arXiv:1605.01497v3 [cs.FL] 30 Sep 2016

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

Yu-Fang Chen[1], Lei Song[2], Zhilin Wu[2]

[1] Institute of Information Science, Academia Sinica
[2] State Key Laboratory of Computer Science,
Institute of Software, Chinese Academy of Sciences

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 [3]. More specifically, taking the "word-counting" program as an example. It counts the occurrences 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 $(word_1, list_1), (word_2, list_2), \dots, (word_n, list_n)$ and sent

* We found some inaccuracies in the conference version, which are fixed in this long version. We suggest the readers to refer to this version.

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

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 (Section 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* (Section 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 (Section 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 [9,5,11,7,6], 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 Section 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* over Z is a formula defined by the rules $g ::= \mathsf{true} \mid x_1 = x_2 \mid x_1 > x_2 \mid x_1 < x_2 \mid g \wedge g$, where $x_1, x_2 \in Z$. Let ρ be a valuation of X^+ and g be a guard over X^+ 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 Section 6, we will show an extension that allows resetting the iterators so that an input data word can be traversed multiple times.

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

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 appendix.

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

```
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;}
```

Fig. 2. Examples of Reducers Performing Data Analytics Operations

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. In SNT, we assume all data words end at a special symbol $\triangleright$, i.e., in the form of $\mathbb{Z}^*\triangleright$. 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 Section 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 $(q, g \wedge \mathsf{cur} \neq \triangleright, \eta, q')$, where $q, q' \in Q$, g is a guard over X^+ (defined in Section 2), and $\mathsf{cur} \neq \triangleright$ denotes the fact that the current position is not the end of the input, and η is an assignment function which is a partial function mapping $X \cup Y$ to $\mathcal{E}_{X^+ \cup Y}$ such that for each $x \in \mathsf{dom}(\eta) \cap X$, $\eta(x) = \mathsf{cur}$,
- and the tuples $(q, g \wedge \mathsf{cur} = \triangleright, \eta, q')$, where $q, q' \in Q$, g is a guard over X, and η is an assignment function such that $\mathsf{cur} \notin \mathsf{rng}(\eta)$ (they are called the $\triangleright$-transitions).

We write $q \xrightarrow{(g,\eta)} q'$ to denote $(q, g, \eta, q') \in \delta$ for convenience.

In the following, we assume that for each $\triangleright$-transition $(q, g \wedge \mathsf{cur} = \triangleright, \eta, q')$, q' is a sink-state, that is, that are no transitions out of q'. Moreover, we adopt the convention that when we mention the transition graph of an SNT $\mathcal{S}$, we always *ignore* the $\triangleright$-transitions.

In this paper, if not explicitly stated, we always assume that an SNT $\mathcal{S}$ satisfies the following additional 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) S of the transition graph of $\mathcal{S}$ is either a single state or a set of simple cycles $\{C_1, \ldots, C_n\}$ such that there

is a state q satisfying that each cycle C_i (where $1 \leq i \leq n$) is of length one and is a self-loop around q, (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^+, (4) *Monotone:* For each control variable $x \in X$ and each nontrivial SCC S of the transition graph, when staying in S, either the value of x is unchanged, or x computes the maximum or minimum value. Formally, for each nontrivial SCC S of the transition graph, the following conditions hold.

1. Each transition (q, g, η, q) in S satisfies that g is a conjunction of the formulae of the form $\mathsf{cur} = x$, $\mathsf{cur} < x$, or $\mathsf{cur} > x$, where $x \in X$, and for each $x' \in \mathsf{dom}(\eta)$, $\eta(x') = \mathsf{cur}$.
2. For each control variable $x \in X$, the following constraints hold,
   - for each self-loop (q, g, η, q) in S, it holds that $\mathsf{cur} = x$, $\mathsf{cur} < x$, or $\mathsf{cur} > x$ is a conjunct of g,
   - either all the self-loops (q, g, η, q) such that $\mathsf{cur} > x$ is a conjunct of g satisfy that $\eta(x) = \mathsf{cur}$, or none of them satisfies this constraint, similarly for the self-loops where $\mathsf{cur} < x$ occurs,
   - there do not exist self-loops (q, g_1, η_1, q) and (q, g_2, η_2, q) in S such that $\mathsf{cur} > x$ is a conjunct of g_1, $\eta_1(x) = \mathsf{cur}$, $\mathsf{cur} < x$ is a conjunct of g_2, and $\eta_2(x) = \mathsf{cur}$.

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 $(q_0, \rho_0)(q_1, \rho_1) \dots (q_n, \rho_n)$ is a *run* of $\mathcal{S}$ over a data word $w = d_1 \dots d_n \triangleright$ iff there exists a path (sequence of transitions) $P = q_0 \xrightarrow{(g_1,\eta_1)} q_1 \xrightarrow{(g_2,\eta_2)} q_2 \dots q_{n-1} \xrightarrow{(g_n,\eta_n)} q_n \xrightarrow{(g_{n+1},\eta_{n+1})} q_{n+1}$ such that for each $i \in [n+1]$, $\rho_{i-1}[d_i/\mathsf{cur}] \models g_i$, 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 $\rho_i(x) = \rho_{i-1}(x)$. (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+1}, ρ_{n+1}) 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.

Given a data word $w = d_1 \dots d_n \triangleright$ and an initial configuration (q_0, ρ_0), if there is a run of $\mathcal{S}$ over $w\triangleright$ starting from (q_0, ρ_0) and with the final configuration (q_{n+1}, ρ_{n+1}), then the output of $\mathcal{S}$ over $w\triangleright$ w.r.t. ρ_0, denoted by $\mathcal{S}_{\rho_0}(w\triangleright)$, is $[\![O(q_{n+1})]\!]_{\rho_{n+1}}$. Otherwise, $\mathcal{S}_{\rho_0}(w\triangleright)$ 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 $(\{q_0, q_1, q_2\}, \{\mathsf{max}\}, \emptyset, \delta, q_0, O)$, 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).

Proposition 1. *For each reducer program p, one can construct an equivalent SNT $\mathcal{S}_p$ which satisfies all the additional four constraints of SNTs, except the "Monotone" constraint. Moreover, the number of states of $\mathcal{S}_p$ and the maximum number of simple cycles in an SCC of the transition graph of $\mathcal{S}_p$ are at most exponential in the number of branching statements in p.*

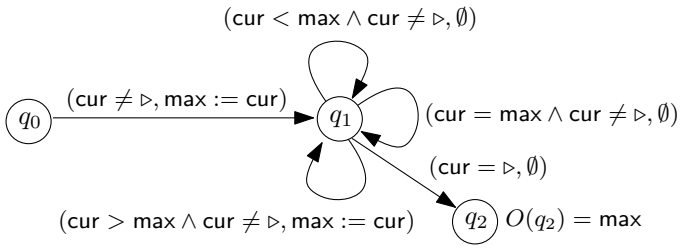

Fig. 3. The SNT $\mathcal{S}_{\max}$ for computing the maximum value

Intuitively, the main difference between reducer programs and SNTs are as follows: 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. As a result of this difference, when constructing $\mathcal{S}_p$ in Proposition 1, for each pair of consecutive "next;" statements in p, the subprogram between them is transformed into a collection of transitions in $\mathcal{S}_p$, one for each execution path in the subprogram. Since the number of execution paths in a program is exponential in the number of branching statements therein, there is an exponential blow-up in the construction. For a reducer program p, the SNT $\mathcal{S}_p$ constructed in Proposition 1 is not necessarily monotone. We define the class of *monotone programs* as the class of reducer programs p such that $\mathcal{S}_p$ is monotone. All the examples in Fig. 2 are monotone programs.

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 $w\triangleright$ and each permutation w' of w, $\mathcal{S}_{\rho_0}(w\triangleright) = \mathcal{S}_{\rho_0}(w'\triangleright)$ 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 $w\triangleright$, $\mathcal{S}_{\rho_0}(w\triangleright) = \mathcal{S}'_{\rho_0}(w\triangleright)$ 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 are a data word $w\triangleright$ and an initial valuation ρ_0 such that $\mathcal{S}_{\rho_0}(w\triangleright) \notin \{\bot, 0\}$.

We first observe that the commutativity problem of SNTs can be reduced to the equivalence problem, 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 will 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 number of simple cycles 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 $w = d_1 d_2 d_3 \dots d_n \triangleright$ with $n \geq 2$, $\mathcal{S}_1$ simulates the run of $\mathcal{S}$ over $d_2 d_1 d_3 \dots d_n \triangleright$, that is, the data word obtained from $w\triangleright$ by swapping the first two data values. An additional write-once control variable x' is introduced in $\mathcal{S}_1$ to store the data value d_1.
- Intuitively, over a data word $w = d_1 d_2 d_3 \dots d_n \triangleright$ with $n \geq 2$, $\mathcal{S}_2$ simulates the run of $\mathcal{S}$ over $d_2 d_3 \dots d_n d_1 \triangleright$, that is, the data word obtained from $w\triangleright$ by moving the first data value to the end. An additional write-once control variable x' is introduced in $\mathcal{S}_2$ to store the data value d_1.

The correctness of this reduction follows from the fact that all the permutations of $d_1 \dots d_n \triangleright$ 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 $\mathcal{S}$.

Next, we reduce the equivalence problem of two SNTs $\mathcal{S}_1, \mathcal{S}_2$ to the non-zero output problem of another SNT $\mathcal{S}_3$.

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

The SNT $\mathcal{S}_3$ can be constructed by the product construction. Note that the product construction preserves both the "Generalized-flat" and "Monotone" constraint.

To facilitate the decision procedure in the next section, in the following, we will show for each SNT $\mathcal{S}$, the nonzero-output problem of $\mathcal{S}$ can be reduced to a series of nonzero-output problems of normalized SNTs $\mathcal{S}'$ which enjoy some nice properties. To this end, we introduce some additional notations below.

Let P be a path in $\mathcal{S}$ from q_0 to some state q. We define Conf_P as the set of configurations (q, ρ) such that there are a data word w and an initial valuation ρ_0 satisfying that the run of $\mathcal{S}$ over w, starting from the initial configuration (q_0, ρ_0), follows a path PP' (i.e. P is a prefix of the path PP'), and reaches the configuration (q, ρ) after going through P. Let $x_1, x_2 \in X$. Then P is said to *enforce* $x_1 < x_2$ (resp. $x_1 = x_2, x_1 > x_2$) if for each $(q, \rho) \in \mathsf{Conf}_P$, it holds that $\rho \models x_1 < x_2$ (resp. $\rho \models x_1 = x_2$, $\rho \models x_1 > x_2$).

An SNT $\mathcal{S} = (Q, X, Y, \delta, q_0, O)$ is said to be *normalized* if (1) *Path-feasible:* for each path P starting from q_0, $\mathsf{Conf}_P \neq \emptyset$, (2) *State-dominating:* all the paths starting from the initial state q_0 and ending at a given state q (including the empty path q_0) enforce the same order relation between control variables, more precisely, for each state q and each pair of distinct control variables $x_1, x_2 \in X$, all the paths from q_0 to q enforce $x_1 > x_2$, or all the paths from q_0 to q enforce $x_1 < x_2$, or all the paths from q_0 to q enforce $x_1 = x_2$, (3) *$\triangleright$-transition-guard-free*: Each $\triangleright$-transition $(q, g \wedge \mathsf{cur} = \triangleright, \eta, q')$ satisfies that $g = \mathsf{true}$.

Proposition 4. *For each SNT $\mathcal{S}$, the nonzero-output problem of $\mathcal{S}$ can be reduced to a series of the nonzero-output problems of normalized SNTs $\mathcal{S}'$ in exponential time.*

The main idea of the reduction in Proposition 4 is to record in the states the order relation between control variables, which is enforced by the path that has been traversed

so far. Since a state in $\mathcal{S}$ may be split into several states in $\mathcal{S}'$, the transition graph of $\mathcal{S}'$ might not be generalized flat any more. Nevertheless, we can show that this will not happen and the "Monotone constraint" guarantees that the normalized SNTs $\mathcal{S}'$ are still generalized flat. The SNT$\mathcal{S}_{\mathsf{max}}$ illustrated in Fig. 3 is normalized, since there is just one control variable in $\mathcal{S}_{\mathsf{max}}$.

In the rest of this paper, we assume that all the SNTs $\mathcal{S} = (Q, X, Y, \delta, q_0, O)$ are normalized. Moreover, for each $q \in Q$, we use $\preceq_q$ to denote the total preorder comprising the pairs $(x_i, x_j), (x_j, x_i) \in X \times X$ such that all the paths from q_0 to q enforce $x_i = x_j$, the pairs $(x_i, x_j) \in X \times X$ such that all the paths from q_0 to q enforce $x_i < x_j$. Let $\sim_q$ denote the equivalence relation on $[k]$ induced by $\preceq_q$, that is, $i \sim_q j$ iff $x_i \preceq_q x_j$ and $x_j \preceq_q x_i$. We assume that each normalized SNT $\mathcal{S} = (Q, X, Y, \delta, q_0, O)$ satisfies that for each $q \in Q$, $\preceq_q$ and $\sim_q$ can be computed from q in linear time. In addition, we use $r^{\overline{q}}$ to denote the number of equivalence classes of $\sim_q$.

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 normalized 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 Section 5.1, then present a decision procedure for the case that the transition graph of $\mathcal{S}$ is a *generalized lasso* in Section 5.2. The transition graph of $\mathcal{S}$ is said to be a generalized lasso if it comprises a handle $H = q_0 \xrightarrow{(g_1,\eta_1)} q_1 \ldots q_{m-1} \xrightarrow{(g_m,\eta_m)} q_m$, a collection of simple cycles $C_1, \ldots, C_n$ such that each cycle C_i is a self-loop around q_m, and a $\triangleright$-transition $q_m \xrightarrow{(\mathsf{cur}=\triangleright,\eta_{m+1})} q_{m+1}$. We extend the procedure to SNTs whose transition graphs are not necessarily generalized lassos in Section 5.3.

