j'} + 1$ *when* $r_{j'} < t$.

The constant atom of $\Theta^{(\mathfrak{s},\Omega)}(y_j)$ *is*

$$\sum_{s_1\in[t]}\left(1+\lambda_j^{\overline{C_{i_{s_1}}}}+\cdots+(\lambda_j^{\overline{C_{i_{s_1}}}})^{\ell_{s_1}-1}\right)\varepsilon_j^{\overline{C_{i_{s_1}}}}\prod_{s_2\in[s_1+1,t]}\left(\lambda_j^{\overline{C_{i_{s_2}}}}\right)^{\ell_{s_2}}$$

Moreover, for all $j\in[l]$*, in* $\Theta^{(\mathfrak{s},\Omega)}(y_j)$*, only the constant atom and the coefficients of the* $\Omega(x_{j'})$*-atoms with* $j'\in I_{pe}^{\overline{C_{i_1}}}$ *contain a subexpression of the form* $\mu_{\mathfrak{s}}\ell_1$ *for some* $\mu_{\mathfrak{s}}\in\mathbb{Z}$*.*

Notice that above, $\lambda_j^{\overline{C_{i_{s_1}}}}\in\{0,1\}$ for $j\in[l]$ and $s_1\in[t]$. Hence the value of $(1+\lambda_j^{\overline{C_{i_{s_1}}}}+\cdots+(\lambda_j^{\overline{C_{i_{s_1}}}})^{\ell_{s_1}-1})$ can only be 1 or ℓ_{s_1} and $\left(\lambda_j^{\overline{C_{i_{s_2}}}}\right)^{\ell_{s_2}}\in\{0,1\}$. Therefore, both the constant atom and the coefficient of the $\Omega(x_{j'})$-atom with $j'\in I_{pe}^{\overline{C_{i_1}}}$ can be rewritten to the form of $c_0+c_1\ell_1+c_2\ell_2+\cdots+c_t\ell_t$ for $c_0\ldots c_t\in\mathbb{Z}$. Note that some of $c_0\ldots c_t$ might be zero.

Step I: We are ready to present the decision procedure. At first, we observe that after traversing H with the initial values of the variables given by some valuation Ω_0, for each $j'\in I_{tr}^{\overline{H}}$, the value of the control variable $x_{j'}$ becomes $\mathfrak{d}_{\pi^{\overline{H}}(j')}^{\overline{H}}$, more formally, $\Theta^{(H,\Omega_0)}(x_{j'})=\mathfrak{d}_{\pi^{\overline{H}}(j')}^{\overline{H}}$.

In Step I, we check if $[\![O(q_m)]\!]_{\Theta^{(H,\Omega_0)}}$ is not identical to zero. This can be done by checking if the constant-atom or the coefficient of some non-constant atom of the output expression $[\![O(q_m)]\!]_{\Theta^{(H,\Omega_0)}}$ is not identical to zero.

Step I. Decide whether $[\![O(q_m)]\!]_{\Theta^{(H,\Omega_0)}}$ is not identical to zero. If the answer is yes, then the decision procedure terminates and returns the answer true. Otherwise, go to Step II.

Complexity Analysis of Step I. Since $\Theta^{(H,\Omega_0)}$ can be computed in polynomial time from H, it follows that Step I can be done in polynomial time.

Step II: The goal of Step II is either showing that in $f=[\![O(q_m)]\!]_{\Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})}}$, all subexpressions containing the cycle counter variables are identical to zero and hence can be ignored or showing that f is not identical to zero. Let $\mathfrak{s}=C_{i_1}^{\ell_1}C_{i_2}^{\ell_2}\ldots C_{i_t}^{\ell_t}$ be a cycle scheme. From Lemma 1, for each $j'\in I_{pe}^{\overline{C_{i_1}}}$ and symbolic valuation Ω, the only subexpression containing ℓ_1 in the coefficient of $\Omega(x_{j'})$-atom of $[\![O(q_m)]\!]_{\Theta^{(\mathfrak{s},\Omega)}}$ is

$$\sum_{1\le j\le l}b_j\left((\lambda_j^{\overline{C_{i_2}}})^{\ell_2}\ldots(\lambda_j^{\overline{C_{i_t}}})^{\ell_t}\right)\left(1+\lambda_j^{\overline{C_{i_1}}}+\cdots+(\lambda_j^{\overline{C_{i_1}}})^{\ell_1-1}\right)\alpha_{j,j'}^{\overline{C_{i_1}}}.\qquad(*)$$

