A Decidable Extension of Data Automata*

Zhilin Wu

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

wuzl@ios.ac.cn

Data automata on data words is a decidable model proposed by Bojańczyk et al. in 2006. Class automata, introduced recently by Bojańczyk and Lasota, is an extension of data automata which unifies different automata models on data words. The nonemptiness of class automata is undecidable, since class automata can simulate two-counter machines. In this paper, a decidable model called class automata with priority class condition, which restricts class automata but strictly extends data automata, is proposed. The decidability of this model is obtained by establishing a correspondence with priority multicounter automata. This correspondence also completes the picture of the links between various class conditions of class automata and various models of counter machines. Moreover, this model is applied to extend a decidability result of Alur, Cerný and Weinstein on the algorithmic analysis of array-accessing programs.

1 Introduction

With the momentums from the XML document processing and the statical analysis and verification of programs, formalisms over infinite alphabets are becoming a research focus of theoretical computer science (c.f. [6] for a survey).

The infinite alphabet means $\Sigma \times \mathbb{D}$, with Σ a finite tag set and $\mathbb{D}$ an infinite data domain. Words and trees with the labels of nodes from the infinite alphabet $\Sigma \times \mathbb{D}$ are called data words and data trees. Formally, a data word is a pair (w, π), with w denoting the sequence of tags and π denoting the corresponding sequence of data values. Data trees can be defined similarly.

Among various models of logic and automata over infinite alphabets that have been proposed, data automata were introduced by Bojańczyk et al. in 2006 to prove the decidability of two-variable logic on data words ([4]).

A data automaton $\mathscr{D}$ consists of two parts, a nondeterministic letter-to-letter transducer $\mathscr{A} : \Sigma^* \to \Gamma^*$, and a class condition which is a finite automaton $\mathscr{B}$ with the alphabet Γ. $\mathscr{D}$ accepts a data word (w, π) iff from w, $\mathscr{A}$ is able to produce a Γ-string w' such that,

> for each class X of (w, π) (a class of a data word is a maximal set of positions with the same data value), $\mathscr{B}$ has an accepting run over $w'|_X$ (the restriction of w' to the positions in X).

Several extensions of data automata have appeared in the literature.

Extended data automata, was proposed by Alur, Cerný and Weinstein in 2009, in order to analyze the array-accessing programs ([1]). Extended data automata extend data automata by the class condition, which is now a finite automaton $\mathscr{B}$ with the alphabet $\Gamma \cup \{0\}$. $\mathscr{D}$ accepts a data word (w, π) iff from w, $\mathscr{A}$ is able to produce a Γ-string w' such that,

*This work was done while the author was a postdoc at LaBRI, Université Bordeaux 1, France.

To appear in EPTCS.

> for each class X of (w,π), $\mathscr{B}$ has an accepting run over $w' \oplus X$, where $w' \oplus X$ is the string in $(\Gamma \cup \{0\})^*$ obtained from w' by replacing each letter w'_i such that $i \notin X$ by 0 (note that $w' \oplus X$ has the same length as w').

However, as shown in [1], it turns out that extended data automata are expressively equivalent to data automata, thus they are a syntactic extension, but not a semantic extension of data automata.

Another extension of data automata, class automata, was proposed by Bojańczyk and Lasota in 2010 to capture the full XPath, including forward and backward modalities and all types of data tests ([3]).

Class automata generalize both data automata and extended data automata by the class condition, which is now a finite automaton $\mathscr{B}$ with the alphabet $\Gamma \times \{0,1\}$. $\mathscr{D}$ accepts a data word (w,π) iff from w, $\mathscr{A}$ is able to produce a Γ-string w' such that,

> for each class X of (w,π), $\mathscr{B}$ has an accepting run over $w' \otimes X$, where $w' \otimes X$ is the string in $(\Gamma \times \{0,1\})^*$ obtained from w' by replacing each letter w'_i by $(w'_i,1)$ if $i \in X$, and by $(w'_i,0)$ otherwise.

In [3], Bojańczyk and Lasota also defined various class conditions of class automata and established their correspondences with different models of counter machines, including multicounter machines with or without zero tests, counter machines with increasing errors, and Presburger automata.

Besides the models of counter machines considered in [3], there is still another type of counter machines, called priority multicounter automata, proposed by Reinhardt in his Habilitation thesis ([5]), where he showed that the nonemptiness of priority multicounter automata is decidable. Priority multicounter automata were also used by Björklund and Bojanczyk to prove the decidability of two-variable first order logic over data trees of bounded depth ([2]).

A priority multicounter automaton (PMA) is a multicounter automaton M with the restricted zero tests: The n counters in M are ordered as $C_1, \dots, C_n$. M can select an index $i \leq n$, and test whether for each $j \leq i$, $C_j = 0$.

In this paper, we propose a new type of class condition for class automata, called *priority class condition*, and show its correspondence with priority multicounter automata, thus showing the decidability as well as completing the picture of the links between class automata and counter machines established by Bojańczyk and Lasota.

The main idea of the priority class condition of class automata is roughly as follows:

> Let $\mathscr{D} = (\mathscr{A}, \mathscr{B})$ be a class automaton such that the output alphabet of the transducer $\mathscr{A}$ is Γ. Then a priority class condition is obtained by putting an order (priority) over the letters $\gamma \in \Gamma$ and using this order to restrict the $(\gamma,0)$-transitions of $\mathscr{B}$.

In this sense, a data automaton is a class automaton with priority class condition (PCA) in which all the $(\gamma,0)$-transitions are self-loops, while an extended data automaton is a PCA in which the different γ's are non-distinguishable in $(\gamma,0)$-transitions.

With respect to the closure properties, we show that PCAs are closed under letter projection and union, but not under intersection nor complementation. While data automata (and the expressively equivalent extended data automata) are closed under letter projection, union and intersection, it turns out that PCAs strictly extend data automata and still preserve the decidability.

In addition, we demonstrate the usefulness of PCAs by applying them to generalize a decidability result of Alur, Cerný and Weinstein on the analysis of array-accessing programs ([1]).