Theorem 1. *The non-zero output problem of normalized SNTs can be decided in time exponential in the number of control and data variables and the number of simple cycles of the transition graph.*

Corollary 1. *The commutativity problem of monotone reducer programs can be decided in time exponential in the number of control and 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 monotone 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 $X \cup Y$ such that for each pair of variables $x_i, x_j \in X$, $\Omega(x_j) = \Omega(x_j)$ iff $x_i \sim_{p_0} x_j$. When P is traversed in a run of $\mathcal{S}$ over a data word w, the data value in a position of w may have to be (un)equal to the initial value of some control variable or some other data value in w that have been met before (enforced by the guards and assignments in P). Let $\sim_P$ denote the equivalence relation on $[n+k]$ induced by P defined as follows:

- For each $i, j \in [k]$, $i \sim_P j$ iff $x_i \sim_{p_0} x_j$.
- For each $i, j \in [n]$, $k+i \sim_P k+j$ iff the guards and assignments on P enforce that the data value in the i-th position of w must be equal to that in the j-th position of w.
- For each $i \in [k]$ and $j \in [n]$, $i \sim_P k+j$ iff the guards and assignments on P enforce that the data value in the j-th position of w must be equal to the initial value of x_i.

Assuming that there are $r^{\overline{P}}$ "*fresh*" equivalence classes of $\sim$, that is, equivalence classes J of $\sim_P$ such that $J \cap [k] = \emptyset$ (intuitively, the data value represented by J is not enforced to be equivalent to the initial values of control variables). 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 corresponding to these "fresh" equivalence classes, one for each such equivalence class. Note 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. In addition, we assume that there are $s^{\overline{p_0}}$ equivalence classes of $\sim_P$ on $[k]$, that is, equivalence classes J of $\sim_P$ on $[n+k]$ such that $J \cap [k] \neq \emptyset$. Suppose $J_1, \ldots, J_{s^{\overline{p_0}}}$ is an enumeration of these equivalence classes of $\sim_P$ on $[k]$. Let $\pi^{\overline{p_0}} : [s^{\overline{p_0}}] \to [k]$ such that $\pi^{\overline{p_0}}(j) = \min(J_j \cap [k])$ for each $j \in [s^{\overline{p_0}}]$. Intuitively, $\pi^{\overline{p_0}}$ chooses a representative control variable for each equivalence class. Note that $\pi^{\overline{p_0}}$ is an injective function, moreover, $s^{\overline{p_0}}$ and $\pi^{\overline{p_0}}$ are completely determined by $\sim_{p_0}$.

Example 2. Let $\mathcal{S}$ be an SNT where $X = \{x\}$, $Y = \{y\}$, and $P = p_0 \xrightarrow{(g_1,\eta_1)} p_1 \xrightarrow{(g_2,\eta_2)} p_2 \xrightarrow{(g_3,\eta_3)} p_3$ be a path of $\mathcal{S}$ such that $(g_1, \eta_1) = (\mathsf{cur} = x, y := \mathsf{cur})$, $(g_2, \eta_2) = (\mathsf{true}, (x := \mathsf{cur}, y := y + \mathsf{cur}))$, $(g_3, \eta_3) = (\mathsf{cur} = x, y := y + \mathsf{cur})$. Then $k = 1$, $n = 3$. The guards and assignments enforce that the data value in position 1 is equal to the initial value of x, which implies that $1 \sim_P 1+1$, i.e. $1 \sim_P 2$, in addition, the data value in position 2 is equal to that in position 3, which implies that $1+2 \sim_P 1+3$, i.e. $3 \sim_P 4$. Therefore, the equivalence relation $\sim_P$ has two equivalence classes, $\{1, 2\}$ and $\{3, 4\}$, of which $\{3, 4\}$ is the fresh equivalence class. We conclude that $r^{\overline{P}} = 1$ and $\mathfrak{d}_1^{\overline{P}}$ is used to denote the data value corresponding to this fresh equivalence class.

Proposition 5. *Suppose that P is a path starting form p_0 and the initial values of $X \cup Y$ are represented by a symbolic valuation Ω such that for each pair of variables $x_i, x_j \in X$, $\Omega(x_j) = \Omega(x_j)$ iff $x_i \sim_{p_0} x_j$. 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 it stores the initial value of some control variable after traversing P, otherwise, it is transient.*

- *For each $x_j \in X$ such that $j \in I^{\overline{P}}_{pe}$, $\Theta^{(P,\Omega)}(x_j) = \Omega(x_{\pi^{\overline{P_0}}(\pi^{\overline{P}}_{pe}(j))})$, where $\pi^{\overline{P}}_{pe} : I^{\overline{P}}_{pe} \to [s^{\overline{P_0}}]$ is a mapping from the index of a persistent control variable x_j to the index of the equivalence class such that the initial value of control variables corresponding to this equivalence class is assigned to x_j after traversing P.*
- *For each $x_j \in X$ such that $j \in I^{\overline{P}}_{tr}$, $\Theta^{(P,\Omega)}(x_j) = \mathfrak{d}^{\overline{P}}_{\pi^{\overline{P}}_{tr}(j)}$, where $\pi^{\overline{P}}_{tr} : I^{\overline{P}}_{tr} \to [r^{\overline{P}}]$ is a mapping from the index of a transient control variable to the index of the data value assigned to it.*
- *For each $y_j \in Y$,*

$$\Theta^{(P,\Omega)}(y_j) = \varepsilon^{\overline{P}}_j + \lambda^{\overline{P}}_j \Omega(y_j) + \sum_{j' \in [s^{\overline{P_0}}]} \alpha^{\overline{P}}_{j,j'} \Omega(x_{\pi^{\overline{P_0}}(j')}) + \sum_{j'' \in [r^{\overline{P}}]} \beta^{\overline{P}}_{j,j''} \mathfrak{d}^{\overline{P}}_{j''},$$

where $\varepsilon^{\overline{P}}_j, \lambda^{\overline{P}}_j, \alpha^{\overline{P}}_{j,1}, \ldots, \alpha^{\overline{P}}_{j,s^{\overline{P_0}}}, \beta^{\overline{P}}_{j,1}, \ldots, \beta^{\overline{P}}_{j,r^{\overline{P}}}$ are integer constants such that $\lambda^{\overline{P}}_j \in \{0,1\}$ (as a result of the "independently evolving and copyless" constraint). It can happen that $\lambda^{\overline{P}}_j = 0$, which means that $\Omega(y_j)$ is irrelevant to $\Theta^{(P,\Omega)}(y_j)$. Similarly for $\alpha^{\overline{P}}_{j,1} = 0$, and so on.

In Proposition 5, the sets $I^{\overline{P}}_{pe}$ and $I^{\overline{P}}_{tr}$, the mapping $\pi^{\overline{P}}_{pe}$ and $\pi^{\overline{P}}_{tr}$, and the constants $\varepsilon^{\overline{P}}_j, \lambda^{\overline{P}}_j, \ldots, \beta^{\overline{P}}_{j,r^{\overline{P}}}$ 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 $(\pi^{\overline{P}}_{pe})^{-1}$ as the inverse function of $\pi^{\overline{P}}_{pe}$, that is, for each $j' \in \mathsf{rng}(\pi^{\overline{P}}_{pe})$, $(\pi^{\overline{P}}_{pe})^{-1}(j') = \{j \in I^{\overline{P}}_{pe} \mid \pi^{\overline{P}}(j) = j'\}$. Similarly for $(\pi^{\overline{P}}_{tr})^{-1}$.

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 same as 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 $\Theta^{(P_2,\, \Theta^{(P_1,\Omega)})}$.*

Let the first state of P_1 and P_2 be $p_{1,0}$ and $p_{2,0}$ respectively. 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 $\Theta^{(P_1,\Omega)}$,

$$\begin{aligned}
\Theta^{(P_2,\, \Theta^{(P_1,\Omega)})}(y_j) = {} & \left(\varepsilon^{\overline{P_2}}_j + \lambda^{\overline{P_2}}_j \varepsilon^{\overline{P_1}}_j\right) + \left(\lambda^{\overline{P_2}}_j \lambda^{\overline{P_1}}_j\right) \Omega(y_j) + \\
& \sum_{j' \in \mathsf{rng}(\pi^{\overline{P_1}}_{pe})} \left(\lambda^{\overline{P_2}}_j \alpha^{\overline{P_1}}_{j,j'} + \sum_{j'' \in (\pi^{\overline{P_1}}_{pe})^{-1}(j') \cap \mathsf{rng}(\pi^{\overline{P_{2,0}}})} \alpha^{\overline{P_2}}_{j,(\pi^{\overline{P_{2,0}}})^{-1}(j'')} \right) \Omega(x_{\pi^{\overline{P_{1,0}}}(j')}) + \\
& \sum_{j' \in [s^{\overline{P_{1,0}}}] \setminus \mathsf{rng}(\pi^{\overline{P_1}}_{pe})} \left(\lambda^{\overline{P_2}}_j \alpha^{\overline{P_1}}_{j,j'}\right) \Omega(x_{\pi^{\overline{P_{1,0}}}(j')}) + \\
& \sum_{j' \in \mathsf{rng}(\pi^{\overline{P_1}}_{tr})} \left(\lambda^{\overline{P_2}}_j \beta^{\overline{P_1}}_{j,j'} + \sum_{j'' \in (\pi^{\overline{P_1}}_{tr})^{-1}(j') \cap \mathsf{rng}(\pi^{\overline{P_{2,0}}})} \alpha^{\overline{P_2}}_{j,(\pi^{\overline{P_{2,0}}})^{-1}(j'')} \right) \mathfrak{d}^{\overline{P_1}}_{j'} + \\
& \sum_{j' \in [r^{\overline{P_1}}] \setminus \mathsf{rng}(\pi^{\overline{P_1}}_{tr})} \left(\lambda^{\overline{P_2}}_j \beta^{\overline{P_1}}_{j,j'}\right) \mathfrak{d}^{\overline{P_1}}_{j'} + \sum_{j' \in [r^{\overline{P_2}}]} \beta^{\overline{P_2}}_{j,j'} \mathfrak{d}^{\overline{P_2}}_{j'}.
\end{aligned}$$

In the equation, $j' \in \mathsf{rng}(\pi_{pe}^{\overline{P_1}})$ implies that for each $j'' \in (\pi_{pe}^{\overline{P_1}})^{-1}(j')$, $x_{j''}$ stores the initial value of $x_{\pi^{\overline{p_{1,0}}}(j')}$ after traversing P_1, which means that the initial value of $x_{j''}$ for each $j'' \in (\pi_{pe}^{\overline{P_1}})^{-1}(j')$ before traversing P_2 is $\Omega(x_{\pi^{\overline{p_{1,0}}}(j')})$, and some of $x_{j''}$ for $j'' \in (\pi_{pe}^{\overline{P_1}})^{-1}(j')$ are chosen as the representatives of the equivalence classes of $\sim_{p_{2,0}}$ (in this case, j'' is in the range of $\pi^{\overline{p_{2,0}}}$), therefore we have the item $\left(\sum_{j'' \in (\pi_{pe}^{\overline{P_1}})^{-1}(j') \cap \mathsf{rng}(\pi^{\overline{p_{2,0}}})} \alpha_{j,(\pi^{\overline{p_{2,0}}})^{-1}(j'')}^{\overline{P_2}}\right) \Omega(x_{\pi^{\overline{p_{1,0}}}(j')})$. When $j' \in \mathsf{rng}(\pi_{tr}^{\overline{P_1}})$, the initial value of $x_{j''}$ for each $j'' \in (\pi_{tr}^{\overline{P_1}})^{-1}(j')$ before traversing P_2 is $\mathfrak{d}_{j'}^{\overline{P_1}}$, and some of $x_{j''}$ for $j'' \in (\pi_{tr}^{\overline{P_1}})^{-1}(j')$ are chosen as the representatives of equivalence classes of $\sim_{p_{2,0}}$, therefore we have the item $\left(\sum_{j'' \in (\pi_{tr}^{\overline{P_1}})^{-1}(j') \cap \mathsf{rng}(\pi^{\overline{p_{2,0}}})} \alpha_{j,(\pi^{\overline{p_{2,0}}})^{-1}(j'')}^{\overline{P_2}}\right) \mathfrak{d}_{j'}^{\overline{P_1}}$. For $j' \in [s^{\overline{p_{1,0}}}] = \mathsf{rng}(\pi_{pe}^{\overline{P_1}}) \cup ([s^{\overline{p_{1,0}}}] \setminus \mathsf{rng}(\pi_{pe}^{\overline{P_1}}))$, we have the item $(\lambda_j^{\overline{P_2}} \alpha_{j,j'}^{\overline{P_1}}) \Omega(x_{\pi^{\overline{p_{1,0}}}(j')})$, i.e. the coefficient of $\Omega(x_{\pi^{\overline{p_{1,0}}}(j')})$ in $\Theta^{(P_1,\Omega)}$ multiplied by $\lambda_j^{\overline{P_2}}$. Moreover, for $j' \in [r^{\overline{P_1}}] = \mathsf{rng}(\pi_{tr}^{\overline{P_1}}) \cup ([r^{\overline{P_1}}] \setminus \mathsf{rng}(\pi_{tr}^{\overline{P_1}}))$, 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 ($\ell \geq 1$) which is obtained by iterating a simple cycle $C = (q, g, \eta, q)$ 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*. It is easy to observe that both $I_{pe}^{\overline{C}}$ and $I_{tr}^{\overline{C}}$ are the union of the equivalence classes of $\sim_q$. From the "Monotone" constraint, we know that for each $x \in \mathsf{dom}(\eta)$, it holds that $\eta(x) = \mathsf{cur}$. Therefore, if $j \in I_{pe}^{\overline{C}}$, then x_j still stores the initial value of x_j after traversing C. This implies that for each $j' \in \mathsf{rng}(\pi_{pe}^{\overline{C}})$, let $\pi^{\overline{q}}(j') = j$, then $\pi_{pe}^{\overline{C}}(j) = j'$. Therefore, for each $j' \in \mathsf{rng}(\pi_{pe}^{\overline{C}})$, the value of $x_{\pi^{\overline{q}}(j')}$ is unchanged after traversing C.