Since $\lambda_j^{\overline{C_{i_1}}}, \lambda_j^{\overline{C_{i_2}}}, \ldots, \lambda_j^{\overline{C_{i_t}}} \in \{0,1\}$, the expression $(*)$ can be rewritten as $\mu_{\mathfrak{s},(i_1,j')}\ell_1 + \nu_{\mathfrak{s},(i_1,j')}$ for some integer constants $\mu_{\mathfrak{s},(i_1,j')}$ and $\nu_{\mathfrak{s},(i_1,j')}$.

The only subexpression containing ℓ_1 in the constant atom of $[\![O(q_m)]\!]_{\Theta^{(\mathfrak{s},\Omega)}}$ is

$$\sum_{1\le j\le l} b_j \left((\lambda_j^{\overline{C_{i_2}}})^{\ell_2} \ldots (\lambda_j^{\overline{C_{i_t}}})^{\ell_t}\right)\left(1 + \lambda_j^{\overline{C_{i_1}}} + \cdots + (\lambda_j^{\overline{C_{i_1}}})^{\ell_1 - 1}\right)\varepsilon_j^{\overline{C_{i_1}}}. \qquad (**)$$

The expression $(**)$ can be rewritten as $\mu_{\mathfrak{s},(i_1,0)}\ell_1 + \nu_{\mathfrak{s},(i_1,0)}$ for some integer constants $\mu_{\mathfrak{s},(i_1,0)}$ and $\nu_{\mathfrak{s},(i_1,0)}$. If $\mu_{\mathfrak{s},(i_1,0)} = 0$ and $\mu_{\mathfrak{s},(i_1,j')} = 0$ for all $j' \in I_{pe}^{\overline{C_{i_1}}}$, then we can ignore all subexpressions containing the cycle counter variable ℓ_1 in $[\![O(q_m)]\!]_{\Theta^{(\mathfrak{s},\Omega)}}$, i.e., the subexpressions $\mu_{\mathfrak{s},(i_1,0)}\ell_1$ and $\mu_{\mathfrak{s},(i_1,j')}\ell_1$ for all $j' \in I_{pe}^{\overline{C_{i_1}}}$.

Step II. For each $i_1 \in [n]$, check all cycle scheme $\mathfrak{s} = C_{i_1}^{\ell_1} C_{i_2} \ldots C_{i_t}$ such that $i_2, \ldots, i_t$ are mutually distinct. There are only finitely many this kind of cycle schemes. If one of the following constraints is satisfied, then return true.

(1) There is $j' \in I_{pe}^{\overline{C_{i_1}}}$ such that $\mu_{\mathfrak{s},(i_1,j')} \neq 0$. (2) $\mu_{\mathfrak{s},(i_1,0)} \neq 0$.

If the decision procedure has not returned yet, then go to Step III.

Complexity analysis of Step II. Since $i_1, \ldots, i_t$ are mutually distinct, the number of cycle schemes $\mathfrak{s} = C_{i_1}^{\ell_1} C_{i_2} \ldots C_{i_t}$ in Step II is exponential in the number of cycles in the generalized lasso. Once the cycle scheme is fixed, the two constraints in Step II can be decided in polynomial time. Therefore, the complexity of Step II is exponential in the number of cycles in the generalized lasso.