This paper is organized as follows. In Section 2, some preliminaries are given. Then in Section 3, the concepts of 0-priority finite automata and 0-priority regular languages are introduced and PCA is defined. In Section 4, the correspondence between PCA and PMA is established. Section 5 discusses the application of PCAs to the algorithmic analysis of array-accessing programs.

2 Preliminaries

In this paper, we fix a finite tag set Σ and an infinite data domain $\mathbb{D}$, e.g. the set of natural numbers $\mathbb{N}$.

A *word* w over Σ is a function from $[n] = \{1,\dots,n\}$ to Σ for some $n \geq 1$. Suppose $w : [n] \to \Sigma$ is a word, then $|w|$ is used to denote the length of w, namely n. If in addition $X \subseteq [n]$, then $w|_X$ is used to denote the subword of w restricted to the set of positions in X. A *language* is a set of words.

A *data word* is a pair (w,π), where w is a word in Σ^* of length n and $\pi : [n] \to \mathbb{D}$. A *class* of a data word (w,π) (of length n) corresponding to a data value $d \in \mathbb{D}$ is a collection of all the positions $i \in [n]$ such that $\pi(i) = d$. For instance, the class of the data word $(a,0)(b,1)(c,0)$ corresponding to the data value 0 is $\{1,3\}$. A *data language* is a set of data words. Let L be a data language, the language of words corresponding to L, denoted by $str(L)$, is $\{w \mid (w,\pi) \in L\}$.

A *data automaton* $\mathscr{D}$ consists of two parts,

- a nondeterministic letter-to-letter transducer $\mathscr{A} : \Sigma^* \to \Gamma^*$,
- and a class condition, which is a finite automaton $\mathscr{B}$ over the alphabet Γ.

A data automaton $\mathscr{D} = (\mathscr{A},\mathscr{B})$ accepts a data word (w,π) iff from w, $\mathscr{A}$ is able to produce a string $w' \in \Gamma^*$ (with the same length as w) such that for each class X of (w,π), $\mathscr{B}$ has an accepting run over $w'|_X$. The set of data words accepted by $\mathscr{D}$ is denoted by $\mathscr{L}(\mathscr{D})$.

Class automata $\mathscr{D} = (\mathscr{A},\mathscr{B})$ is an extension of data automata with the class condition changed into a finite automaton $\mathscr{B}$ over the alphabet $\Gamma \times \{0,1\}$.

A class automaton $\mathscr{D} = (\mathscr{A},\mathscr{B})$ accepts a data word (w,π) iff from w, $\mathscr{A}$ is able to produce a Γ-string w' such that for each class X of (w,π), $\mathscr{B}$ has an accepting run over $w' \otimes X$, where $w' \otimes X \in (\Gamma \times \{0,1\})^*$ is obtained from w' by replacing each letter w'_i by $(w'_i,1)$ if $i \in X$, and by $(w'_i,0)$ otherwise, e.g. if $w' = abc$ and $X = \{1,3\}$, then $w' \otimes X = (a,1)(b,0)(c,1)$. The set of data words accepted by $\mathscr{D}$ is denoted by $\mathscr{L}(\mathscr{D})$.

A *multicounter automaton* $\mathscr{C}$ is a hexa-tuple $(Q,\Sigma,k,\delta,q_0,F)$ such that

- Q is a finite set of states,
- Σ is the finite alphabet,
- k is the number of counters,
- $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times L \times Q$ is the set of transition relations over the instruction set $L = \{inc_i, dec_i, ifz_i \mid 1 \leq i \leq k\}$,
- q_0 is the initial state,
- F is the set of accepting states.

Let $\mathscr{C} = (Q,\Sigma,k,\delta,q_0,F)$ be a multicounter automaton. A *configuration* of $\mathscr{C}$ is a state together with a list of counter values, namely, an element from $Q \times \mathbb{N}^k$. A configuration $(q',\overline{c'})$ is said to be an *immediate successor* of $(q,\bar{c})$ induced by a letter $\sigma \in \Sigma \cup \{\varepsilon\}$ and an instruction $l \in L$, denoted as $(q,\bar{c}) \xrightarrow{\sigma,l} (q',\overline{c'})$, if $(q,\sigma,l,q') \in \delta$ and

- if $l = inc_i$, then $c'_i = c_i + 1$ and $c'_j = c_j$ for $j \neq i$,
- if $l = dec_i$, then $c_i > 0$, $c'_i = c_i - 1$, and $c'_j = c_j$ for $j \neq i$,
- if $l = ifz_i$, then $c_i = 0$ and $c'_j = c_j$ for each $j : 1 \leq j \leq k$.

A *run* of $\mathscr{C}$ over a word w is a nonempty sequence $(q_0,\overline{c_0}) \xrightarrow{\sigma_1,l_1} (q_1,\overline{c_1}) \xrightarrow{\sigma_2,l_2} \dots \xrightarrow{\sigma_n,l_n} (q_n,\overline{c_n})$ such that $w=\sigma_1\dots\sigma_n$. A run is *accepting* if $q_n \in F$. $\mathscr{C}$ accepts a word w if there is an accepting run of $\mathscr{C}$ over w.

A *priority multicounter automaton* (abbreviated as PMA) is a multicounter automaton $\mathscr{C}$ with the following restricted zero tests:

> The k counters in $\mathscr{C}$ are ordered as $C_1,\dots,C_k$. $\mathscr{C}$ can select some index $i \le k$, and test whether for each $j \le i$, the counter C_j has value 0.

Namely, a priority multicounter automaton is the same as a multicounter automaton, except that the instruction set L is changed into $\{inc_i, dec_i, ifz_{\le i} \mid 1 \le i \le k\}$.

Theorem 1 ([5]). *The nonemptiness of priority multicounter automata is decidable.*

3 Class automata with priority class condition

Intuitively, class automata with priority class condition are obtained from class automata by restricting the class condition to 0-priority regular languages defined in the following.

We first introduce several notations.