Proposition 6. *Suppose that C is a simple cycle (i.e. a self-loop around a state q) 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:*

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

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

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

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

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

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

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

Example 3. Let $\mathcal{S}'_{\mathsf{max}}$ be the SNT illustrated in Fig. 4, which is obtained from $\mathcal{S}_{\mathsf{max}}$ by replacing the control variable max with x_1 and introducing the data variables y_1, y_2, y_3. Consider the cycle C_1 in $\mathcal{S}'_{\mathsf{max}}$. Then $O(q_2) = a_0 + a_1x_1 + b_1y_1 + b_2y_2 + b_3y_3 = y_1 - 2y_2 + y_3$. Moreover, $I_{pe}^{\overline{C_1}} = \{1\}$, $I_{tr}^{\overline{C_1}} = \emptyset$, $\pi_{pe}^{\overline{C_1}}(1) = 1$, $\lambda_1^{\overline{C_1}} = \lambda_2^{\overline{C_1}} = 1$, and $\lambda_3^{\overline{C_1}} = 0$. On the other hand, $I_{pe}^{\overline{C_2}} = \emptyset$, $I_{tr}^{\overline{C_2}} = \{1\}$, $\pi_{tr}^{\overline{C_2}}(1) = 1$, and $\lambda_1^{\overline{C_2}} = \lambda_2^{\overline{C_2}} = \lambda_3^{\overline{C_2}} = 1$. Suppose $\ell \geq 2$. Let $\mathfrak{d}_1^{\overline{C_1,1}}, \ldots, \mathfrak{d}_1^{\overline{C_1,\ell}}$ represent the data values introduced when traversing the path C_1^ℓ, then

$$\begin{aligned}
\Theta^{(C_1^\ell,\Omega)}(y_1) &= \Omega(y_1) + (4\ell)\Omega(x_1) + \mathfrak{d}_1^{\overline{C_1,1}} + \cdots + \mathfrak{d}_1^{\overline{C_1,\ell}},\\
\Theta^{(C_1^\ell,\Omega)}(y_2) &= \Omega(y_2) + (2\ell)\Omega(x_1) + 2\mathfrak{d}_1^{\overline{C_1,1}} + \cdots + 2\mathfrak{d}_1^{\overline{C_1,\ell}},\\
\Theta^{(C_1^\ell,\Omega)}(y_3) &= 3\mathfrak{d}_1^{\overline{C_1,\ell}}.
\end{aligned}$$

On the other hand, let $\mathfrak{d}_1^{\overline{C_2,1}}, \ldots, \mathfrak{d}_1^{\overline{C_2,\ell}}$ represent the data values introduced when traversing the path C_2^ℓ, then

$$\begin{aligned}
\Theta^{(C_2^\ell,\Omega)}(y_1) &= \Omega(y_1) + \Omega(x_1) + 4\mathfrak{d}_1^{\overline{C_2,1}} + \cdots + 4\mathfrak{d}_1^{\overline{C_2,\ell-1}} + 3\mathfrak{d}_1^{\overline{C_2,\ell}},\\
\Theta^{(C_2^\ell,\Omega)}(y_2) &= \Omega(y_2) + 3\Omega(x_1) + 5\mathfrak{d}_1^{\overline{C_2,1}} + \cdots + 5\mathfrak{d}_1^{\overline{C_2,\ell-1}} + 2\mathfrak{d}_1^{\overline{C_2,\ell}},\\
\Theta^{(C_2^\ell,\Omega)}(y_3) &= \Omega(y_3) + 5\Omega(x_1) + 6\mathfrak{d}_1^{\overline{C_2,1}} + \cdots + 6\mathfrak{d}_1^{\overline{C_2,\ell-1}} + \mathfrak{d}_1^{\overline{C_2,\ell}}.
\end{aligned}$$

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

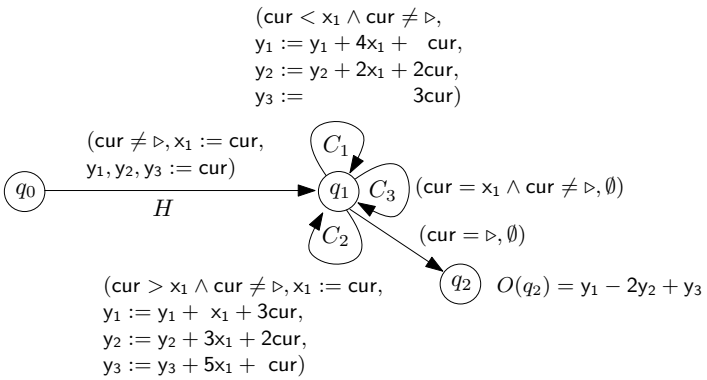

Fig. 4. The SNT $\mathcal{S}'_{\max}$: Extending $\mathcal{S}_{\max}$ with data variables

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 ouput problem of SNTs 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 $H = q_0 \xrightarrow{(g_1,\eta_1)} q_1 \ldots q_{m-1} \xrightarrow{(g_m,\eta_m)} q_m$, 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 $\{C_1, \ldots, C_n\}$, and a $\triangleright$-transition $q_m \xrightarrow{(\mathsf{cur}=\triangleright,\eta)} q_{m+1}$. Moreover, without loss of generality, we assume that $O(q_{m+1}) = a_0 + a_1x_1 + \cdots + a_kx_k + b_1y_1 + \cdots + b_ly_l$, and $O(q)$ is undefined for all the other states q. For convenience, we define $O(q_m)$ by replacing simultaneously z with $\eta(z)$ in $O(q_{m+1})$, for each $z \in \mathsf{dom}(\eta)$. Suppose $O(q_m) = a'_0 + a'_1x_1 + \cdots + a'_kx_k + b'_1y_1 + \cdots + b'_ly_l$. Then for the non-zero output problem, we can ignore the $\triangleright$-transition and use $O(q_m)$ directly.

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 $\mathfrak{s} = C_{i_1}^{\ell_1} C_{i_2}^{\ell_2} \dots C_{i_t}^{\ell_t}$ is a cycle scheme, and Ω is a symbolic valuation representing the initial values of the control and data variables such that for each $x_i, x_j \in X$, $\Omega(x_i) = \Omega(x_j)$ iff $i \sim_{q_m} j$. For all $j' \in I_{pe}^{\overline{C_{i_1}}} \cap \mathsf{rng}(\pi^{\overline{q_m}})$, 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 $C_{i_1}^{\ell_1} C_{i_2}^{\ell_2} \dots C_{i_{r_{j'}}}^{\ell_{r_{j'}}}$. Then for each $j \in [l]$ and $j' \in I_{pe}^{\overline{C_{i_1}}} \cap \mathsf{rng}(\pi^{\overline{q_m}})$, the coefficient of the $\Omega(x_{j'})$-atom in $\Theta^{(\mathfrak{s},\Omega)}(y_j)$ is*

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

where (1) $e = 0$ when $r_{j'} = t$ and (2) $e = (\lambda_j^{\overline{C_{i_s}}})^{\ell_s-1} \alpha^{\overline{C_{i_s}}}_{j,(\pi^{\overline{q_m}})^{-1}(j')} \prod_{s'\in[s+1,t]} \left(\lambda_j^{\overline{C_{i_{s'}}}}\right)^{\ell_{s'}}$ with $s = r_{j'} + 1$ when $r_{j'} < t$.
The constant atom of $\Theta^{(\mathfrak{s},\Omega)}(y_j)$ is

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

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 $j' \in I_{pe}^{\overline{C_{i_1}}} \cap \mathsf{rng}(\pi^{\overline{q_m}})$ 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 $(1 + \lambda_j^{\overline{C_{i_{s_1}}}} + \dots + (\lambda_j^{\overline{C_{i_{s_1}}}})^{\ell_{s_1}-1})$ 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 coefficients of the $\Omega(x_{j'})$-atoms with $j' \in I_{pe}^{\overline{C_{i_1}}} \cap \mathsf{rng}(\pi^{\overline{q_m}})$ can be rewritten to the form of $c_0 + c_1\ell_1 + c_2\ell_2 + \dots + c_t\ell_t$ for $c_0 \dots c_t \in \mathbb{Z}$. Note that some of $c_0 \dots c_t$ 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 $\mathfrak{d}^{\overline{H}}_{\pi_{tr}^{\overline{H}}(j')}$, more formally, $\Theta^{(H,\Omega_0)}(x_{j'}) = \mathfrak{d}^{\overline{H}}_{\pi_{tr}^{\overline{H}}(j')}$.

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.

Example 4. Let $\mathcal{S}'_{\mathsf{max}}$ be the SNT in Example 3. Since $q_1 \xrightarrow{(\mathsf{cur}=\triangleright,\emptyset)} q_2$, we can define $O(q_1) := a'_0 + a'_1x_1 + b'_1y_1 + b'_2y_2 + b'_4y_3 = O(q_2) = y_1 - 2y_2 + y_3$. The handle H comprises exactly one transition. Thus $r^{\overline{H}} = 1$ and

$$[\![O(q_1)]\!]_{\Theta^{(H,\Omega_0)}} = \mathfrak{d}_1^{\overline{H}} - 2\mathfrak{d}_1^{\overline{H}} + \mathfrak{d}_1^{\overline{H}} = 0.$$

Therefore, the handle H does not produce a non-zero output.

Step II: The goal of Step II is to show that

- either for each cycle scheme $\mathfrak{s}$, all subexpressions in $[\![O(q_m)]\!]_{\Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})}}$ containing the cycle counter variables are identical to zero and hence can be ignored,
- or there exists a cycle scheme $\mathfrak{s}$ such that $[\![O(q_m)]\!]_{\Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})}}$ is not identical to zero.

Let $\mathfrak{s} = C_{i_1}^{\ell_1}C_{i_2}^{\ell_2}\dots C_{i_t}^{\ell_t}$ be a cycle scheme. From Lemma 1, for each $j' \in I_{pe}^{\overline{C_{i_1}}} \cap \mathsf{rng}(\pi^{\overline{q_m}})$ 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

$$\sum_{1\le j\le l} b'_j \left((\lambda_j^{\overline{C_{i_2}}})^{\ell_2}\dots(\lambda_j^{\overline{C_{i_t}}})^{\ell_t}\right)\left(1+\lambda_j^{\overline{C_{i_1}}}+\dots+(\lambda_j^{\overline{C_{i_1}}})^{\ell_1-1}\right)\alpha^{\overline{C_{i_1}}}_{j,(\pi^{\overline{q_m}})^{-1}(j')}. \quad (*)$$

Since $\lambda_j^{\overline{C_{i_1}}}, \lambda_j^{\overline{C_{i_2}}}, \dots, \lambda_j^{\overline{C_{i_t}}} \in \{0,1\}$, 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

$$\sum_{1\le j\le l} b'_j \left((\lambda_j^{\overline{C_{i_2}}})^{\ell_2}\dots(\lambda_j^{\overline{C_{i_t}}})^{\ell_t}\right)\left(1+\lambda_j^{\overline{C_{i_1}}}+\dots+(\lambda_j^{\overline{C_{i_1}}})^{\ell_1-1}\right)\varepsilon_j^{\overline{C_{i_1}}}. \quad (**)$$

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 $\nu_{\mathfrak{s},(i_1,0)}$.

We are ready to present Step II.

Step II. For each $i_1 \in [n]$, check all cycle scheme $\mathfrak{s} = C_{i_1}^{\ell_1}C_{i_2}\dots C_{i_t}$ such that $i_2, \dots, i_t$ 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 $j' \in I_{pe}^{\overline{C_{i_1}}} \cap \mathsf{rng}(\pi^{\overline{q_m}})$ 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 $i_1, \dots, i_t$ are mutually distinct, the number of cycle schemes $\mathfrak{s} = C_{i_1}^{\ell_1}C_{i_2}\dots C_{i_t}$ in Step II is exponential in the number of cycles in the generalized lasso. Once the cycle scheme $\mathfrak{s}$ 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 simple cycles in the generalized lasso.

Example 5. Let $\mathcal{S}'_{\mathsf{max}}$ be the SNT in Example 4. We need consider the following cycle schemes, $C_1^{\ell_1}$, $C_2^{\ell_1}$, $C_1^{\ell_1}C_2$, $C_2^{\ell_1}C_1$, $C_1^{\ell_1}C_2C_1$, $C_2^{\ell_1}C_1C_2$. We use $\mathfrak{s}_1 = C_1^{\ell_1}$, $\mathfrak{s}_2 = C_1^{\ell_1}C_2$, and $\mathfrak{s}_3 = C_1^{\ell_1}C_2C_1$ to illustrate Step II.