If there exists $j' \in I_{pe}^{\overline{C_{i_1}}}$ such that $\mu_{\mathfrak{s},(i_1,j')} \neq 0$, then $[\![O(q_m)]\!]_{\Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})}}$ contains some nontrivial non-constant atom for $\mathfrak{s} = C_{i_1}^{\ell_1} C_{i_2} \ldots C_{i_t}$ and some $\ell_1 = s$. The guards in the path $C_{i_1}^{s} C_{i_2} \ldots C_{i_t}$ enforce a preorder over the subjects of those nontrivial non-constant atoms. Pick one of the nontrivial non-constant atoms with a maximal subject w.r.t. the preorder. Since the subject is maximal, it can be assigned an arbitrarily large number so that the corresponding atom dominates $[\![O(q_m)]\!]_{\Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})}}$. This is sufficient to make $[\![O(q_m)]\!]_{\Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})}}$ non-zero. Otherwise, if $[\![O(q_m)]\!]_{\Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})}}$ contains some other nontrivial non-constant atoms, then we can apply a similar argument as above and conclude that $[\![O(q_m)]\!]_{\Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})}}$ can be made non-zero. On the other hand, if $[\![O(q_m)]\!]_{\Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})}}$ contains no nontrivial non-constant atoms, but $\mu_{\mathfrak{s},(i_1,0)} \neq 0$, then we can let ℓ_1 arbitrarily large to make the expression $[\![O(q_m)]\!]_{\Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})}}$ non-zero. Therefore, when there is $j' \in I_{pe}^{\overline{C_{i_1}}}$ such that $\mu_{\mathfrak{s},(i_1,j')} \neq 0$, or $\mu_{\mathfrak{s},(i_1,0)} \neq 0$, we are able to conclude that there *must be* an input to make $[\![O(q_m)]\!]_{\Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})}}$ non-zero. Similar arguments can be applied to $\ell_2 \ldots \ell_n$.

If Step II does not return true, we show below that for all cycle schemes $\mathfrak{s}_1 = C_{i_1}^{\ell_1} C_{i_2}^{\ell_2} \ldots C_{i_{s_1}}^{\ell_{s_1}}$ with $i_1, i_2, \ldots, i_{s_1} \in [n]$, all subexpressions containing

cycle counter variables in $[\![O(q_m)]\!]_{\Theta^{(\mathfrak{s},\Omega)}}$ are identical to zero and hence can be removed. Let $i'_2 \dots i'_{s_2}$ be the sequence obtained from $i_2 \dots i_{s_1}$ by keeping just one copy for each duplicated index therein. In Step II we already checked a cycle scheme $\mathfrak{s}_2 = C_{i_1}^{\ell_1} C_{i'_2} \dots C_{i'_{s_2}}$. Step II guarantees that all subexpressions containing ℓ_1 in $[\![O(q_m)]\!]_{\Theta^{(\mathfrak{s}_2,\Omega)}}$ are identical to zero and hence can be removed. Because for all $j \in [l]$, $\lambda_j^{\overline{C_1}}, \dots, \lambda_j^{\overline{C_n}} \in \{0,1\}$, $(\lambda_j^{\overline{C_{i_2}}})^{\ell_2} \dots (\lambda_j^{\overline{C_{i_{s_1}}}})^{\ell_{s_1}} = \lambda_j^{\overline{C_{i'_2}}} \dots \lambda_j^{\overline{C_{i'_{s_2}}}}$. We proved that the $(*)$ and $(**)$ style expressions are equivalent in both $\mathfrak{s}_1$ and $\mathfrak{s}_2$. Hence we can also remove all subexpressions containing ℓ_1 from $[\![O(q_m)]\!]_{\Theta^{(\mathfrak{s}_1,\Omega)}}$, without affecting its value. Those subexpressions containing ℓ_2 can also be removed by considering the cycle scheme $\mathfrak{s}_3 = C_{i_2}^{\ell_2} C_{i''_3} \dots C_{i''_{s_3}}$ and applying a similar reasoning, where the sequence $i''_3 \dots i''_{s_3}$ is obtained from $i_3 \dots i_{s_1}$, similarly to the construction of $i'_2 \dots i'_{s_2}$ from $i_2 \dots i_{s1}$. The same applies to all other cycle counter variables $\ell_3, \dots, \ell_{s_1}$. We use the notation $\Theta^{(\mathfrak{s},\Omega)^-}(y_j)$ to denote the expression obtained by removing from the constant atom and coefficients of the non-constant atoms of $\Theta^{(\mathfrak{s},\Omega)}(y_j)$ all subexpressions containing the cycle counter variables, for all $y_j \in Y$.