Let $\mathscr{B}=(Q,\Gamma\times\{0,1\},\delta,q_0,F)$ be a *deterministic complete* finite automaton over the alphabet $\Gamma\times\{0,1\}$. We use the notation $q \xrightarrow{(\gamma,b)} q'$ to denote the fact that $\delta(q,(\gamma,b))=q'$, where $b=0,1$, and $q \xrightarrow{*} q'$ to denote the fact that q' is reachable from q in the transition graph of $\mathscr{B}$. The transitions $q \xrightarrow{(\gamma,1)} q'$ (resp. $q \xrightarrow{(\gamma,0)} q'$) are called the one-transitions (resp. zero-transitions) of $\mathscr{B}$.

Let G_0 be the directed subgraph of the transition graph (Q,δ) obtained from (Q,δ) by restricting the set of arcs to those labeled by letters from $\Gamma\times\{0\}$. Formally, $G_0=(Q,\delta\cap(Q\times(\Gamma\times\{0\})\times Q))$. We use the notation $q \xrightarrow[0]{*} q'$ to denote the fact that q' is reachable from q in G_0.

A state $q\in Q$ is called *0-cyclic* if q belongs to some nontrivial (containing at least one arc) strongly-connected component (SCC) C of G_0. Otherwise q is called *0-acyclic*.

For each $\gamma\in\Gamma$, let $G_{(\gamma,0)}$ be the directed subgraph of (Q,δ) obtained from (Q,δ) by restricting the set of arcs to those labeled by $(\gamma,0)$. Formally, $G_{(\gamma,0)}=(Q,\delta\cap(Q\times\{(\gamma,0)\}\times Q))$. The out-degree of each vertex in $G_{(\gamma,0)}$ is exactly one, thus it has a simple structure: Each connected component of $G_{(\gamma,0)}$ consists of a unique cycle and a set of directed paths towards that cycle.

Let $\gamma\in\Gamma$. The cycles in $G_{(\gamma,0)}$ are called the $(\gamma,0)$-*cycles* of $\mathscr{B}$. If a state q belongs to some $(\gamma,0)$-cycle in $G_{(\gamma,0)}$, then q is called a $(\gamma,0)$-*cyclic* state, otherwise, it is called a $(\gamma,0)$-*acyclic* state of $\mathscr{B}$. Note that $(\gamma,0)$-acyclic states may be 0-cyclic.

Example 2. *An example of the deterministic complete automaton $\mathscr{B}$ over the alphabet $\{a,b\}\times\{0,1\}$ is given in Figure 1(a). Its associated G_0 and $G_{(b,0)}$ are given in Figure 1(b) and Figure 1(c) respectively. The state q_0 and q_2 are both 0-cyclic and $(b,0)$-cyclic, while q_1 is 0-cyclic but $(b,0)$-acyclic, since q_1 belongs to a cycle in G_0 and does not belong to any cycle in $G_{(b,0)}$.*

Definition 3 ($((\gamma_1,0),(\gamma_2,0))$-pattern). *Let $\gamma_1,\gamma_2\in\Gamma$. A* $((\gamma_1,0),(\gamma_2,0))$-pattern *in $\mathscr{B}$ is a state-tuple (q_1,q_2,q_3,q_4) such that $q_1 \xrightarrow{(\gamma_1,0)} q_2 \xrightarrow[0]{*} q_3 \xrightarrow{(\gamma_2,0)} q_4$, q_1 is 0-cyclic, and q_3 is $(\gamma_2,0)$-acyclic.*

Example 4. *For the automaton $\mathscr{B}$ in Figure 1(a), because $q_1 \xrightarrow{(a,0)} q_1 \xrightarrow[0]{*} q_1 \xrightarrow{(b,0)} q_0$, q_1 is 0-cyclic and $(b,0)$-acyclic, it follows that (q_1,q_1,q_1,q_0) is a $((a,0),(b,0))$-pattern in $\mathscr{B}$.*

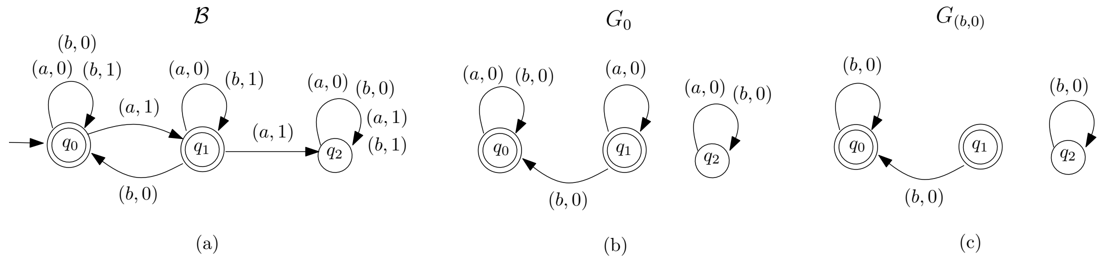

Figure 1: Automaton $\mathscr{B}$, G_0 and $G_{(b,0)}$

Definition 5 (0-priority finite automata and 0-priority regular languages)**.** *Let $\mathscr{B}$ be a finite automaton over the alphabet $\Gamma \times \{0,1\}$. Then $\mathscr{B}$ is called a* 0*-priority finite automaton if $\mathscr{B}$ is a* deterministic complete *automaton such that*

> *the letters in Γ can be ordered as a sequence $\gamma_1\gamma_2\dots\gamma_k$ satisfying that there are no $((\gamma_i,0),(\gamma_j,0))$-patterns with $i \geq j$ in $\mathscr{B}$.*

A regular language $L \subseteq (\Gamma \times \{0,1\})^*$ is called a 0*-priority regular language if there is a* 0*-priority finite automaton $\mathscr{B}$ over the alphabet $\Gamma \times \{0,1\}$ accepting L.*

Now we state several properties of 0-priority finite automata and 0-priority regular languages.