- In $[\![O(q_1)]\!]_{\Theta^{(\mathfrak{s}_1,\Omega)}}$, the subexpression containing ℓ_1 in the coefficient of the $\Omega(x_1)$-atom is
$$\sum_{1\le j\le 3} b'_j \left(1+\lambda_j^{\overline{C_1}}+\cdots+(\lambda_j^{\overline{C_1}})^{\ell_1-1}\right)\alpha_{j,1}^{\overline{C_1}}$$
$$= 1\times\ell_1\times 4+(-2)\times\ell_1\times 2+1\times 1\times 0 = 0.$$
- In $[\![O(q_1)]\!]_{\Theta^{(\mathfrak{s}_2,\Omega)}}$, the subexpression containing ℓ_1 in the coefficient of the $\Omega(x_1)$-atom is
$$\sum_{1\le j\le 3} b'_j\,\lambda_j^{\overline{C_2}}\left(1+\lambda_j^{\overline{C_1}}+\cdots+(\lambda_j^{\overline{C_1}})^{\ell_1-1}\right)\alpha_{j,1}^{\overline{C_1}}$$
$$= 1\times 1\times\ell_1\times 4+(-2)\times 1\times\ell_1\times 2+1\times 1\times 1\times 0 = 0.$$
- In $[\![O(q_1)]\!]_{\Theta^{(\mathfrak{s}_3,\Omega)}}$, the subexpression containing ℓ_1 in the coefficient of the $\Omega(x_1)$-atom is
$$\sum_{1\le j\le 3} b_j\,(\lambda_j^{\overline{C_2}}\lambda_j^{\overline{C_1}})\left(1+\lambda_j^{\overline{C_1}}+\cdots+(\lambda_j^{\overline{C_1}})^{\ell_1-1}\right)\alpha_{j,1}^{\overline{C_1}}$$
$$= 1\times 1\times\ell_1\times 4+(-2)\times 1\times\ell_1\times 2+1\times 0\times 1\times 0 = 0.$$

From Example 3, we know that if $\mathfrak{s} = C_2^{\ell_1}, C_2^{\ell_1}C_1$, or $C_2^{\ell_1}C_1C_2$, then for each $j = 1,2,3$, the coefficient of the $\Omega(x_1)$-atom in $\Theta^{(\mathfrak{s},\Omega)}(y_j)$ does not contain the cycle counter variable ℓ_1. Therefore, the coefficient of the $\Omega(x_1)$-atom in $[\![O(q_1)]\!]_{\Theta^{(\mathfrak{s},\Omega)}}$ does not contain ℓ_1 as well. From this, we conclude that with $\mathcal{S}'_{\mathsf{max}}$ as the input, the decision procedure does not return in Step II and it will go to Step III.

In the following, we present the arguments for the correctness of Step II. Suppose $\mathfrak{s} = C_{i_1}^{\ell_1}C_{i_2}\dots C_{i_t}$.

- If there exists $j' \in I_{pe}^{\overline{C_{i_1}}} \cap \mathsf{rng}(\pi^{\overline{q_m}})$ and $\mu_{\mathfrak{s},(i_1,j')} \neq 0$. Then from the cycle scheme $\mathfrak{s} = C_{i_1}^{\ell_1}C_{i_2}\dots C_{i_t}$, we can assign a sufficiently large value s to ℓ_1 so that the coefficient of the $\Omega(x_{j'})$-atom in $[\![O(q_m)]\!]_{\Theta^{(H_{\mathfrak{s}},\Omega_0)}}$, which is equal to the sum of $\ell_1\mu_{\mathfrak{s},(i_1,j')}$ and some constant, becomes non-zero. The guards and assignments in the path $HC_{i_1}^sC_{i_2}\dots C_{i_t}$ 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 $[\![O(q_m)]\!]_{\Theta^{(P_{\mathfrak{s}},\Omega_0)}}$. This is sufficient to make $[\![O(q_m)]\!]_{\Theta^{(H_{\mathfrak{s}},\Omega_0)}}$ non-zero.
- Otherwise, if $[\![O(q_m)]\!]_{\Theta^{(H_{\mathfrak{s}},\Omega_0)}}$ contains some other nontrivial non-constant atoms, then we can apply a similar argument as above and conclude that $[\![O(q_m)]\!]_{\Theta^{(H_{\mathfrak{s}},\Omega_0)}}$ can be made non-zero.
- On the other hand, if $[\![O(q_m)]\!]_{\Theta^{(H_{\mathfrak{s}},\Omega_0)}}$ contains no nontrivial non-constant atoms, but $\mu_{\mathfrak{s},(i_1,0)} \neq 0$, then we can let ℓ_1 sufficiently large to make the expression $[\![O(q_m)]\!]_{\Theta^{(H_{\mathfrak{s}},\Omega_0)}}$ non-zero.

Therefore, when Step II returns true, that is, at least one of the two conditions in Step II holds, then we are able to conclude that there *must be* an input data word w and an initial valuation ρ_0 such that $\mathcal{S}_{\rho_0}(w) \notin \{\bot, 0\}$.

If Step II does not return true, we show below that for all cycle schemes $\mathfrak{s}_1 = C_{i_1}^{\ell_1} C_{i_2}^{\ell_2} \dots C_{i_{t_1}}^{\ell_{t_1}}$ with $i_1, i_2, \dots, i_{t_1} \in [n]$, all the subexpressions containing cycle counter variables $\ell_1, \dots, \ell_{t_1}$ in $[\![O(q_m)]\!]_{\Theta^{(\mathfrak{s}_1, \Theta^{(H,\Omega_0)})}}$ are identical to zero and hence can be removed. Let $i'_2 \dots i'_{t_2}$ be the sequence obtained from $i_2 \dots i_{t_1}$ by keeping just one copy for each duplicated index therein. In Step II we already checked the cycle scheme $\mathfrak{s}_2 = C_{i_1}^{\ell_1} C_{i'_2} \dots C_{i'_{t_2}}$. Step II guarantees that all the subexpressions containing ℓ_1 in $[\![O(q_m)]\!]_{\Theta^{(\mathfrak{s}_2, \Theta^{(H,\Omega_0)})}}$ are identical to zero and hence can be removed. Because for all $j \in [l]$, $\lambda_j^{\overline{C_1}}, \dots, \lambda_j^{\overline{C_n}} \in \{0,1\}$, $(\lambda_j^{\overline{C_{i_2}}})^{\ell_2} \dots (\lambda_j^{\overline{C_{i_{t_1}}}})^{\ell_{t_1}} = \lambda_j^{\overline{C_{i'_2}}} \dots \lambda_j^{\overline{C_{i'_{t_2}}}}$. We proved that the $(*)$ and $(**)$ style expressions are equivalent in both $[\![O(q_m)]\!]_{\Theta^{(\mathfrak{s}_1, \Theta^{(H,\Omega_0)})}}$ and $[\![O(q_m)]\!]_{\Theta^{(\mathfrak{s}_2, \Theta^{(H,\Omega_0)})}}$. Hence we can also remove all subexpressions containing ℓ_1 from $[\![O(q_m)]\!]_{\Theta^{(\mathfrak{s}_1, \Theta^{(H,\Omega_0)})}}$, without affecting its value. Those subexpressions containing ℓ_2 can also be removed by considering the cycle scheme $\mathfrak{s}_3 = C_{i_2}^{\ell_2} C_{i''_3} \dots C_{i''_{t_3}}$ and applying a similar reasoning, where the sequence $i''_3 \dots i''_{t_3}$ is obtained from $i_3 \dots i_{t_1}$, similarly to the construction of $i'_2 \dots i'_{t_2}$ from $i_2 \dots i_{t_1}$. The same applies to all other cycle counter variables $\ell_3, \dots, \ell_{t_1}$. For each $y_j \in Y$, we use the notation $\Theta^{(\mathfrak{s}_1,\Omega)^-}(y_j)$ to denote the expression obtained by removing from the constant atom and coefficients of the non-constant atoms of $\Theta^{(\mathfrak{s}_1,\Omega)}(y_j)$ all subexpressions containing the cycle counter variables.

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: From Proposition 6, we can observe that for each cycle C_i (where $1 \leq i \leq n$), the expression $\Theta^{(C_i^\ell, \Theta^{(H,\Omega_0)})^-}(y_j)$ for $y_j \in Y$ is of one of the the following forms.

- If $\lambda_j^{\overline{C_i}} = 0$, then $\Theta^{(C_i^\ell, \Theta^{(H,\Omega_0)})^-}(y_j) = \Theta^{(C_i^\ell, \Theta^{(H,\Omega_0)})}(y_j)$.
- If $\lambda_j^{\overline{C_i}} = 1$, then

$$\Theta^{(C_i^\ell, \Theta^{(H,\Omega_0)})^-}(y_j) = \Theta^{(H,\Omega_0)}(y_j) + \sum_{j' \in [s^{\overline{q_m}}] \setminus \mathsf{rng}(\pi_{pe}^{\overline{C_i}})} \alpha_{j,j'}^{\overline{C_i}} \Omega(x_{\pi^{\overline{q_m}}(j')}) + \sum_{j' \in \mathsf{rng}(\pi_{tr}^{\overline{C_i}})} \sum_{s \in [\ell-1]} \left(\beta_{j,j'}^{\overline{C_i}} + \sum_{j'' \in (\pi_{tr}^{\overline{C_i}})^{-1}(j') \cap \mathsf{rng}(\pi^{\overline{q_m}})} \alpha_{j,(\pi^{\overline{q_m}})^{-1}(j'')}^{\overline{C_i}} \right) \mathfrak{d}_{j'}^{\overline{C_i,s}} + \sum_{j' \in [r^{\overline{C_i}}] \setminus \mathsf{rng}(\pi_{tr}^{\overline{C_i}})} \sum_{s \in [\ell-1]} \beta_{j,j'}^{\overline{C_i}} \mathfrak{d}_{j'}^{\overline{C_i,s}} + \sum_{j' \in [r^{\overline{C_i}}]} \beta_{j,j'}^{\overline{C_i}} \mathfrak{d}_{j'}^{\overline{C_i,\ell}}.$$

Note that if $\lambda_j^{\overline{C_i}} = 1$, then the expressions containing the cycle counter variable ℓ, e.g. $\ell \varepsilon_j^{\overline{C_i}}$, are removed from $\Theta^{(C_i^\ell, \Theta^{(H,\Omega_0)})^-}(y_j)$.

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

- the tuple for the constant atom: $\left\{\left(0, \left(\varepsilon_1^{(\overline{\mathfrak{s}})^-} + \lambda_1^{\overline{\mathfrak{s}}}\varepsilon_1^{\overline{H}}, \ldots, \varepsilon_l^{(\overline{\mathfrak{s}})^-} + \lambda_l^{\overline{\mathfrak{s}}}\varepsilon_l^{\overline{H}}\right)\right)\right\}$,
- 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 $(c_1, \ldots, c_l)$ 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 $\mathfrak{d}' \notin \{\Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})}(x_{j'}) \mid x_{j'} \in X\}$.

Note that in $\mathsf{Abs}(H\mathfrak{s})$, if $j_1, j_2 \in [k]$ and $j_1 \sim_{q_m} j_2$, then we have $\Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})}(x_{j_1}) = \Theta^{(\mathfrak{s},\Theta^{(H,\Omega_0)})}(x_{j_2})$. Therefore, $(c_{j_1,1}, \ldots, c_{j_1,l}) = (c_{j_2,1}, \ldots, c_{j_2,l})$.

Let $\mathscr{A} = \bigcup\{\mathsf{Abs}(H\mathfrak{s}) \mid \mathfrak{s} \text{ a cycle scheme}\}$. Then $\mathscr{A}$ can be constructed inductively as follows, until $\mathscr{A}_{i+1} = \mathscr{A}_i$.

1. $\mathscr{A}_0 = \{\mathsf{Abs}(HC_1), \ldots \mathsf{Abs}(HC_n)\}$,
2. For $i \geq 0$, $\mathscr{A}_{i+1}$ is the union of $\mathscr{A}_i$ and the set of Λ' such that Λ' is constructed from some $\Lambda \in \mathscr{A}_i$ and some cycle $C_{i'}$ (where $i' \in [n]$) as follows. At first we observe that $\sim_{q_m} \subseteq (I_{pe}^{\overline{C_{i'}}} \times I_{pe}^{\overline{C_{i'}}}) \cup (I_{tr}^{\overline{C_{i'}}} \times I_{tr}^{\overline{C_{i'}}})$. The argument is as follows: If $j' \in I_{tr}^{\overline{C_{i'}}}$ and $j'' \in I_{pe}^{\overline{C_{i'}}}$, then $x_{j'}$ is assigned a fresh value and $x_{j''}$ is assigned the initial value of some control variable, thus $x_{j'}$ and $x_{j''}$ are not equivalent w.r.t. $\sim_{q_m}$.
   - Suppose $(0, (c_1, \ldots, c_l)) \in \Lambda$. Then $(0, (c'_1, \ldots, c'_l)) \in \Lambda'$, where for each $j \in [l]$, if $\lambda_j^{\overline{C_{i'}}} = 0$, then $c'_j = \varepsilon_j^{\overline{C_{i'}}}$, otherwise, $c'_j = c_j$ (in this case, the expression $\varepsilon_j^{\overline{C_{i'}}}$ is removed).
   - For each $j' \in [k]$, suppose $(j', (c_{j',1}, \ldots, c_{j',l})) \in \Lambda$, do the following.
     - If $j' \in I_{pe}^{\overline{C_{i'}}}$, then $(j', (c'_{j',1}, \ldots, c'_{j',l})) \in \Lambda'$, where for each $j \in [l]$,
       - if $\lambda_j^{\overline{C_{i'}}} = 0$, then $c'_{j',j} = \alpha^{\overline{C_{i'}}}_{j,\pi_{pe}^{\overline{C_{i'}}}(j')}$,
       - otherwise, $c'_{j',j} = c_{j',j}$ (in this case, the expressions $\alpha^{\overline{C_{i'}}}_{j,\pi_{pe}^{\overline{C_{i'}}}(j')}$ is removed).
     - If $j' \in I_{tr}^{\overline{C_{i'}}}$, then

       $$(k+1, (c'_{j',1}, \ldots, c'_{j',l})), \left(j', \left(\beta^{\overline{C_{i'}}}_{1,\pi_{tr}^{\overline{C_{i'}}}(j')}, \ldots, \beta^{\overline{C_{i'}}}_{l,\pi_{tr}^{\overline{C_{i'}}}(j')}\right)\right) \in \Lambda',$$

       where for each $j \in [l]$, if $\lambda_j^{\overline{C_{i'}}} = 0$, then $c'_{j',j} = \alpha^{\overline{C_{i'}}}_{j,(\pi^{\overline{q_m}})^{-1}(j''_0)}$, otherwise, $c'_{j',j} = c_{j',j} + \alpha^{\overline{C_{i'}}}_{j,(\pi^{\overline{q_m}})^{-1}(j''_0)}$, where $j''_0 = \min(\{j'' \in [k] \mid j'' \sim_{q_m} j'\})$. In this case, after going through $C_{i'}$, the control variable $x_{j'}$ stores a fresh value and the initial value of $x_{j'}$ is not stored in any control variable, thus the $(j', \ldots)$-tuple is updated and a $(k+1, \ldots)$-tuple is added for the initial value of $x_{j'}$.
   - For each tuple $(k+1, (c_1, \ldots, c_l)) \in \Lambda$, we have $(k+1, (c'_1, \ldots, c'_l)) \in \Lambda'$, where for each $j \in [l]$, if $\lambda_j^{\overline{C_{i'}}} = 0$, then $c'_j = 0$, otherwise, $c'_j = c_j$. In addition, for each $j' \in [r^{\overline{C_{i'}}}] \setminus \mathsf{rng}(\pi_{tr}^{\overline{C_{i'}}})$, $(k+1, (\beta_{1,j'}^{\overline{C_{i'}}}, \ldots, \beta_{l,j'}^{\overline{C_{i'}}})) \in \Lambda'$.