Lemma 2. *Suppose that the decision procedure has not returned* true *after Step II. For each cycle scheme* $\mathfrak{s}$*, let* $f = [\![O(q_m)]\!]_{\Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})}}$ *and* $f' = [\![O(q_m)]\!]_{\Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})^-}}$*. For all valuations* ρ*,* $[\![f]\!]_\rho \neq 0$ *iff* $[\![f']\!]_\rho \neq 0$*.*

Step III: For each cycle scheme $\mathfrak{s}$, let

$$
\begin{aligned}
&\Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})^-}(y_j) \\
&= \quad \varepsilon_j^{(\overline{\mathfrak{s}})^-} + \lambda_j^{\overline{\mathfrak{s}}} \Theta^{(H,\Omega_0)}(y_j) + \sum_{j' \in [k]} \alpha_{j,j'}^{(\overline{\mathfrak{s}})^-} \Theta^{(H,\Omega_0)}(x_{j'}) + \sum_{j' \in [r^{\overline{\mathfrak{s}}}]} \beta_{j,j'}^{\overline{\mathfrak{s}}} \mathfrak{d}_{j'}^{\overline{\mathfrak{s}}} \\
&= \quad \left(\varepsilon_j^{(\overline{\mathfrak{s}})^-} + \lambda_j^{\overline{\mathfrak{s}}} \varepsilon_j^{\overline{H}}\right) + \left(\lambda_j^{\overline{\mathfrak{s}}} \lambda_j^{\overline{H}}\right) \Omega_0(y_j) + \sum_{j' \in I_{pe}^{\overline{H}}} \left(\alpha_{j,j'}^{(\overline{\mathfrak{s}})^-} + \lambda_j^{\overline{\mathfrak{s}}} \alpha_{j,j'}^{\overline{H}}\right) \Omega_0(x_{j'}) \\
&\quad + \sum_{j' \in I_{tr}^{\overline{H}}} \left(\lambda_j^{\overline{\mathfrak{s}}} \alpha_{j,j'}^{\overline{H}}\right) \Omega_0(x_{j'}) + \sum_{j' \in \mathsf{rng}(\pi^{\overline{H}})} \left(\lambda_j^{\overline{\mathfrak{s}}} \beta_{j,j'}^{\overline{H}} + \sum_{j'' \in (\pi^{\overline{H}})^{-1}(j')} \alpha_{j,j''}^{(\overline{\mathfrak{s}})^-}\right) \mathfrak{d}_{j'}^{\overline{H}} \\
&\quad + \sum_{j' \in [r^{\overline{H}}] \setminus \mathsf{rng}(\pi^{\overline{H}})} \left(\lambda_j^{\overline{\mathfrak{s}}} \beta_{j,j'}^{\overline{H}}\right) \mathfrak{d}_{j'}^{\overline{H}} + \sum_{j' \in [r^{\overline{\mathfrak{s}}}]} \beta_{j,j'}^{\overline{\mathfrak{s}}} \mathfrak{d}_{j'}^{\overline{\mathfrak{s}}}.
\end{aligned}
$$

Note that the coefficients of the $\mathfrak{d}_1^{\overline{\mathfrak{s}}}$-atom, …, and $\mathfrak{d}_{r^{\overline{\mathfrak{s}}}}^{\overline{\mathfrak{s}}}$-atom in $\Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})^-}(y_j)$ are the same as those in $\Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})}(y_j)$.

We first observe that the coefficients of the atoms in $\Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})^-}(y_j)$ are from a bounded set.

Lemma 3. *Suppose that the decision procedure has not returned yet after Step II. For all cycle scheme* $\mathfrak{s}$ *and* $y_j \in Y$*, the constant atom and the coefficients of all non-constant atoms in* $\Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})^-}(y_j)$ *are from a finite set* $U \subset \mathbb{Z}$ *comprising*

(1) the constant atom and the coefficients of the non-constant atoms in the expression $\Theta^{(C_i^{\ell_i},\Theta^{(H,\Omega_0)})^-}(y_j)$ *for* $i \in [n]$ *and* $\ell_i \in \{1,2\}$,

(2) the numbers $\alpha_{j,j'}^{\overline{C_{s_2}}} + \beta_{j,\pi^{\overline{C_{s_1}}}(j')}^{\overline{C_{s_1}}}$ *and* $\alpha_{j,j''}^{\overline{C_{s_1}}} + \alpha_{j,j''}^{\overline{C_{s_2}}}$, *where* $s_1, s_2 \in [n], j \in [l], j' \in I_{tr}^{\overline{C_{s_1}}} \cap I_{tr}^{\overline{C_{s_2}}}, j'' \in [k]$.