Proposition 6. *Let $\mathscr{B} = (Q, \Gamma \times \{0,1\}, \delta, q_0, F)$ be a deterministic complete finite automaton. Then $\mathscr{B}$ is a* 0*-priority finite automaton iff $\mathscr{B}$ satisfies the following two conditions,*

1. *for any $\gamma \in \Gamma$, there are no $((\gamma,0),(\gamma,0))$-patterns in $\mathscr{B}$;*
2. *for any $\gamma_1,\gamma_2 \in \Gamma$ such that $\gamma_1 \neq \gamma_2$, if there is a $((\gamma_1,0),(\gamma_2,0))$-pattern in $\mathscr{B}$, then there do not exist $((\gamma_2,0),(\gamma_1,0))$-patterns in $\mathscr{B}$.*

Corollary 7. *Given a deterministic complete automaton $\mathscr{B}$ over the alphabet $\Gamma \times \{0,1\}$, it is decidable in polynomial time whether $\mathscr{B}$ is a* 0*-priority finite automaton.*

For each nontrivial SCC, strongly-connected-component, C of G_0, let L_C denote the set of labels $(\gamma,0)$ of the arcs belonging to C.

Proposition 8. *If $\mathscr{B}$ is a* 0*-priority finite automaton, then G_0 enjoys the following two properties.*

1. *Suppose that $q_1 \xrightarrow{(\gamma,0)} q_2$ such that q_1 is* 0*-cyclic, then q_2 is $(\gamma,0)$-cyclic.*
2. *For each nontrivial SCC C of G_0 and each $(\gamma,0) \in L_C$, every state in C is $(\gamma,0)$-cyclic.*

From Proposition 8, the following property can be easily deduced.

Corollary 9. *Let $\mathscr{B}$ be a* 0*-priority finite automaton. If a state q is reachable from some* 0*-cyclic state in $\mathscr{B}$, then q is* 0*-cyclic as well.*

In other words, the above corollary says that 0-acyclic states cannot be reached from 0-cyclic states in a 0-priority finite automaton.

Proposition 10. *Let $L \subseteq (\Gamma \times \{0,1\})^*$ be a regular language. Then L is a* 0*-priority regular language iff the unique minimal deterministic complete finite automaton $\mathscr{B}$ accepting L is a* 0*-priority finite automaton.*

Definition 11 (Class automata with priority class condition, PCA)**.** *A class automaton $(\mathscr{A},\mathscr{B})$ is said to have priority class condition, if the alphabet Γ can be partitioned into k ($k \geq 1$) disjoint subsets $\Gamma_1,\ldots,\Gamma_k$ such that $\mathscr{L}(\mathscr{B})$ is a union of languages $L_1,\ldots,L_k$ satisfying that $L_i \subseteq (\Gamma_i \times \{0,1\})^*$ is a 0-priority regular language for each $i: 1 \leq i \leq k$.*

Intuitively, a class automaton $\mathscr{D} = (\mathscr{A},\mathscr{B})$ with priority class condition is a class automaton such that

> over a data word (w,π), $\mathscr{A}$ nondeterministically chooses an index $i: 1 \leq i \leq k$, then produces a word $w' \in \Gamma_i^*$, and verifies that each class string $w' \otimes X$ belongs to the 0-priority regular language L_i.

Remark 12. *In the definition of PCAs, $\mathscr{L}(\mathscr{B})$ is defined as a disjoint union of 0-priority regular languages, instead of a single 0-priority regular language. PCAs defined in this way can be shown to be closed under union (c.f. Proposition 15), while preserving the decidability (Theorem 18).*

Example 13. *Let $\mathscr{C}$ be the class automaton $(\mathscr{A},\mathscr{B})$ such that $\mathscr{A}$ is the identity transducer and $\mathscr{B}$ is the automaton over the alphabet $\{a,b\} \times \{0,1\}$ in Figure 1(a). Then $\mathscr{C}$ accepts the data words satisfying the property "between any two occurrences of the letter a with the same data value, there is a letter b with a different data value". If $\{a,b\}$ is ordered as ab, then there are no $((a,0),(a,0))$-patterns, nor $((b,0),(a,0))$-patterns, nor $((b,0),(b,0))$-patterns, in $\mathscr{B}$. Thus $\mathscr{B}$ is a 0-priority finite automaton under the ordering ab, so $\mathscr{C}$ is a PCA.*

Remark 14. *Data automata can be seen as PCAs by adding self-loops $q \xrightarrow{(\gamma,0)} q$. Moreover, the extended data automata introduced in [1] can also be seen as a special case of PCA. In extended data automata, the class condition is a finite automaton $\mathscr{B}$ over the alphabet $\Gamma \cup \{0\}$, where the letters in Γ are omitted in zero-transitions. Without loss of generality, $\mathscr{B}$ can be assumed to be deterministic and complete, then a deterministic complete finite automaton $\mathscr{B}'$ over the alphabet $\Gamma \times \{0,1\}$ can be defined as follows: $q \xrightarrow{(\gamma,1)} q'$ in $\mathscr{B}'$ iff $q \xrightarrow{\gamma} q'$ in $\mathscr{B}$, and $q \xrightarrow{(\gamma,0)} q'$ in $\mathscr{B}'$ iff $q \xrightarrow{0} q'$ in $\mathscr{B}$. In the subgraph G_0 of $\mathscr{B}'$, different letters $(\gamma,0)$ are non-distinguishable, so G_0 has the same structure as $G_{(\gamma,0)}$ for any $\gamma \in \Gamma$. Therefore, $\mathscr{B}'$ is a 0-priority finite automaton under any ordering of letters in Γ, and extended data automata can also be seen as PCAs.*

Proposition 15. *The class of data languages accepted by PCAs are closed under letter projection and union, but not under intersection nor complementation.*

The fact that PCAs are not closed under intersection is proved by contradiction: If PCAs are closed under intersection, then PCAs are able to simulate two-counter machines, thus become undecidable, contradicting to Corollary 19 in the next section.

Since data automata are closed under both union and intersection, it can be deduced that PCAs are strictly more expressive than data automata.