By a simple analysis on the inductive computation of $\mathscr{A}$, we can show that all the tuples $(j', (c_1, \ldots, c_l))$ occurring in $\mathscr{A}$ satisfy that $c_1, \ldots, c_l$ are from a bounded domain U, which is stated in the following lemma.

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 the constant atom and the coefficients of the non-constant atoms in the expression $\Theta^{(C_{i_1}^{\ell_{i_1}},\Theta^{(H,\Omega_0)})^-}(y_j)$ or $\Theta^{(C_{i_1}C_{i_2},\Theta^{(H,\Omega_0)})^-}(y_j)$ for $i_1, i_2 \in [n]$ such that $i_1 \neq i_2$ and $\ell_{i_1} \in \{1, 2\}$.*

Note that although the set U can be exhausted by the cycle schemes stated in Lemma 3, the inductive computation of $\mathscr{A}$ may not be.

Step III. We first construct the set $\mathscr{A}$. Then for each $\Lambda \in \mathscr{A}$, do the following.

1. If $(0, (c_{0,1}, \ldots, c_{0,l})) \in \Lambda$ such that $a'_0 + b'_1 c_{0,1} + \cdots + b'_l c_{0,l} \neq 0$, then return true.
2. If there is $j' \in [k]$ such that $(\sum_{j'' \sim_{q_m} j'} a'_{j''}) + b'_1 c_{j',1} + \cdots + b'_l c_{j',l} \neq 0$, then return true, where $(j', (c_{j',1}, \ldots, c_{j',l})) \in \Lambda$.
3. If there is $(k+1, (c_1, \ldots, c_l)) \in \Lambda$ such that $b'_1 c_1 + \cdots + b'_l c_l \neq 0$, then return true.

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

In order to reduce the size of $\mathscr{A}$, we can restructure $\mathscr{A}$ into a pair $\mathscr{A}' = (\Xi, \Delta)$ as follows, without affecting the computation in Step III.

1. Initially, let $\Xi = \Delta := \emptyset$.
2. For each $\Lambda \in \mathscr{A}$, do the following: Let $\Lambda' := \Lambda \setminus \{(k+1, (c_1, \ldots, c_l)) \in \Lambda\}$. In addition, let $\Xi := \Xi \cup \{\Lambda'\}$ and $\Delta := \Delta \cup \{(k+1, (c_1, \ldots, c_l)) \in \Lambda\}$.

Complexity analysis of Step III. The size of the set U is polynomial in the size of generalized lasso. Then size of Ξ is at most exponential in kl and the size of Δ is at most exponential in l. Therefore, the size of $\mathscr{A}'$ is at most exponential in kl and the computation of $\mathscr{A}'$ takes exponential time in the worst case. 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 in kl, the product of the number of control and data variables.

Example 6. Let $\mathcal{S}'_{\mathsf{max}}$ be the SNT in Example 5. Then

$$\begin{aligned}
\Theta^{(C_1^\ell,\Omega)^-}(y_1) &= \Omega(y_1) + 0\Omega(x_1) + \mathfrak{d}_1^{\overline{C_{1,1}}} + \cdots + \mathfrak{d}_1^{\overline{C_{1,\ell}}}, \\
\Theta^{(C_1^\ell,\Omega)^-}(y_2) &= \Omega(y_2) + 0\Omega(x_1) + 2\mathfrak{d}_1^{\overline{C_{1,1}}} + \cdots + 2\mathfrak{d}_1^{\overline{C_{1,\ell}}}, \\
\Theta^{(C_1^\ell,\Omega)^-}(y_3) &= 3\mathfrak{d}_1^{\overline{C_{1,\ell}}},
\end{aligned}$$

and $\Theta^{(C_2^\ell,\Omega)^-}(y_j) = \Theta^{(C_2^\ell,\Omega)}(y_j)$ for each $j = 1, 2, 3$. The computation of $\mathscr{A}$ starts with the set $\mathscr{A}_0 = \{\mathsf{Abs}(HC_1), \mathsf{Abs}(HC_2)\}$. We illustrate how to compute $\mathsf{Abs}(HC_1)$ and $\mathsf{Abs}(HC_2)$.

- Since none of $\Theta^{(H,\Omega_0)}(y_j)$'s , $\Theta^{(C_1^\ell,\Omega)^-}(y_j)$'s and $\Theta^{(C_2^\ell,\Omega)^-}(y_j)$'s contains constant atoms, we know that $(0,(0,0,0))$ occurs in $\mathsf{Abs}(HC_1)$ and $\mathsf{Abs}(HC_2)$.
- After going through H, each of y_1, y_2, y_3 is assigned the first data value, and after going through HC_1, x_1 holds the first data value. From $\Theta^{(C_1^\ell,\Omega)^-}(y_j)$'s mentioned above, we know that each of $\Theta^{(C_1,\Theta^{(H,\Omega_0)^-})}(y_1)$ and $\Theta^{(C_1,\Theta^{(H,\Omega_0)^-})}(y_2)$ holds one copy of the first data value, and $\Theta^{(C_1,\Theta^{(H,\Omega_0)^-})}(y_3)$ contains no copies of the first data value. Therefore, $(1,(1,1,0))$ occurs in $\mathsf{Abs}(HC_1)$. Similarly, since x_1 holds the second data value after going through HC_2, we have $(1,(3,2,1))$ occurs in $\mathsf{Abs}(HC_2)$.
- In addition, by a simple calculation, we know that $\mathsf{Abs}(HC_1)$ contains another tuple $(2,(1,2,3))$ and $\mathsf{Abs}(HC_2)$ contains another tuple $(2,(2,4,6))$.

To summarize, we have $\mathsf{Abs}(HC_1) = \{(0,(0,0,0)),(1,(1,1,0)),(2,(1,2,3))\}$ and $\mathsf{Abs}(HC_2) = \{(0,(0,0,0)),(1,(3,2,1)),(2,(2,4,6))\}$. Then starting from $\mathscr{A}_0$, we compute $\mathscr{A}_1 = \mathscr{A}_0 \cup \{\mathscr{A}(HC_{i_1}C_{i_2}) \mid i_1, i_2 = 1,2\}$, and so on. We illustrate how to compute $\mathsf{Abs}(HC_1C_2)$ from $\mathsf{Abs}(HC_1)$.

- Since $(0,(0,0,0))$ occurs in $\mathsf{Abs}(HC_1)$ and $\lambda_1^{\overline{C_2}} = \lambda_2^{\overline{C_2}} = \lambda_3^{\overline{C_2}} = 1$, we know that $(0,(0,0,0)) \in \mathsf{Abs}(HC_1C_2)$.
- From $1 \in I_{tr}^{\overline{C_2}}$ and $\lambda_1^{\overline{C_2}} = \lambda_2^{\overline{C_2}} = \lambda_3^{\overline{C_2}} = 1$, we have $(2,(1+\alpha_{1,1}^{\overline{C_2}}, 1+\alpha_{2,1}^{\overline{C_2}}, 0+\alpha_{3,1}^{\overline{C_2}})) = (2,(1+1,1+3,0+5)) = (2,(2,4,5)) \in \mathsf{Abs}(HC_1C_2)$. In addition, we have $(1,(\beta_{1,1}^{\overline{C_2}},\ldots,\beta_{3,1}^{\overline{C_2}})) = (1,(3,2,1)) \in \mathsf{Abs}(HC_1C_2)$.
- Since $(2,(2,4,6))$ occurs in $\mathsf{Abs}(HC_1)$ and $\lambda_1^{\overline{C_2}} = \lambda_2^{\overline{C_2}} = \lambda_3^{\overline{C_2}} = 1$, we have $(2,(2,4,6)) \in \mathsf{Abs}(HC_1C_2)$. Moreover, because $[r^{\overline{C_2}}] \setminus \mathsf{rng}(\pi_{tr}^{\overline{C_2}}) = \emptyset$, no other tuples are added into $\mathsf{Abs}(HC_1C_2)$.

In summary,

$$\mathsf{Abs}(HC_1C_2) = \{(0,(0,0,0)),(1,(3,2,1)),(2,(2,4,5)),(2,(2,4,6))\}.$$

From the fact that $(1,(1,1,0))$ occurs in $\mathsf{Abs}(HC_1)$, we know that $a_1' + b_1' \times 1 + b_2' \times 1 + b_3' \times 0 = 0 + 1 \times 1 + (-2) \times 1 + 1 \times 0 = -1 \neq 0$. Therefore, Step III returns true and a non-zero output can be produced by following the path HC_1.

5.3 Decision procedure for SNTs

We generalize the decision procedure for the special case that the transition graphs of SNTs are generalized lassos to the full class of SNTs. We define a *generalized multi-lasso* as $\mathfrak{m} = H_1(C_{1,1},\ldots,C_{1,n_1})H_2(C_{2,1},\ldots,C_{2,n_2})\ldots H_r(C_{r,1},\ldots,C_{r,n_r})H_{r+1}$ s.t. (1) for each $s \in [r]$, $H_s = q_{s,1} \xrightarrow{(g_2,\eta_2)} q_{s,2}\ldots q_{s,m_s-1} \xrightarrow{(g_{m_s},\eta_{m_s})} q_{s,m_s}$ is a generalized lasso, $H_{r+1} = q_{r,m_r} \xrightarrow{(\mathsf{cur}=\triangleright,\eta')} q'$, (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.

In the following, we fix a generalized multi-lasso

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

and assume without loss of generality that $O(q') = a_0 + a_1x_1 + \cdots + a_kx_k + b_1y_1 + \cdots + b_ly_l$ and $O(q'')$ is undefined for every other state q'' in $\mathfrak{m}$. For convenience, we adapt the output function O a bit to define $O(q_{r,m_r}) := a'_0 + a'_1x_1 + \cdots + a'_kx_k + b'_1y_1 + \cdots + b'_ly_l$, where $a'_0 + a'_1x_1 + \cdots + a'_kx_k + b'_1y_1 + \cdots + b'_ly_l$ is obtained from $O(q')$ by replacing simultaneously each $z \in \mathsf{dom}(\eta')$ with $\eta'(z)$.

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 $[\![O(q_{s,m_s})]\!]_{\Theta^{(s,\Theta^{(H_1 \ldots H_s, \Omega_0)})}}$, 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 coefficients of atoms 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 as if we have a generalized lasso where the handle is $H_1 \ldots H_s$, the collection of cycles is $\{C_{s,1}, \ldots, C_{s,n_s}, \ldots, C_{r,1}, \ldots, C_{r,n_r}\}$, 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 $\mathfrak{s} = H_1\mathfrak{s}_1H_2 \ldots H_s\mathfrak{s}_s$ such that for each $s' \in [s]$, $\mathfrak{s}_{s'}$ is a cycle scheme over the collection of cycles $\{C_{s',1}, \ldots, C_{s',n_{s'}}\}$, 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 is 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 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 control and data variables in $\mathcal{S}$. In total, the complexity of the decision problem for the non-zero output problem of normalized SNTs is exponential in the number of control and data variables, as well as in the number of simple cycles, in the worst case.