We define the abstraction of $\Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})^-}$, denoted by $\mathsf{Abs}(\mathfrak{s})$, as the union of the following three sets of tuples:

- the tuple for the constant atom: $\left\{\left(0, \left(\varepsilon_1^{(\overline{\mathfrak{s}})^-} + \lambda_1^{\overline{\mathfrak{s}}}\varepsilon_1^{\overline{H}}, \ldots, \varepsilon_1^{(\overline{\mathfrak{s}})^-} + \lambda_1^{\overline{\mathfrak{s}}}\varepsilon_1^{\overline{H}}\right)\right)\right\}$,
- tuples for the control variable atoms: $\{(j', (c_{j',1}, \ldots, c_{j',l})) \mid j' \in [k]\}$, where $c_{j',j}$ is the coefficient of the $\Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})^-}(x_{j'})$-atom in $\Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})^-}(y_j)$ for $j \in [l]$,
- tuples for the other atoms: $\{(k+1, (c_1, \ldots, c_l))\}$, where $(c_1, \ldots, c_l) \in U^l$ is the vector of coefficients of the $\mathfrak{d}'$-atom in $(\Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})^-}(y_j)$ for all $j \in [l]$ and $\mathfrak{d}' \notin \{\Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})^-}(x_{j'}) \mid x_{j'} \in X\}$.

Let $\mathscr{A} = \bigcup\{\mathsf{Abs}(\mathfrak{s}) \mid \mathfrak{s}$ a cycle scheme$\}$. Then $\mathscr{A}$ can be constructed as follows: We first compute $\mathsf{Abs}(HC_1), \ldots \mathsf{Abs}(HC_n)$, put them into $\mathscr{A}$, then compute from them the abstractions $\mathsf{Abs}(HC_1C_1), \ldots, \mathsf{Abs}(HC_1C_n), \mathsf{Abs}(HC_2C_1), \ldots$ by appending $C_1, \ldots, C_n$, put them into $\mathscr{A}$, and so on, until reaching a fixed point.

Step III. We first construct the set $\mathscr{A}$ and then.

1. Check whether there is $(0, (c_{0,1}, \ldots, c_{0,l})) \in \mathscr{A}$ such that $a_0 + b_1c_{0,1} + \cdots + b_lc_{0,l} \neq 0$. If the answer is yes, then return true.
2. Check whether there are $j' \in [k]$ and $(j', (c_{j',1}, \ldots, c_{j',l})) \in \mathscr{A}$ such that $a_{j'} + b_1c_{j',1} + \cdots + b_lc_{j',l} \neq 0$. If the answer is yes, then return true.
3. Check whether there is $(k+1, (c_1, \ldots, c_l)) \in \mathscr{A}$ such that $b_1c_1 + \cdots + b_lc_l \neq 0$. If the answer is yes, then return true.

If the decision procedure has not returned yet, return false.

Complexity Analysis of Step III. The size of the set U is polynomial over the size of the generalized lasso (i.e. the size of the transitions in the generalized lasso). The size of $\mathscr{A}$ is exponential over l, the number of data variables. The three conditions in Step III can be checked in time polynomial over the size of $\mathscr{A}$. In summary, the complexity of Step III is exponential over the number of data variables.

5.3 Decision Procedure for SNTs

We generalize the decision procedure to the case that the transition graphs of SNTs are generalized lassos to the full class of SNTs. We first define a

generalized multi-lasso as a sequence $\mathfrak{m} = H_1(C_{1,1}, \ldots, C_{1,n_1})H_2(C_{2,1}, \ldots, C_{2,n_2}) \ldots H_r(C_{r,1}, \ldots, C_{r,n_r})$ s.t. (1) for each $s \in [r]$, $H_s = q_{s,1} \xrightarrow{(g_2,\eta_2)} q_{s,2} \cdots q_{s,m_s-1} \xrightarrow{(g_{m_s},\eta_{m_s})} q_{s,m_s}$ is a generalized lasso, (2) for $1 \leq s < s' \leq r$, $H_s(C_{s,1}, \ldots, C_{s,n_s})$ and $H_{s'}(C_{s',1}, \ldots, C_{s',n_{s'}})$ are state-disjoint, except the case that when $s' = s + 1$, $q_{s,m_s} = q_{s',1}$, and (3) $q_{1,1} = q_0$.