Corollary 16. *Class automata with priority class condition are strictly more expressive than data automata.*

Remark 17. *From Corollary 16, we know that there is a data language recognized by PCAs, but not by data automata. It would be nice if we could prove for instance that the data language in Example 13, namely, "Between any two occurrences of the letter a of the same data value, there is an occurrence of the letter b with a different data value", cannot be recognized by data automata. This is stated as an open problem in this paper.*

4 Correspondence between PCA and PMA

The aim of this section is to show that a correspondence between PCAs and PMAs can be established so that the decidability of the nonemptiness of PCAs follows from that of PMAs.

Let $prj : \Sigma \to \Sigma' \cup \{\varepsilon\}$, then the *projection* of a data word (w,π) under prj, denoted by $prj((w,\pi))$, is $prj(w_1)\dots prj(w_{|w|})$, and the projection of a data language L, denoted by $prj(L)$, is $\{prj((w,\pi)) \mid (w,\pi) \in L\}$. Note that the projection of a data language is a language, not a data language.

Theorem 18. *The following two language classes are equivalent:*

- *projections of data languages accepted by PCAs,*
- *languages accepted by PMAs.*

Corollary 19. *The nonemptiness of PCAs is decidble.*

We prove Theorem 18 by showing the following two lemmas.

Lemma 20. *For a PCA $\mathscr{D}$, a PMA $\mathscr{C}$ can be constructed such that $\mathscr{L}(\mathscr{C}) = str(\mathscr{L}(\mathscr{D}))$.*

From Lemma 20, it follows that the first language class in Theorem 18 is included in the second one, since the class of languages accepted by PMAs is closed under mappings $prj : \Sigma_1 \to \Sigma_2 \cup \{\varepsilon\}$. The next lemma says that the second language class in Theorem 18 is included in the first one.

Lemma 21. *For a given PMA $\mathscr{C}$, a PCA $\mathscr{D}$ can be constructed such that $\mathscr{L}(\mathscr{C})$ is a projection of $\mathscr{L}(\mathscr{D})$.*

The rest of this section is devoted to the proof of the Lemma 20. The proof of Lemma 21 is omitted and will appear in the full version of this paper.

The idea of the proof is to consider the abstract runs of class automata, simulate them by multicounter automata, and illustrate that the simulation can be fulfilled by a priority multicounter automaton if the priority class condition is assumed. The proof is inspired by the proof of Theorem 2 in [1].

4.1 From class automata to multicounter automata

Let $\mathscr{D} = (\mathscr{A}, \mathscr{B})$ be a class automaton, where $\mathscr{A} = (Q_g, \Sigma, \Gamma, \delta_g, q_0^g, F_g)$ and $\mathscr{B} = (Q_c, \Gamma \times \{0,1\}, \delta_c, q_0^c, F_c)$. Without loss of generality, we assume that $\mathscr{B}$ is deterministic and complete.

Given a data word (w,π), let $\mathscr{S}(w,\pi)$ be the set of data values occurring in (w,π), namely, $\mathscr{S}(w,\pi) = \{\pi_i \mid 1 \le i \le |w|\}$, and $(w,\pi)_{\le i}$ be the restriction of (w,π) to the set of positions $\{1,\dots,i\}$ for each $i \le |w|$.

Intuitively, a run of $\mathscr{D}$ over a data word (w,π) is a parallel running of the transducer $\mathscr{A}$ and the copies of the automaton $\mathscr{B}$ over (w,π), with one copy for each data value occurring in (w,π). A run of $\mathscr{D}$ over a data word (w,π) can be seen as a sequence $(q_1^g, q_1^c, \gamma_1, R_1)(q_2^g, q_2^c, \gamma_2, R_2)\dots(q_{|w|}^g, q_{|w|}^c, \gamma_{|w|}, R_{|w|})$ such that

- the sequence $(q_1^g, \gamma_1)\dots(q_{|w|}^g, \gamma_{|w|})$ corresponds to a run of the transducer $\mathscr{A}$,
- q_i^c records the state of a copy of $\mathscr{B}$ corresponding to a data value that has not been met until the position i, namely, a data value $d \notin \mathscr{S}((w,\pi)_{\le i})$,
- each time a new data value π_i is met, $R_i(\pi(i))$ is set as $\delta_c(q_{i-1}^c, (\gamma_i, 1))$, since $\pi(i)$ has not been met before and q_{i-1}^c records the current state of $\mathscr{B}$ for the new data values.

Formally, A run of $\mathscr{D}$ over a data word (w,π) is a sequence $(q_1^g, q_1^c, \gamma_1, R_1)\dots(q_{|w|}^g, q_{|w|}^c, \gamma_{|w|}, R_{|w|})$ satisfying the following conditions,

- for each $i : 1 \le i \le |w|$, $(q_{i-1}^g, w_i, \gamma_i, q_i^g) \in \delta_g$, $\delta_c(q_{i-1}^c, (\gamma_i, 0)) = q_i^c$ (where q_0^g, q_0^c are the initial states of respectively $\mathscr{A}, \mathscr{B}$),

- for each i, R_i is a function from $\mathscr{S}((w,\pi)_{\leq i})$ to Q_c, satisfying the following conditions,
  - $R_1(\pi_1) = \delta_c(q_0^c, (\gamma_1, 1))$,
  - for each $i : 1 < i \leq |w|$,
    $R_i(\pi_i) = \delta_c(R_{i-1}(\pi_i), (\gamma_i, 1))$ if $\pi_i \in \mathscr{S}((w,\pi)_{\leq i-1})$, otherwise $R_i(\pi_i) = \delta_c(q_{i-1}^c, (\gamma_i, 1))$.
    For each $d \in \mathscr{S}((w,\pi)_{\leq i-1})$ such that $d \neq \pi_i$, $R_i(d) = \delta_c(R_{i-1}(d), (\gamma_i, 0))$.

A run $(q_1^g, q_1^c, \gamma_1, R_1) \ldots (q_{|w|}^g, q_{|w|}^c, \gamma_{|w|}, R_{|w|})$ is *successful* if $q_{|w|}^g \in F_g$ and $R_{|w|}(d) \in F_c$ for each $d \in \mathscr{S}(w,\pi)$.