6 Extensions

```
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. 5. More Challenging Examples of Reducers Performing Data Analytics Operations

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.

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. 6) is commutative.

We handle the division at the end of the program in Fig. 6 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. 6. 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. R. Alur and P. Cerny. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *POPL*, pages 599–610. ACM, 2011.
2. R. Alur, L. DAntoni, J. Deshmukh, M. Raghothaman, and Y. Yuan. Regular functions and cost register automata. In *LICS*, pages 13–22, 2013.
3. Y. Chen, C. Hong, N. Sinha, and B. Wang. Commutativity of reducers. In *TACAS*, pages 131–146, 2015.
4. Y. Chen, S. Lei, and Z. Wu. The commutativity problem of the mapreduce framework: A transducer-based approach. *CoRR*, abs/1605.01497, 2016.
5. H. Comon and Y. Jurski. Multiple counters automata, safety analysis and presburger arithmetic. In *CAV*, pages 268–279, 1998.

6. A. Finkel, S. Göller, and C. Haase. Reachability in register machines with polynomial updates. In *MFCS*, pages 409–420, 2013.
7. C. Haase and S. Halfon. Integer vector addition systems with states. In *RP*, pages 112–124, 2014.
8. Hadoop. `https://hadoop.apache.org`.
9. O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *J. ACM*, 25(1):116–133, Jan. 1978.
10. M. Kaminski and N. Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.
11. J. Leroux and G. Sutre. Flat counter automata almost everywhere! In *Software Verification: Infinite-State Model Checking and Static Program Analysis*, volume 06081 of *Dagstuhl Seminar Proceedings*, 2006.
12. M. L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1971.
13. F. Neven, N. Schweikardt, F. Servais, and T. Tan. Distributed streaming with finite memory. In *ICDT*, pages 324–341, 2015.
14. F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Logic*, 5(3):403–435, 2004.
15. K. Reinhardt. Reachability in petri nets with inhibitor arcs. *Electronic Notes in Theoretical Computer Science*, 223:239 – 264, 2008. RP 2008.
16. Spark. `http://spark.apache.org`.
17. M. Veanes, P. Hooimeijer, B. Livshits, D. Molnar, and N. Bjorner. Symbolic finite state transducers: Algorithms and applications. *ACM SIGPLAN Notices*, 47(1):137–150, 2012.
18. T. Xiao, J. Zhang, H. Zhou, Z. Guo, S. McDirmid, W. Lin, W. Chen, and L. Zhou. Nondeterminism in mapreduce considered harmful? an empirical study on non-commutative aggregators in mapreduce programs. In *ICSE*, pages 44–53, 2014.

A Formal Semantics of the Programming Language

Transitions	Side Condition
$(y := e; p, w, \rho) \longrightarrow (p, w, \rho')$	$\rho' = \rho[[\![e]\!]_\rho / y]$
$(y \mathrel{+}= e; p, w, \rho) \longrightarrow (p, w, \rho')$	$\rho' = \rho[[\![y + e]\!]_\rho / y]$
$(\text{if } (g)\{s_1\} \text{ [else } \{s_2\}]; p, w, \rho) \longrightarrow (s_1; p, w, \rho)$	$\rho \models g$
$(\text{if } (g)\{s_1\} \text{ [else } \{s_2\}]; p, w, \rho) \longrightarrow (s_2; p, w, \rho)$	$\rho \not\models g$
$(\mathsf{next}; p, w, \rho) \longrightarrow (p, \mathsf{tl}(w), \rho')$	$\rho' = \rho[\mathsf{hd}(w)/\mathsf{cur}]$ if $w \neq \varepsilon$
$(x := \mathsf{cur}; p, w, \rho) \longrightarrow (p, w, \rho')$	$\rho' = \rho[\rho(\mathsf{cur})/x]$
$(\text{loop}\{s\}; \text{ret } r, w, \rho) \longrightarrow (s; \text{loop}\{s\}; \text{ret } r, w, \rho)$	
$(\text{loop}\{s\}; \text{ret } r, \epsilon, \rho) \longrightarrow (\text{ret } r, \epsilon, \rho)$	

Fig. 7. The Semantics of the Programming Language

Formally, the semantics of a program p in the programming language is defined as a transition system in Fig. 7. Let p be a reducer program and w be an input data word. Each configuration of the transition system is a triple (p', w', ρ), where p' is a program, w' is a suffix of w, and ρ is a valuation over $X^+ \cup Y$ such that $\rho(\mathsf{cur}) = \mathsf{hd}(w')$ (where if $w' = \varepsilon$, then $\mathsf{hd}(w') = \perp$). Let ρ_0 denote a valuation which assigns each control and data variable an initial value, and ρ_w be the valuation such that $\rho_w(\mathsf{cur}) = \mathsf{hd}(w)$ and $\rho_w(z) = \rho_0(z)$ for each $z \in X \cup Y$. The initial configuration is $(p, \mathsf{tl}(w), \rho_w)$. We use $p_{\rho_0}(w)$ to denote the *output* of p on w wrt. ρ_0. Then $p_{\rho_0}(w) = d$ if there exists a path from the initial configuration (p, w, ρ_w) to some return configuration $(\text{ret } r, \epsilon, \rho_r)$ such that $[\![r]\!]_{\rho_r} = d$. Otherwise, $p_{\rho_0}(w) = \perp$. Since the program is deterministic, i.e., given an initial valuation ρ_0, each input data word has at most one output, the semantics of p is well-defined.

B Proofs in Section 4

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.*

Proof. We introduce a few notations first.

Let s be a loop-free program. An *execution path* π of s is a maximal path in the control flow graph of s (here we use the standard definition of control flow graphs). Each execution path π corresponds to a program s_π obtained by sequentially composing the statements in π, where the statements $\mathsf{assume}(g)$ are used to represent the guards g. Then s can be seen as a union of s_π, where π ranges over the execution paths of s.

Let p be a reducer program of the form $s_1; \mathsf{next}; \text{loop}\{s_2; \mathsf{next}; \}; \text{ret } r$. In the following, we show how to construct an SNT $\mathcal{S}_p$ to simulate p.

The loop body $s_2; \mathsf{next}$ can be seen as a union of programs p_π for execution paths π. We assume that no two distinct programs p_π share locations. We first transform

the loop into a collection of state-disjoint (except the state $\mathfrak{l}_0$, the entry point of the loop) cycles C_π, one for each program p_π. Let us focus on a program p_π. The set of states in C_π comprises the location $\mathfrak{l}_0$ which is the entry point of the loop, and the locations succeeding each next statement in p_π. Moreover, we identify the location succeeding the last next statement and the entry point. The effect of the subprogram s' between two successive next statements in the locations $\mathfrak{l}_1, \mathfrak{l}_2$ can be summarized into a transition $(\mathfrak{l}_1, g', \eta', \mathfrak{l}_2)$ of p_π. This is possible due to the following two constraints: 1) the conditions g in the statements if $(g)\{s'_1\}$ [else $\{s'_2\}$] of p_π are the conjunctions of $\mathsf{cur} \odot c$ and $\mathsf{cur} \odot x$, 2) the assignments to the control variables are of the form $x := \mathsf{cur}$, and the assignments to the data variables are of the form $y := e$ and $y+ = e$, where e contains only control variables or cur. As a result of the two constraints, we can trace the evolvement of the values of the control variables and simulate all the statements $\mathsf{assume}(g)$ occurring in s' by a guard g' (obtained from these guards g by some variable substitutions), moreover, the effects of all the assignments therein can be summarized into an assignment function η'. Similarly, we can do the same for the subprogram between the entry point and the first next statement of p_π.

In addition, each execution path of $s_1; \mathsf{next};$ can be simulated by a simple path of transitions of $\mathcal{S}_p$, which ends in the state $\mathfrak{l}_0$, the entry point of the loop.

The return statement t is handled by adding a transition with guards of the form $\mathsf{cur} = \triangleright$ from the state $\mathfrak{l}_0$ to a sink state $\mathfrak{l}'$. The output function O_p of $\mathcal{S}_p$ is defined as follow: $O_p(\mathfrak{l}') = r$ and $O(\mathfrak{l})$ is undefined for all the other states $\mathfrak{l}$.

Because in the program "$s_1; \mathsf{next}; \mathsf{loop}\{s_2; \mathsf{next};\}$; ret r", the subprogram s_2 contains no occurrences of next;, we know that each nontrivial SCC in $\mathcal{S}_p$ comprises a collection of self-loops around a unique state. Therefore, $\mathcal{S}_p$ is generalized-flat. □

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

Proof. 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 $\mathcal{S}_2$.

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

The correctness of this reduction follows from Proposition 1 in [3].

The construction of $\mathcal{S}_1$.

Intuitively, over a data word $w = d_1 d_2 d_3 \ldots d_n \triangleright$, we introduce an additional write-once variable x' to store d_1, then simulates the run of $\mathcal{S}$ over $d_2 d_1 d_3 \ldots d_n \triangleright$ as follows: When reading d_2 in w, the variables are updated properly by letting x' to represent d_1 and cur to represent d_2.

Let $q_0', q_1' \notin Q$ and $x' \notin X$. Then $\mathcal{S}_1 = (Q \cup \{q_0', q_1'\}, X \cup \{x'\}, Y, \delta_1, q_0', O_1)$ such that

- $O_1(q_0')$ and $O_1(q_1')$ are undefined, and for each $q \in Q$, $O_1(q) = O(q)$,
- δ_1 is constructed from δ as follows,
  - each element of δ is an element of δ_1,
  - for each pair of transitions $q_0 \xrightarrow{(g_1,\eta_1)} q_1 \xrightarrow{(g_2,\eta_2)} q_2$ in $\mathcal{S}$, we add the transitions $(q_0', \mathsf{cur} \neq \triangleright, \eta_1', q_1')$ and $(q_1', g' \wedge \mathsf{cur} \neq \triangleright, \eta_2', q_2)$ into δ_1. Intuitively, we use x' to store the value of d_1 in η_1' and summarize the computation of η_1 and η_2 in η_2' with the information that d_1 is stored in x' and d_2 is stored in cur.
    Formally, η_1', g', η_2' are defined as follows.
    - $\eta_1'(x') = \mathsf{cur}$ and $\eta_1'(z)$ is undefined for all $z \in X \cup Y$. This implies that after the transition $(q_0', \mathsf{true}, \eta_1', q_1')$, each variable $z \in X \cup Y$ still holds the initial value.
    - $g' = g_1 \wedge g_2'$, where g_2' is obtained from g_2 by replacing cur with x', and each $x \in X \cap \mathsf{dom}(\eta_1)$ with $\eta_1(x)$.
    - For each $z \in X \cup Y$, if $z \in \mathsf{dom}(\eta_2)$, let $\eta_2^r(z)$ be the expression obtained by replacing all occurrences of cur in $\eta_2(z)$ with x', then $\eta_2'(z)$ is obtained by substituting all occurrences of variables $z' \in \mathsf{dom}(\eta_1)$ in $\eta_2^r(z)$ with $\eta_1(z')$.
    - For each $z \in X \cup Y$, if $z \notin \mathsf{dom}(\eta_2)$, then $\eta_2'(z) = \eta_1(z)$.

The construction of $\mathcal{S}_2$.

Intuitively, over a data word $w = d_1 \dots d_n \triangleright$, we introduce an additional control variable x' to store d_1, then simulates the run of $\mathcal{S}$ over $d_2 \dots d_n d_1$: When reading $\triangleright$ in w, the variables are updated properly by letting x' to represent d_1.

Suppose $q_0' \notin Q$ and $x' \notin X$. Then $\mathcal{S}_2 = (\{q_0'\} \cup Q, X \cup \{x'\}, Y, \delta_2, q_0', O_2)$ such that

- $O_2(q_0')$ is undefined, and for each $q \in Q$, $O_2(q) = O(q)$,
- δ_2 is constructed from δ as follows,
  - each element of δ whose guard does not contain $\mathsf{cur} = \triangleright$ is an element of δ_2,
  - we add the transition $q_0' \xrightarrow{(\mathsf{true},\eta_1')} q_0$ to δ_2, where $\eta_1'(x') = \mathsf{cur}$ and $\eta_1'(z)$ is undefined for all $z \in X \cup Y$.
  - for each pair of transitions $q_1 \xrightarrow{(g_1 \wedge \mathsf{cur} \neq \triangleright, \eta_1)} q_2 \xrightarrow{(g_2 \wedge \mathsf{cur} = \triangleright, \eta_2)} q_3$ in $\mathcal{S}$, we add the transition (q_1, g', η_2', q_3) into δ_2. Intuitively, we use x' to store the value of d_1 in η_1' and summarize the computation of η_1 and η_2 in η_2' with the information that d_1 is stored in x'.
    Formally, g', η_2' are defined in the following.