Since the transition graph of $\mathcal{S}$ can be seen as a finite collection of generalized multi-lassos, in the following, we shall present the decision procedure by showing how to decide the non-zero output problem for generalized multi-lassos.

We fix a generalized multi-lasso below and assume without loss of generality that $O(q_{r,m_r}) = a_0 + a_1x_1 + \cdots + a_kx_k + b_1y_1 + \cdots + b_ly_l$ and $O(q')$ is undefined for every other state q' in $\mathfrak{m}$.

$$\mathfrak{m} = H_1(C_{1,1}, \ldots, C_{1,n_1})H_2(C_{2,1}, \ldots, C_{2,n_2}) \ldots H_r(C_{r,1}, \ldots, C_{r,n_r}).$$

Step I′: We do the same analysis as in Step I for the path $H_1 \ldots H_r$.

Step II′: Let $s \in [1, r-1]$. In order to analyze the set of cycles $\mathcal{C} = \{C_{s,1}, \ldots, C_{s,n_s}\}$, next we show how to summarize the effect of the path $H_{s+1} \ldots H_r$ on the values of the variables in the state q_{s,m_s} by extending the output function and defining $O(q_{s,m_s})$ (note that q_{s,m_s} is the unique state shared by all those cycles in $\mathcal{C}$). Suppose that $[\![O(q_{r,m_r})]\!]_{\Theta^{(H_{s+1} \ldots H_r, \Omega)}} = a_0 + a_1\Omega(x_1) + \cdots + a_k\Omega(x_k) + b_1\Omega(y_1) + \cdots + b_l\Omega(y_l) + e$, where $\Omega(x_1) \ldots \Omega(x_k)$ and $\Omega(y_1) \ldots \Omega(y_l)$ represent the values of $x_1 \ldots x_k$ and $y_1 \ldots y_l$ in the state q_{s,m_s}, and e is a linear combination of the variables that represent the data values introduced when traversing $H_{s+1} \ldots H_r$. Then we let $O(q_{s,m_s}) := a_0 + a_1x_1 + \cdots + a_kx_k + b_1y_1 + \cdots + b_ly_l$.

Step II′. For each $s \in [r]$, $s' \in [n_s]$, and each cycle scheme $\mathfrak{s} = C^{\ell_1}_{s,s'}C_{i_2} \ldots C_{i_t}$ such that $C_{i_2} \ldots C_{i_t} \in \{C_{s,1}, \ldots, C_{s,n_s}, \ldots, C_{r,1}, \ldots, C_{r,n_r}\}$ and $C_{i_2} \ldots C_{i_t}$ are mutually distinct, we perform an analysis of the expression $[\![O(q_{s,m_s})]\!]_{\Theta^{(\mathfrak{s}, \Theta^{(H_1 \ldots H_s, \Omega_0)})}}$, in a way similar to Step II. If the decision procedure does not return during the analysis, then go to Step III′.

Intuitively, in Step II′, during the analysis of the cycle scheme $\mathfrak{s} = C^{\ell_1}_{s,s'}C_{i_2} \ldots C_{i_t}$, the effect of the paths $H_{s+1}, \ldots, H_r$ and the cycles $C_{i_2}, \ldots, C_{i_t}$ on the atom coefficients which contain the cycle counter variable ℓ_1, is described by the expressions $\lambda_j^{\overline{H_{s+1}}} \ldots \lambda_j^{\overline{H_r}} \lambda_j^{\overline{C_{i_2}}} \ldots \lambda_j^{\overline{C_{i_t}}}$ for $j \in [l]$. Since the output expression $O(q_{s,m_s})$ defined above has already taken into consideration the expressions $\lambda_j^{\overline{H_{s+1}}} \ldots \lambda_j^{\overline{H_r}}$ for $j \in [l]$, in Step II′, we can do the analysis for the cycles in $\mathcal{C}$ as if we have a generalized lasso where the handle is $H_1 \ldots H_s$, the collection of cycles is $\{C_{s,1}, \ldots, C_{s,n_s}, \ldots, C_{r,1}, \ldots, C_{r,n_r}\}$, and the output state is q_{s,m_s}, with the output expression $O(q_{s,m_s})$.