The functions $R_1, \ldots, R_{|w|}$ in a run of $\mathscr{D}$ on the data word (w,π) can be abstracted into a sequence of functions $C_1, \ldots, C_{|w|}$ such that each C_i is a function $Q_c \to \mathbb{N}$ satisfying that for each $q \in Q_c$, $C_i(q)$ is the number of data values $d \in \mathscr{S}((w,\pi)_{\leq i})$ such that $R_i(d) = q$.

Intuitively, each C_i is a tuple of counter values, with one counter for each state in Q_c. The sequence $C_1, \ldots, C_n$ can be seen in a more abstract way, without directly referring to the data values in $\mathscr{S}((w,\pi))$, as follows:

> For each $1 < i \leq |w|$, C_i is obtained from C_{i-1} by nondeterministically choosing one of the following two possibilities:
>
> - either (corresponding to the situation $\pi_i \in \mathscr{S}((w,\pi)_{\leq i-1})$)
>   - select some counter q' with non-zero value (i.e. $C_{i-1}(q') > 0$), decrement the counter q',
>   - then for each counter q'', the value of q'' is assigned as the sum of those of counters p such that $\delta_c(p, (\gamma_i, 0)) = q''$,
>   - finally increment the counter $\delta_c(q', (\gamma_i, 1))$.
> - or (corresponding to the situation $\pi_i \notin \mathscr{S}((w,\pi)_{\leq i-1})$)
>   - for each counter q'', the value of q'' is assigned the sum of those of counters p such that $\delta_c(p, (\gamma_i, 0)) = q''$,
>   - increment the counter $\delta_c(q_{i-1}^c, (\gamma_i, 1))$.

The sequence $(q_1^g, q_1^c, \gamma_1, C_1)(q_2^g, q_2^c, \gamma_2, C_2) \ldots (q_{|w|}^g, q_{|w|}^c, \gamma_{|w|}, C_{|w|})$ is said to be an *abstract run* of $\mathscr{D}$ over the data word (w,π).

With such an abstract view of runs, $\mathscr{D}$ can be transformed into a multicounter automaton (with zero tests) $\mathscr{C} = (Q_a, \Sigma, k, \delta_a, q_0^a, F_a = \{q_{acc}\})$ as follows,

- Q_a includes $Q_g \times Q_c$ and some auxiliary states, e.g. for controlling the updates of the counter values.
- $\mathscr{C}$ consists of $k = |Q_c|$ counters, one counter for each state in Q_c.
- $q_0^a = (q_0^g, q_0^c)$.
- Each $\gamma \in \Gamma$ induces a series of transition rules in δ_a as follows:
  If
  > the current state of $\mathscr{C}$ is (p^g, p^c), the read head is in a position labeled by $\sigma \in \Sigma$, and there are $q^g \in Q_g, q^c \in Q_c$ such that $(p^g, \sigma, \gamma, q^g) \in \delta_g$ and $\delta_c(p^c, (\gamma, 0)) = q^c$,

  then
  > the state of $\mathscr{C}$ is changed into (q^g, q^c), the counter values are updated in such a way to obtain C_i from C_{i-1} as above, and the read head is moved to the next position.
- Nondeterministically, $\mathscr{C}$ changes the state into a special state q_s and repeats the following action:

$\mathscr{C}$ arbitrarily chooses a non-zero counter $q \in F_c$, decrements q. Then it tests whether all the counters have zero value. If so, $\mathscr{C}$ changes the state into q_{acc} and accepts.

We now specify in detail how to update the counter values in $\mathscr{C}$, essentially, how to perform the following updates:

For each counter q'' in $\mathscr{C}$, the value of q'' is assigned the sum of those of the counters p such that $\delta_c(p,(\gamma,0)) = q''$.

Recall that each connected component of $G_{(\gamma,0)}$ of $\mathscr{B}$ consists of a unique cycle C and several paths towards C. Let $C = q_1 \dots q_r$, then for each $1 < i \leq r$, the value of the counter q_{i+1} is assigned as the sum of the value of the counter q_i and the values of the counters of its predecessors not in C, where $q_{r+1} = q_1$ by convention. Then the counter values can be updated as follows,

1. the counters corresponding to the states in C are first renamed[1]: For each $i : 1 \leq i \leq r$, q_i is renamed as q_{i+1}, where $q_{r+1} = q_1$ by convention. The renaming is remembered by the finite-state control of $\mathscr{C}$. With this renaming, the counter q_{i+1} takes the value of the counter q_i for each $i : 1 \leq i \leq r$.
2. then the values of the counters on the paths towards C are updated in a backward way: For instance, let $p_1 \xrightarrow{(\gamma,0)} p_2 \xrightarrow{(\gamma,0)} p_3$ such that $p_3 \in C, p_1, p_2 \notin C$, then the value of p_2 is first added into p_3, by decrementing p_2 and incrementing p_3 until the value of p_2 becomes zero; afterwards, the value of p_1 is added into p_2, and so on.

The above updates of counter values of $\mathscr{C}$ need (unrestricted) zero tests. In the following we will show that if $\mathscr{D}$ is a PCA, then these updates can be done with the restricted zero tests of PMAs, namely, testing zero for a prefix of counters as a whole, instead of a single counter.

4.2 From PCA to PMA

We first assume that $(\mathscr{A},\mathscr{B})$ is a PCA such that $\mathscr{L}(\mathscr{B})$ is a 0-priority regular language, and $\mathscr{B}$ is a 0-priority finite automaton. Later we will consider the more general case that $\mathscr{L}(\mathscr{B})$ is a disjoint union of 0-priority regular languages.

We first introduce some notations and prove a property of abstract runs of PCA.

Suppose that Γ is ordered as $\gamma_1 \dots \gamma_l$ under which $\mathscr{B}$ is a 0-priority finite automaton.

Let $D_{scc}(G_0)$ be the strongly-connected-component directed graph of G_0 of $\mathscr{B}$, then $D_{scc}(G_0)$ is an acyclic directed graph. Let $\#_{scc}(G_0)$ denote the maximal length (number of arcs) of paths in $D_{scc}(G_0)$.