    - $g' = g_1' \wedge g_2' \wedge (\mathsf{cur} = \triangleright)$, where g_1' and g_2' are obtained from g_1, g_2 as follows: g_1' is obtained from g_1 by replacing all occurrences of cur with x', and g_2' is obtained from g_2 by replacing each $x \in X \cap \mathsf{dom}(\eta_1)$ with $\eta_1(x)$, then substituting all occurrences of cur with x'.
    - For each $z \in X \cup Y$, if $z \in \mathsf{dom}(\eta_2)$, then $\eta_2'(z)$ is the expression obtained from $\eta_2(z)$ by replacing all occurrences of variables $z' \in \mathsf{dom}(\eta_1)$ therein with $\eta_1(z')$ and then substituting all occurrences of cur with x'.

- For each $z \in X \cup Y$, if $z \in \mathsf{dom}(\eta_1) \backslash \mathsf{dom}(\eta_2)$, then $\eta_2'(z)$ is the expression obtained from $\eta_1(z)$ by substituting all occurrences of cur with x'.
- For each $z \in X \cup Y$, if $z \notin \mathsf{dom}(\eta_2) \cup \mathsf{dom}(\eta_1)$, then $\eta_2'(z)$ is undefined.

It is easy to see that the size of both $\mathcal{S}_1$ and $\mathcal{S}_2$ are polynomial with respect to the size of $\mathcal{S}$. Note that $\mathcal{S}_1$ and $\mathcal{S}_2$ constructed above preserve the generalized-flatness and mononicity of $\mathcal{S}$, since the constructions do not modify the transitions in the nontrivial SCCs of the transition graph. □

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

Proof. Let $\mathcal{S}_1 = (Q_1, X_1, Y_1, \delta_1, q_{1,0}, O_1)$ and $\mathcal{S}_2 = (Q_2, X_2, Y_2, \delta_2, q_{2,0}, O_2)$ be two monotone SNTs. Without loss of generality, we assume that $Q_1 \cap Q_2 = \emptyset$, $X_1 \cap X_2 = \emptyset$, and $Y_1 \cap Y_2 = \emptyset$.

We first construct $\mathcal{S}$ as the product of $\mathcal{S}_1$ and $\mathcal{S}_2$. Specifically, $\mathcal{S} = (Q_1 \times Q_2, X_1 \cup X_2, Y_1 \cup Y_2, \delta, (q_{1,0}, q_{2,0}), O)$, where

- δ comprises $((q_1, q_2), g_1 \wedge g_2, \eta_1 \cup \eta_2, (q_1', q_2'))$ such that $(q_1, g_1, \eta_1, q_1') \in \delta_1$, $(q_2, g_2, \eta_2, q_2') \in \delta_2$, and $g_1 \wedge g_2$ is satisfiable,
- for each $(q_1, q_2) \in Q_1 \times Q_2$,
  - if $O_1(q_1)$ is defined and $O_2(q_2)$ is undefined or vice versa, then $O((q_1, q_2)) = 1$,
  - otherwise, if both $O_1(q_1)$ and $O_2(q_2)$ are defined, then $O((q_1, q_2)) = O_1(q_1) - O_2(q_2)$,
  - otherwise (both $O_1(q_1)$ and $O_2(q_2)$ are undefined), $O((q_1, q_2))$ is undefined.

From the aforementioned construction, it is easy to see that $\mathcal{S}_1$ and $\mathcal{S}_2$ are inequivalent iff there is a data word w such that the output of $\mathcal{S}$ over w is non-zero. Moreover, since both $\mathcal{S}_1$ and $\mathcal{S}_2$ are generalized-flat and monotone, we know that $\mathcal{S}$ is generalized-flat and monotone as well. □

Proposition 4. For each SNT $\mathcal{S}$, the nonzero-output problem of $\mathcal{S}$ can be reduced to a series of the nonzero-output problems of normalized SNTs $\mathcal{S}'$ in exponential time.

Proof. Suppose $\mathcal{S} = (Q, X, Y, \delta, q_0, O)$ is an SNT such that $X = \{x_1, \ldots, x_k\}$. Our goal is to reduce the nonzero-output problem of $\mathcal{S}$ to a series of nonzero-output problems of normalized SNTs $\mathcal{S}' = (Q', X, Y, \delta', q_0', O')$.

Let TPO_X denote the set of total preorders between control variables (Recall that a total preorder over X is reflexive and transitive relation $\preceq$ over X such that for each $x_i, x_j \in X$, either $(x_i, x_j) \in \preceq$ or $(x_j, x_i) \in \preceq$). For $\preceq \in \mathsf{TPO}_X$ and $x_i, x_j \in X$, x_i is said to be a $\preceq$-*successor* of x_j or x_j is said to be a $\preceq$-*predecessor* of x_i, if $(x_j, x_i) \in \preceq$, $(x_i, x_j) \notin \preceq$, and for each $x_{i'} \in X$ such that $(x_j, x_{i'}) \in \preceq$, it holds that $(x_i, x_{i'}) \in \preceq$.

Then for each $\preceq_0 \in \mathsf{TPO}_X$, we construct an SNT $\mathcal{S}' = (Q', X, Y, \delta', q_0', O')$ as follows: $Q' = Q \times \mathsf{TPO}_X$, and $q_0' = (q_0, \preceq_0)$. Moreover, O' is defined as follows: For each $(q, \preceq) \in Q'$, $O'((q, \preceq)) = O(q)$. It remains to define δ'.

The transition set δ' is defined by the following rules:

– For each $(q, g \wedge \mathsf{cur} \neq \triangleright, \eta, q') \in \delta$, δ' includes all the transitions $(q, \preceq) \xrightarrow{(g' \wedge \mathsf{cur} \neq \triangleright, \eta)} (q', \preceq')$ satisfying the following constraints.
  - g' is of one of the following forms,
    * $g' \equiv \varphi_{\preceq} \wedge \mathsf{cur} = x_i$, or
    * $g' \equiv \varphi_{\preceq} \wedge \mathsf{cur} > x_j \wedge \mathsf{cur} < x_i$ such that $x_i \in X$ is a $\preceq$-successor of $x_j \in X$, or
    * $g' \equiv \varphi_{\preceq} \wedge \mathsf{cur} > x_i$ such that there does not exist a $\preceq$-successor of x_i, or
    * $g' \equiv \varphi_{\preceq} \wedge \mathsf{cur} < x_i$ such that there does not exist a $\preceq$-predecessor of x_i,

    where $\varphi_{\preceq} \equiv \bigwedge_{i'<j'} \psi_{x_{i'},x_{j'}}$, and for each pair of indices $i', j' \in [k]$ such that $i' < j'$, if $(x_{i'}, x_{j'}), (x_{j'}, x_{i'}) \in \preceq$, then $\psi_{x_{i'},x_{j'}} \equiv x_{i'} = x_{j'}$, otherwise, if $(x_{i'}, x_{j'}) \in \preceq$ and $(x_{j'}, x_{i'}) \notin \preceq$, then $\psi_{x_{i'},x_{j'}} \equiv x_{i'} < x_{j'}$, otherwise, $\psi_{x_{i'},x_{j'}} \equiv x_{i'} > x_{j'}$.
  - g and g' are compatible, more precisely, $g \wedge g'$ is satisfiable (note that $\varphi_{\preceq}$, which encodes the information in $\preceq$, has been included in g').
  - $\preceq'$ is constructed as follows.
    * Case $g' \equiv \varphi_{\preceq} \wedge \mathsf{cur} = x_i$: At first, for each $x_{i'} \in X$, introduce a fresh variable $x'_{i'}$ to denote the value of $x_{i'}$ after the transition $(q, g \wedge \mathsf{cur} \neq \triangleright, \eta, q')$. Let X' denote the set of fresh variables. Let $\preceq''$ be the reflexive and transitive closure of the relation

      $$\preceq \cup \{(x_{i'}, x'_{i'}), (x'_{i'}, x_{i'}) \mid x_{i'} \in X \setminus \mathsf{dom}(\eta)\} \cup \{(\mathsf{cur}, x_i), (x_i, \mathsf{cur})\} \cup \{(x'_{i'}, x_{i''}), (x_{i''}, x'_{i'}) \mid \eta(x_{i'}) = x_{i''}\} \cup \{(x'_{i'}, \mathsf{cur}), (\mathsf{cur}, x'_{i'}) \mid \eta(x_{i'}) = \mathsf{cur}\}.$$

      Then $\preceq'$ is the total preorder obtained from $\preceq'' \cap X' \times X'$ by replacing each $x'_{i'} \in X'$ with $x_{i'} \in X$.
    * Case $g' \equiv \varphi_{\preceq} \wedge \mathsf{cur} > x_j \wedge \mathsf{cur} < x_i$: At first, for each $x_{i'} \in X$, introduce a fresh variable $x'_{i'}$ to denote the value of $x_{i'}$ after the transition $(q, g \wedge \mathsf{cur} \neq \triangleright, \eta, q')$. Let X' denote the set of fresh variables. Let $\preceq''$ be the reflexive and transitive closure of the relation

      $$\preceq \cup \{(x_{i'}, x'_{i'}), (x'_{i'}, x_{i'}) \mid x_{i'} \in X \setminus \mathsf{dom}(\eta)\} \cup \{(x_j, \mathsf{cur}), (\mathsf{cur}, x_i)\} \cup \{(x'_{i'}, x_{i''}), (x_{i''}, x'_{i'}) \mid \eta(x_{i'}) = x_{i''}\} \cup \{(x'_{i'}, \mathsf{cur}), (\mathsf{cur}, x'_{i'}) \mid \eta(x_{i'}) = \mathsf{cur}\}.$$

      Then $\preceq'$ is the total preorder obtained from $\preceq'' \cap X' \times X'$ by replacing each $x'_{i'} \in X'$ with $x_{i'} \in X$.
  - Case $g' \equiv \varphi_{\preceq} \wedge \mathsf{cur} > x_i$, or $g' \equiv \varphi_{\preceq} \wedge \mathsf{cur} < x_i$: Similar.

– For each $(q, g \wedge \mathsf{cur} = \triangleright, \eta, q') \in \delta$, δ' includes all the transitions $(q, \preceq) \xrightarrow{(\mathsf{cur} = \triangleright, \eta)} (q', \preceq')$ satisfying the following constraints.
  - $\preceq$ and g are compatible, more precisely, $\varphi_{\preceq} \wedge g$ is satisfiable, where $\varphi_{\preceq}$ is defined as above.
  - $\preceq'$ is constructed as follows. At first, for each $x_{i'} \in X$, introduce a fresh variable $x'_{i'}$ to denote the value of $x_{i'}$ after the transition $(q, g \wedge \mathsf{cur} = \triangleright, \eta, q)$. Let X' denote the set of fresh variables. Let $\preceq''$ be the reflexive and transitive closure of the relation $\preceq \cup \{(x_{i'}, x'_{i'}), (x'_{i'}, x_{i'}) \mid x_{i'} \in X \setminus \mathsf{dom}(\eta)\} \cup \{(x'_{i'}, x_{i''}), (x_{i''}, x'_{i'}) \mid \eta(x_{i'}) = x_{i''}\}$. Then $\preceq'$ is the total preorder obtained from $\preceq'' \cap X' \times X'$ by replacing each $x'_{i'} \in X'$ with $x_{i'} \in X$.

At first, from the construction, we know that $\mathcal{S}'$ is path-feasible, state-dominating, and $\triangleright$-transition-guard-tree. We then show that $\mathcal{S}'$ is generalized-flat. It is sufficient to prove that for each state q in some nontrivial SCC S of $\mathcal{S}$, there does not exist a nontrivial SCC in $\mathcal{S}'$ that includes at least two distinct states $(q, \preceq_1)$ and $(q, \preceq_2)$.

To the contrary, suppose that there are a state q in some nontrivial SCC S of $\mathcal{S}$ and two distinct states $(q, \preceq_1)$ and $(q, \preceq_2)$ in some nontrivial SCC of $\mathcal{S}'$.

Since $\preceq_1 \neq \preceq_2$, without loss of generality, we assume that there are a pair of distinct control variables x_i, x_j such that $(x_i, x_j) \in \preceq_1$ and $(x_i, x_j) \notin \preceq_2$. We introduce some notations first. For $x \in \{x_i, x_j\}$, we say that x computes the minimum (resp. maximum) value in S if whenever $\mathsf{cur} < x$ (resp. $\mathsf{cur} > x$) occurs in a transition (q, g, η, q) of S, it holds that $\eta(x) = \mathsf{cur}$. We distinguish between the following situations.

- Suppose that both x_i and x_j compute the minimum value in S. Since both x_i and x_j compute the minimum value in S, when starting from some configuration (q, ρ) such that $\rho \models x_i \leq x_j$ and keep applying the transitions in S, we know that in each transition,
  - either the current data value is less or equal to both x_i and x_j, then both x_i and x_j are assigned to the current data value and become equal,
  - or the current data value is greater than x_i and less or equal to x_j, then the current data value is assigned to x_j (with the value of x_i unchanged), then $x_i < x_j$ holds after the transition,
  - or the current data value is greater than both x_i and x_j, then both the value of x_i and that of x_j are unchanged.

  Therefore, when following a path from $(q, \preceq_1)$ to $(q, \preceq_2)$ in $\mathcal{S}'$, the fact $x_i \leq x_j$ persists. This implies that $(x_i, x_j) \in \preceq_2$, a contradiction.
- Suppose that both x_i and x_j compute the maximum value in S. Similarly to the arguments in the previous situation, we know that when following a path from $(q, \preceq_1)$ to $(q, \preceq_2)$ in $\mathcal{S}'$, the fact $x_i \leq x_j$ persists. This implies that $(x_i, x_j) \in \preceq_2$, a contradiction.
- Suppose that x_i computes the minimum value in S and x_j computes the maximum value in S. Since x_i computes the minimum value and x_j computes the maximum value in S, we know that the value of x_i is non-increasing and the value of x_j is non-decreasing. Therefore, when following a path from $(q, \preceq_1)$ to $(q, \preceq_2)$, the fact $x_i \leq x_j$ persists. This implies that $(x_i, x_j) \in \preceq_2$, a contradiction.
- Suppose that x_i computes the minimum value in S and x_j computes neither the minimum value nor the maximum value in S. Then the value of x_i is non-increasing and the value of x_j is unchanged when staying in S. The arguments are similar to the previous case.