Step III′: After Step II′, if the decision procedure has not returned yet, then similar to Lemma 3, the following hold.

- For each $s \in [r]$ and each path $\mathfrak{s} = H_1\mathfrak{s}_1H_2 \dots H_s\mathfrak{s}_s$ such that for each $s' \in [s]$, $\mathfrak{s}_{s'}$ is a cycle scheme over the collection of cycles $\{C_{s',1}, \dots, C_{s',n_{s'}}\}$, it holds that the constant atom and all the coefficients of the non-constant atoms in $\Theta^{(\mathfrak{s},\Omega_0)^-}(y_j)$ are from a bounded domain U.
- Moreover, an abstraction of $\mathfrak{s}$, denoted by $\mathsf{Abs}(\mathfrak{s})$, can be defined, so that $\mathscr{A}$, which contains the set of $\mathsf{Abs}(\mathfrak{s})$ for the paths $\mathfrak{s} = H_1\mathfrak{s}_1H_2 \dots H_s\mathfrak{s}_s$ (where $s \in [r]$), can be computed effectively from $H_1, C_{1,1}, \dots, C_{1,n_1}, H_2, \dots, H_r, C_{r,1}, \dots, C_{r,n_r}$.

Step III′. We apply the same analysis to $\mathscr{A}$ as in Step III. If the procedure does not return during the analysis, then return false.

Complexity Analysis of Step I′–III′. The complexity of Step I′ is polynomial in the maximum length of generalized multi-lassos in $\mathcal{S}$. The complexity of Step II″ is exponential in the maximum number of simple cycles in a generalized multi-lasso. The complexity of Step III′ is exponential in the number of data variables in $\mathcal{S}$.

```
int avg() {
 sum:=cur;
 cnt:=0;next;
 loop{
  sum+=cur;
  cnt+=1;
  next;}
 ret sum/cnt;}
```

```
int MAD() {
 sum:=cur;cnt:=0;next;
 loop{sum+=cur;cnt+=1;next;}
 avg:= sum/cnt;mad:=0;init;
 loop{
  if(cur<avg){mad+=avg-cur;}
  else{mad+=cur-avg;}next;}
 ret mad/cnt;}
```

```
int SD() {
 sum:=cur;cnt:=0;next;
 loop{sum+=cur;cnt+=1;next;}
 avg:= sum/cnt;sd:=0;init;
 loop{
  sd+=(cur-avg)*(cur-avg);next;
 }
 ret SQRT(sd/cnt);}
```

Fig. 4. More challenging examples of reducers performing data analytics operations

6 Extensions

In this section, we discuss some extensions of our approach to deal with the more challenging examples. For cases with multiplication, division, or other more complicated functions at the return point, e.g., the `avg` program, we can model them as an *uninterpreted k-ary function* and verify that all k parameters of the uninterpreted functions remain the same no matter how the input is permuted, e.g., the `avg` program always produces the same `sum` and `cnt` for all permutation of the same input data word. This is a *sound* but *incomplete* procedure for verifying programs of this type. Nevertheless, it is not often that a practical program for data analytics produces, e.g., $2q/2r$ from some input and q/r for its

permutation. Hence this procedure is often enough for proving commutativity for real world programs (Fig. 4).

The MAD (Mean Absolute Deviation) program is a bit more involved. Beside the division operator / that also occurs in the avg example, it uses a new iterator operation init, which resets cur to the head of the input data word. The strategy to verify this program is to divide the task into two parts: (1) ensure that the value of avg is independent of the order of the input, (2) treat avg as a control variable whose value is never updated and then check if the 2nd half of the program (c.f., Fig. 5) is commutative.

We handle the division at the end of the program in Fig. 5 in the same way as we did for the avg program. The guarantee we obtain after the corresponding SNT is checked to be commutative is that the program outputs the same value for any value of avg and any permutation of the input data word.

```
int MAD2() {
 avg:= cur;next;
 loop{
  if(cur<avg){mad+=avg-cur;}
  else{mad+=cur-avg;}
  next;}
 ret mad/cnt;}
```

Fig. 5. The 2nd half of MAD

The SD (Standard Deviation) program is even more challenging. The main difficulty comes from the use of multiplication in the middle of the program (instead of at the return point). In order to have a sound procedure to verify this kind of programs, we can extend the transitions of SNTs to include uninterpreted k-ary functions. However, this is not a trivial extension and we leave it as future work.