Similar to Lemma 1 in [1], we can obtain the following lemma.

Lemma 22. *Let $\mathscr{D} = (\mathscr{A},\mathscr{B})$ be a PCA such that $\mathscr{B}$ is a 0-priority finite automaton. Then any abstract run of $\mathscr{D}$ over a data word (w,π), say $(q^g_1, q^c_1, \gamma_1, C_1)\dots(q^g_{|w|}, q^c_{|w|}, \gamma_{|w|}, C_{|w|})$, enjoys the following property:*

For each $i : 1 \leq i \leq |w|$, the sum of $C_i(q')$'s such that q' is 0-acyclic is bounded by $\#_{scc}(G_0)$.

By utilizing Lemma 22, we then demonstrate how the updates of the counter values of the multi-counter automaton $\mathscr{C}$ obtained from $\mathscr{D}$ in Section 4.1 can be done with the restricted zero tests in PMAs.

We introduce some additional notations.

For each $i : 1 \leq i \leq l$, let $Acyc_i$ denote the set of 0-*cyclic* states $q \in Q_c$ such that q is $(\gamma_i, 0)$-*acyclic*.

In addition, let $Acyc_{l+1}$ denote the set of 0-cyclic states $q \notin \bigcup_{i:1\leq i\leq l} Acyc_i$ by convention.

[1] The idea of renaming is from [1]

Proposition 23. *Let $\mathscr{D} = (\mathscr{A}, \mathscr{B})$ be a PCA such that $\mathscr{B}$ is a* 0*-priority finite automaton under the ordering $\gamma_1 \dots \gamma_l$. Then $Acyc_1, \dots, Acyc_{l+1}$ satisfy the following two properties:*

1. *$Acyc_i \subseteq Acyc_{i+1}$ for each $i < l$.*
2. *For each $i : 1 \leq i \leq l$, if $q \in Acyc_i$ and $q \xrightarrow{(\gamma_i,0)} q'$, then $q' \notin Acyc_i$ and $q' \in Acyc_j$ for some $j > i$. In particular, if $q \in Acycl_l$ and $q \xrightarrow{(\gamma_l,0)} q'$, then $q' \notin Acyc_l$ and $q' \in Acyc_{l+1}$.*

We are ready to show that if $\mathscr{D}$ is a PCA, then $\mathscr{C}$ can be turned into a PMA $\mathscr{C}_p = (Q_p, \Sigma, k, \delta_p, q_0^p, F_p)$.

From Lemma 22, if $\mathscr{D}$ is a PCA, then in the multicounter automaton $\mathscr{C}$, the sum of the values of the counters corresponding to the 0-acyclic states of $\mathscr{B}$ are always bounded. Thus in $\mathscr{C}_p$, the counters corresponding to these 0-acyclic states become *virtual*, in the sense that the values of these counters are stored in the finite state control of $\mathscr{C}_p$, and there are no *real* counters in $\mathscr{C}_p$ corresponding to the 0-acyclic states of $\mathscr{B}$.

The state set of $\mathscr{C}_p$ consists of the states $(p^g, p^c, \mathscr{I}_{Acyc})$ and some auxiliary states for updating the counter values, where $\mathscr{I}_{Acyc}$ is the information about the virtual counters corresponding to the 0-acyclic states of $\mathscr{B}$. The counters of $\mathscr{C}_p$ correspond to the 0-cyclic states of $\mathscr{B}$, with one counter for each 0-cyclic state.

The counters (corresponding to the 0-cyclic states of $\mathscr{B}$) of $\mathscr{C}_p$ are ordered according to the following order of 0-cyclic states of $\mathscr{B}$,

$$Acyc_1 (Acyc_2 \setminus Acyc_1) \dots (Acyc_l \setminus Acyc_{l-1}) Acyc_{l+1},$$

where an arbitrary ordering is given to the states within $Acyc_1$, $Acyc_{l+1}$, and each of $Acyc_{i+1} \setminus Acyc_i$ for $i : 1 \leq i < l$.

Each $\gamma \in \Gamma$ induces a series of transition rules in δ_p specified in the following.

If the current state of $\mathscr{C}_p$ is $(p^g, p^c, \mathscr{I}_{Acyc})$, the read head is in some position labeled by σ, and there are $q^g \in Q_g, q^c \in Q_c$ such that $(p^g, \sigma, \gamma, q^g) \in \delta_g$ and $\delta_c(p^c, (\gamma, 0)) = q^c$, then the state of $\mathscr{C}_p$ is changed into $(q^g, q^c, \mathscr{I}'_{Acyc})$. Now we illustrate how the values of the real counters are updated and how the values of the virtual counters, i.e. $\mathscr{I}_{Acyc}$ in the finite state control of $\mathscr{C}_p$, is updated into $\mathscr{I}'_{Acyc}$, by the following three steps.

1. Either

   the state $p_1^c = \delta_c(p^c, (\gamma, 1))$ (a new data value is met) is stored in the finite state control of $\mathscr{C}_p$,

   or

   some (0-acyclic or 0-cyclic) state $q' \in Q_c$ (an old value is met) is selected, the (virtual or real) counter corresponding to q' is decremented, and the state $p_1^c = \delta_c(q', (\gamma, 1))$ (the virtual or real counter corresponding to it should be incremented) is stored in the finite-state control of $\mathscr{C}_p$.

2. The values of the (virtual or real) counters are updated as follows.

   Let $\gamma = \gamma_i$ for some $i : 1 \leq i \leq l$.