- Suppose that x_i computes neither the minimum value nor the maximum value and x_j computes the maximum value in S. Then the value of x_i is unchanged and the value of x_j is non-decreasing when staying in S. The arguments are similar to the previous case.
- Suppose that x_i computes the maximum value in S and x_j computes the minimum value in S. Then in S, the value of x_i is non-decreasing and the value of x_j is non-increasing. From $(x_i, x_j) \in \preceq_1$ and $(x_i, x_j) \notin \preceq_2$, we know that when following a path from $(q, \preceq_1)$ to $(q, \preceq_2)$ in $\mathcal{S}'$, sometime the value of x_i becomes strictly greater

than that of x_j, and this fact persists afterwards. Therefore, we have $(x_j, x_i) \in \preceq_2$ and $(x_i, x_j) \notin \preceq_2$. Since in S, the value of x_i is non-decreasing and the value of x_j is non-increasing, we know that when following a path from $(q, \preceq_2)$ to $(q, \preceq_1)$ in $\mathcal{S}'$, $x_j < x_i$ persists. Therefore, $(x_j, x_i) \in \preceq_1$ and $(x_i, x_j) \notin \preceq_1$, a contradiction.
- Suppose that x_i computes the maximum value and x_j computes neither the minimum value nor the maximum value in S. Then the value of x_i is non-decreasing and the value of x_j is unchanged when staying in S. The arguments are similar to the previous case.
- Suppose that x_i computes neither the minimum value nor the maximum value in S and x_j computes the minimum value in S. Then the value of x_i is unchanged and the value of x_j is non-increasing when staying in S. The arguments are similar to the previous case.
- Suppose x_i (resp. x_j) computes neither the minimum value nor the maximum value in S. Then the value of x_i and x_j are unchanged when staying in S. Therefore, if $x_i \leq x_j$ holds in the state $(q, \preceq_1)$, then it holds in each state belonging to the same SCC as $(q, \preceq_1)$ in $\mathcal{S}'$. In particular, $(x_i, x_j) \in \preceq_2$, a contradiction.

Consequently, in each of the situations aforementioned, we always get a contradiction. We conclude that the assumption is false and $\mathcal{S}'$ is indeed generalized-flat. □

C Proofs in Section 5.1

Proposition 5. *Suppose that P is a path starting form p_0 and the initial values of $X \cup Y$ are represented by a symbolic valuation Ω such that for each pair of variables $x_i, x_j \in X$, $\Omega(x_j) = \Omega(x_j)$ iff $x_i \sim_{p_0} x_j$. 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 $I^{\overline{P}}_{pe}$ and $I^{\overline{P}}_{tr}$, the indices of* persistent *and* transient *control variables, respectively. A control variable is persistent if it stores the initial value of some control variable after traversing P, otherwise, it is transient.*
- *For each $x_j \in X$ such that $j \in I^{\overline{P}}_{pe}$, $\Theta^{(P,\Omega)}(x_j) = \Omega(x_{\pi^{\overline{p_0}}(\pi^{\overline{P}}_{pe}(j))})$, where $\pi^{\overline{P}}_{pe} : I^{\overline{P}}_{pe} \to [s^{\overline{p_0}}]$ is a mapping from the index of a persistent control variable x_j to the index of the equivalence class such that the initial value of control variables corresponding to this equivalence class is assigned to x_j after traversing P.*
- *For each $x_j \in X$ such that $j \in I^{\overline{P}}_{tr}$, $\Theta^{(P,\Omega)}(x_j) = \mathfrak{d}^{\overline{P}}_{\pi^{\overline{P}}_{tr}(j)}$, where $\pi^{\overline{P}}_{tr} : I^{\overline{P}}_{tr} \to [r^{\overline{P}}]$ is a mapping from the index of a transient control variable to the index of the data value assigned to it.*
- *For each $y_j \in Y$,*

$$\Theta^{(P,\Omega)}(y_j) = \varepsilon^{\overline{P}}_j + \lambda^{\overline{P}}_j \Omega(y_j) + \sum_{j' \in [s^{\overline{p_0}}]} \alpha^{\overline{P}}_{j,j'} \Omega(x_{\pi^{\overline{p_0}}(j')}) + \sum_{j'' \in [r^{\overline{P}}]} \beta^{\overline{P}}_{j,j''} \mathfrak{d}^{\overline{P}}_{j''},$$

where $\varepsilon^{\overline{P}}_j, \lambda^{\overline{P}}_j, \alpha^{\overline{P}}_{j,1}, \ldots, \alpha^{\overline{P}}_{j,s^{\overline{p_0}}}, \beta^{\overline{P}}_{j,1}, \ldots, \beta^{\overline{P}}_{j,r^{\overline{P}}}$ are integer constants such that $\lambda^{\overline{P}}_j \in \{0, 1\}$ (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.

Proof. Suppose $\mathcal{S} = (Q, X, Y, \delta, q_0, O)$ is a normalized SNT and $P = p_0 \xrightarrow{(g_1,\eta_1)} p_1 \dots p_{n-1} \xrightarrow{(g_n,\eta_n)} p_n$ 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 $X \cup Y$ such that for each pair of variables $x_i, x_j \in X$, $\Omega(x_j) = \Omega(x_j)$ iff $x_i \sim_{p_0} x_j$.

We show by an induction that for each $i : 1 \leq i \leq n$, a symbolic valuation Θ_i over $X^+ \cup Y$ can be constructed to describe the value of x_j (resp. y_j) after going through the first i transitions of P. Moreover, an index set $I_i \subseteq [k]$ is computed as well.

- At first, compute Θ_0 and I_0 as follows.
  1. For each $x_j \in X$, $\Theta_0(x_j) := \Omega(x_{j'_0})$, where $j'_0 = \min(\{j' \in [k] \mid j' \sim_{p_0} j\})$.
  2. If $g_1 \models \mathsf{cur} = x_j$ for some $x_j \in X$, then $\Theta_0(\mathsf{cur}) := \Theta_0(x_j)$ and $s := 0$, otherwise, $\Theta_0(\mathsf{cur}) := \mathfrak{d}_1^{\overline{P}}$ and $s := 1$.
  3. For each $y_j \in Y$, $\Theta_0(y_j) := \Omega(y_j)$.
  4. In addition, let $I_0 = \emptyset$.
- Let $i : 1 \leq i \leq n$. Then Θ_i and I_i are computed as follows:
  1. Initially, let $I_i := \emptyset$.
  2. For each $x_j \in X$, we distinguish among the following situations,
     - if $x_j \notin \mathsf{dom}(\eta_i)$, then $\Theta_i(x_j) := \Theta_{i-1}(x_j)$, in addition, if $j \in I_{i-1}$, let $I_i := I_i \cup \{j\}$,
     - if $x_j \in \mathsf{dom}(\eta_i)$, in addition, either $\eta_i(x_j) = x_{j'}$ for some $x_{j'} \in X$, or $\eta_i(x_j) = \mathsf{cur}$ and $\varphi_{q_{i-1}} \wedge g_i \models \mathsf{cur} = x_{j'}$ for some $x_{j'} \in X$, then let $\Theta_i(x_j) := \Theta_{i-1}(x_{j'})$, in addition, if $x_{j'} \in I_{i-1}$, then let $I_i := I_i \cup \{j\}$,
     - if $\eta_i(x_j) = \mathsf{cur}$ and there do not exist $x_{j'} \in X$ such that $\varphi_{q_{i-1}} \wedge g_i \models \mathsf{cur} = x_{j'}$, then let $\Theta_i(x_j) := \Theta_{i-1}(\mathsf{cur})$ and $I_i := I_i \cup \{j\}$.
  3. Compute $\Theta_i(\mathsf{cur})$ as follows:
     - If $i < n$ and there exists $x_j \in X$ such that $\varphi_{q_i} \wedge g_{i+1} \models \mathsf{cur} = x_j$, then let $\Theta_i(\mathsf{cur}) := \Theta_i(x_j)$.
     - If $i < n$ and there do not exist $x_j \in X$ such that $\varphi_{q_i} \wedge g_{i+1} \models \mathsf{cur} = x_j$, then let $s := s + 1$ and $\Theta_i(\mathsf{cur}) := \mathfrak{d}_s^{\overline{P}}$.
     - If $i = n$, then let $\Theta_i(\mathsf{cur}) := \bot$.
  4. For each $y_j \in Y$, if $y_j \in \mathsf{dom}(\eta_i)$, then let $\Theta_i(y_j) := [\![\eta_i(y_j)]\!]_{\Theta_{i-1}}$, otherwise, let $\Theta_i(y_j) := \Theta_{i-1}(y_j)$.

Then let $I_{tr}^{\overline{P}} := I_n$, $I_{pe}^{\overline{P}} := [k] \setminus I_{tr}^{\overline{P}}$, and $r^{\overline{P}} := s$. The mapping $\pi_{pe}^{\overline{P}}$ and $\pi_{tr}^{\overline{P}}$ are defined as follows:

- For each $j \in I_{pe}^{\overline{P}}$, let $x_{j'} \in X$ such that $\Theta_n(x_j) = \Omega(x_{j'})$, then $\pi_{pe}^{\overline{P}}(j) := (\pi^{\overline{p_0}})^{-1}(j')$.
- For each $j \in I_{tr}^{\overline{P}}$, let $s' \in [r^{\overline{P}}]$ such that $\Theta_n(x_j) = \mathfrak{d}_{s'}^{\overline{P}}$, let $\pi_{tr}^{\overline{P}}(j) := s'$.

The symbolic valuation $\Theta^{(P,\Omega)}$ can be defined as the restriction of Θ_n to $X \cup Y$. Since for each assignment η_i and $y_j \in Y$, $\eta_i(y_j) = e$ or $\eta_i(y_j) = y_j + e$ for $e \in \mathcal{E}_{X^+}$, it follows that $\Theta^{(P,\Omega)}(y_j)$ is of the form required by the proposition. □

Proposition 6. *Suppose that C is a simple cycle (i.e. a self-loop around a state q) 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:*

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

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

Proof. We prove by an induction on ℓ that $\Theta^{(C^\ell,\Omega)}(y_j)$ is of the desired form required by the proposition.

The induction base: $\ell = 2$.

Let $\mathfrak{d}_1^{(\overline{C,2})},\ldots,\mathfrak{d}_{r^{\overline{C}}}^{(\overline{C,2})}$ be the data values introduced when traversing the cycle for the second time. Then from Corollary 2, we know that $\Theta^{(C^2,\Omega)} = \Theta^{(C,\Theta^{(C,\Omega)})}$ is defined as follows: For each $y_j \in Y$,

$$\begin{aligned}\Theta^{(C^2,\Omega)}(y_j) = &\left(\varepsilon_j^{\overline{C}} + \lambda_j^{\overline{C}}\varepsilon_j^{\overline{C}}\right) + \left(\lambda_j^{\overline{C}}\right)^2\Omega(y_j) + \sum_{j'\in\mathsf{rng}(\pi_{pe}^{\overline{C}})}\left(1+\lambda_j^{\overline{C}}\right)\alpha_{j,j'}^{\overline{C}}\Omega(x_{\pi^{\overline{q}}(j')}) \\ &+ \sum_{j'\in[s^{\overline{q}}]\setminus\mathsf{rng}(\pi_{pe}^{\overline{C}})}\lambda_j^{\overline{C}}\alpha_{j,j'}^{\overline{C}}\Omega(x_{\pi^{\overline{q}}(j')}) \\ &+ \sum_{j'\in\mathsf{rng}(\pi_{tr}^{\overline{C}})}\left(\lambda_j^{\overline{C}}\beta_{j,j'}^{\overline{C}} + \sum_{j''\in(\pi_{tr}^{\overline{C}})^{-1}(j')\cap\mathsf{rng}(\pi^{\overline{q}})}\alpha_{j,(\pi^{\overline{q}})^{-1}(j'')}^{\overline{C}}\right)\mathfrak{d}_{j'}^{\overline{C,1}} \\ &+ \sum_{j'\in[r^{\overline{C}}]\setminus\mathsf{rng}(\pi_{tr}^{\overline{C}})}\left(\lambda_j^{\overline{C}}\beta_{j,j'}^{\overline{C}}\right)\mathfrak{d}_{j'}^{\overline{C,1}} + \sum_{j'\in[r^{\overline{C}}]}\beta_{j,j'}^{\overline{C}}\mathfrak{d}_{j'}^{\overline{C,2}}.\end{aligned}$$

Induction step: Let $\ell \geq 3$.

From the induction hypothesis, we know that for each $y_j \in Y$, $\Theta^{(C^{\ell-1},\Omega)}(y_j)$ is of the desired form.

From Corollary 2, $\Theta^{(C^\ell,\Omega)} = \Theta^{(C,\Theta^{(C^{\ell-1},\Omega)})}$. Then for each $y_j \in Y$, by unfolding the expressions $\Theta^{(C^{\ell-1},\Omega)}(x_{j'})$ for $j' \in [k]$ and $\Theta^{(C^{\ell-1},\Omega)}(y_{j''})$ for $j'' \in [l]$ in $\Theta^{(C,\Theta^{(C^{\ell-1},\Omega)})}(y_j)$, we can observe that $\Theta^{(C^\ell,\Omega)}(y_j)$ is of the desired form. □

D Proofs in Section 5.2

Lemma 1. *Suppose $\mathfrak{s} = C_{i_1}^{\ell_1} C_{i_2}^{\ell_2} \dots C_{i_t}^{\ell_t}$ is a cycle scheme, and Ω is a symbolic valuation representing the initial values of the control and data variables such that for each $x_i, x_j \in X$, $\Omega(x_i) = \Omega(x_j)$ iff $i \sim_{q_m} j$. For all $j' \in I_{pe}^{\overline{C_{i_1}}} \cap \mathsf{rng}(\pi^{\overline{q_m}})$, 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 $C_{i_1}^{\ell_1} C_{i_2}^{\ell_2} \dots C_{i_{r_{j'}}}^{\ell_{r_{j'}}}$. Then for each $j \in [l]$ and $j' \in I_{pe}^{\overline{C_{i_1}}} \cap \mathsf{rng}(\pi^{\overline{q_m}})$, the coefficient of the $\Omega(x_{j'})$-atom in $\Theta^{(\mathfrak{s},\Omega)}(y_j)$ is*

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

where (1) $e = 0$ when $r_{j'} = t$ and (2) $e = (\lambda_j^{\overline{C_{i_s}}})^{\ell_s-1} \alpha_{j,(\pi^{\overline{q_m}})^{-1}(j')}^{\overline{C_{i_s}}} \prod_{s'\in[s+1,t]} \left(\lambda_j^{\overline{C_{i_{s'}}}}\right)^{\ell_{s'}}$ with $s = r_{j'} + 1$ when $r_{j'} < t$.
The constant atom of $\Theta^{(\mathfrak{s},\Omega)}(y_j)$ is

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

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 $j' \in I_{pe}^{\overline{C_{i_1}}} \cap \mathsf{rng}(\pi^{\overline{q_m}})$ contain a subexpression of the form $\mu_{\mathfrak{s}}\ell_1$ for some $\mu_{\mathfrak{s}} \in \mathbb{Z}$.

Proof. The lemma can be shown by applying Proposition 6, Corollary 2, and an induction on the length t of the cycle schemes. □