7 Conclusion

The contribution of the paper is twofold. We propose a verifiable programming language for reducers. Although it is still far away from a practical programming language, we believe that some ideas behind our language (e.g., the separation of control variables and data variables) would be valuable for the design of a practical reducer language. On the other hand, we propose the model of streaming numerical transducers, a transducer model over infinite alphabets. To our best knowledge, this is the first decidable automata model over infinite alphabets that allows linear arithmetics over the input values and the integer variables. Although we required that the transition graphs of SNTs are generalized flat, SNTs with such kind of transition graphs turn out to be quite powerful, since they are capable of simulating reducer programs without nested loops, which is a typical scenario of reducer programs in practice. At last, we would like to mention that although we assumed the integer data domain, all the results obtained in this paper are still valid when a dense data domain, e.g. the set of rational numbers, is assumed.

Acknowledgements. Yu-Fang Chen is partially supported by the MOST project No. 103-2221-E-001-019-MY3. Zhilin Wu is partially supported by the NSFC grants No. 61100062, 61272135, 61472474, and 61572478.

References

1. Alur, R., Cerny, P.: Streaming transducers for algorithmic verification of single-pass list-processing programs. In: POPL, pp. 599–610. ACM (2011)
2. Alur, R., Antoni, L.D, Deshmukh, J., Raghothaman, M., Yuan, Y.: Regular functions and cost register automata. In: LICS, pp. 13–22 (2013)
3. Chen, Y.F., Hong, C.D., Sinha, N., Wang, B.Y.: Commutativity of reducers. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 131–146. Springer, Heidelberg (2015)
4. Chen, Y., Lei, S., Wu, Z.: The commutativity problem of the mapreduce framework: a transducer-based approach. CoRR, abs/1605.01497 (2016)
5. Comon, H., Jurski, Y.: Multiple counters automata, safety analysis and presburger arithmetic. In: Hu, A.J., Vardi, M.Y. (eds.) CAV 1998. LNCS, vol. 1427, pp. 268–279. Springer, Heidelberg (1998)
6. Finkel, A., Göller, S., Haase, C.: Reachability in register machines with polynomial updates. In: Chatterjee, K., Sgall, J. (eds.) MFCS 2013. LNCS, vol. 8087, pp. 409–420. Springer, Heidelberg (2013)
7. Haase, C., Halfon, S.: Integer vector addition systems with states. In: Ouaknine, J., Potapov, I., Worrell, J. (eds.) RP 2014. LNCS, vol. 8762, pp. 112–124. Springer, Heidelberg (2014)
8. Hadoop. https://hadoop.apache.org
9. Ibarra, O.H.: Reversal-bounded multicounter machines and their decision problems. J. ACM **25**(1), 116–133 (1978)
10. Kaminski, M., Francez, N.: Finite-memory automata. Theor. Comput. Sci. **134**(2), 329–363 (1994)
11. Leroux, J., Sutre, G.: Flat counter automata almost everywhere! In: Software Verification: Infinite-State Model Checking and Static Program Analysis. Dagstuhl Seminar Proceedings, vol. 6081 (2006)
12. Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice-Hall, Englewood Cliffs (1971)
13. Neven, F., Schweikardt, N., Servais, F., Tan, T.: Distributed streaming with finite memory. In: ICDT, pp. 324–341 (2015)
14. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. ACM Trans. Comput. Logic **5**(3), 403–435 (2004)
15. Reinhardt, K.: Reachability in petri nets with inhibitor arcs. Electron. Notes Theor. Comput. Sci. **223**, 239–264 (2008). RP
16. Spark. http://spark.apache.org
17. Veanes, M., Hooimeijer, P., Livshits, B., Molnar, D., Bjorner, N.: Symbolic finite state transducers: algorithms and applications. ACM SIGPLAN Not. **47**(1), 137–150 (2012)
18. Xiao, T., Zhang, J., Zhou, H., Guo, Z., McDirmid, S., Lin, W., Chen, W., Zhou, L.: Nondeterminism in mapreduce considered harmful? An empirical study on non-commutative aggregators in mapreduce programs. In: ICSE, pp. 44–53 (2014)