   The counters corresponding to the states in $Acyc_j \setminus Acyc_{j-1}$ for $j > i$, which are $(\gamma_i, 0)$-cyclic in $\mathscr{B}$, are first updated by renaming, with the renaming stored in the finite state control of $\mathscr{C}_p$. Then for each counter $q \in Acyc_1$, the value of the counter q is added to its $(\gamma_i, 0)$-successor q', which is in $Acyc_j \setminus Acyc_i$ for some $j > i$ according to the fact that $q \in Acyc_1 \subseteq Acyc_i$, $q \xrightarrow{(\gamma_i,0)} q'$ and Proposition 23. Namely, the value of the counter q is decremented and the value of q' is incremented until the value of the counter q becomes zero. Afterwards, for each counter $q \in Acyc_2 \setminus Acyc_1$, the value of the counter q is added to its $(\gamma_i, 0)$-successor (which is also in $Acyc_j \setminus Acyc_i$ for some $j > i$), and so on, until all the counters corresponding to the states in $Acyc_i \setminus Acyc_{i-1}$ are updated.

Note that during these updates of counter values, the zero-tests can be restricted to the zero-tests for a prefix of counters. The reason is that when updating the counter corresponding to a state $q \in Acyc_{j+1} \setminus Acyc_j$ for some $j < i$, the values of the counters corresponding to the states in $Acyc_1, \ldots, Acyc_j \setminus Acyc_{j-1}$ are already zero. Therefore, testing zero for the counter q is equal to testing zero for the counters before q (including q) in the ordering.

Then, $\mathscr{I}_{Acyc}$, i.e. the information about the values of the virtual counters, is updated into $\mathscr{I}'_{Acyc}$ by following G_0, the zero-transitions of $\mathscr{B}$, and some real counters (corresponding to the 0-cyclic states) should also be incremented if they correspond to the $(\gamma_i, 0)$-successors of some 0-acyclic states in $\mathscr{B}$.

3. If p^c_1 is 0-acyclic, then $\mathscr{I}'_{Acyc}$ is further updated by incrementing the value of the virtual counter p^c_1, otherwise, the value of the real counter corresponding to the (0-cyclic) state p^c_1 is incremented.

The definition of the F_p of $\mathscr{C}_p$ is similar to F_a of $\mathscr{C}$ in Section 4.1.

Finally the read head is moved to the next position.

This finishes the description of $\mathscr{C}_p$.

At last, we consider the general case that $\mathscr{L}(\mathscr{B})$ is a disjoint union of 0-priority regular languages, i.e. Γ is a disjoint union of $\Gamma_1, \ldots, \Gamma_k$ ($k \geq 1$) such that

- for each $u \in \Sigma^*$, $\mathscr{A}$ outputs a word in $\Gamma_1^* \cup \ldots \Gamma_k^*$,
- $\mathscr{L}(\mathscr{B})$ is a union of languages $L_1, \ldots, L_k$ satisfying that $L_i \subseteq (\Gamma_i \times \{0,1\})^*$ is a 0-priority regular language for each i.

For each i, let Γ_i be ordered as $\gamma_{i,1} \ldots \gamma_{i,l_i}$ under which L_i is a 0-priority regular language.

For each i, suppose $\mathscr{B}_i$ is a 0-priority finite automaton accepting L_i and $Acyc_{i,j} (1 \leq j \leq l_i + 1)$ is the set of 0-cyclic and $(\gamma_{i,j}, 0)$-acyclic states in $\mathscr{B}_i$.

Then from the PCA $\mathscr{D}$, a PMA $\mathscr{C}$ can be constructed such that the counters of $\mathscr{C}$ correspond to the set of 0-cyclic states in all these $\mathscr{B}_i$'s, and these counters are ordered as follows,

$$Acyc_{1,1}(Acyc_{1,2} \setminus Acyc_{1,1}) \ldots (Acyc_{1,l_1} \setminus Acyc_{1,l_1-1}) Acyc_{1,l_1+1} \ldots$$
$$Acyc_{k,1}(Acyc_{k,2} \setminus Acyc_{k,1}) \ldots (Acyc_{k,l_k} \setminus Acyc_{k,l_k-1}) Acyc_{k,l_k+1}.$$

In the PCA $\mathscr{D}$, after the transducer $\mathscr{A}$ nondeterministically chooses an index i and outputs a string in Γ_i^*, only the 0-priority finite automaton $\mathscr{B}_i$ is used and the other automata $\mathscr{B}_j$ for $j \neq i$ remain idle, thus the values of the counters before $Acyc_{i,1}$ in the above ordering are always zero, and the updates of the counter values corresponding to the states $Acyc_{i,1}, \ldots, Acyc_{i,l_i} \setminus Acyc_{l_i-1} Acyc_{l_i+1}$ can still be fulfilled using the restricted zero tests of PMAs.

5 Application to the analysis of array-accessing programs

In this section, we demonstrate how to apply class automata with priority class condition to the algorithmic analysis of array-processing programs considered in [1]. The notations of this section follow those in [1].

An array A is a list $(A[1].s, A[1].d) \ldots (A[n].s, A[n].d)$ such that $A[i].s \in \Sigma$ and $A[i].d \in \mathbb{D}$ for each $i : 1 \leq i \leq n$.

The syntax of array-accessing programs over an array A are defined by the following rules[2]:

[2]The nondeterministic-choice rule $if\ *\ then\ P\ else\ P$ is not included here for simplicity

$$P ::= skip \mid \{P\} \mid b := B \mid p := IE \mid v := DE \mid$$
$$\text{if } B \text{ then } P \text{ else } P \mid \text{for } i := 1 \ to\ length(A) \text{ do } P \mid P; P$$

where

- $i, j, i_1, j_1, \ldots$ are loop variables, $p, p_1, \ldots$ are index variables, $v, v_1, \ldots$ are data variables, and $b, b_1, \ldots$ are Boolean variables,
- $s, s_1, \cdots \in \Sigma$ and $c, c_1, \cdots \in \mathbb{D}$ are constants,
- $IE ::= p \mid i$ are index expressions, $SE ::= s \mid A[IE].s$ are Σ-expressions, $DE ::= v \mid c \mid A[IE].d$ are data expressions, and B are Boolean expressions defined by the following rules,

$$B ::= true \mid false \mid b \mid B\ and\ B \mid not\ B \mid IE = IE \mid IE < IE \mid DE = DE \mid DE < DE \mid SE = SE.$$

A *state* of the array-accessing program P is an assignment of values to the variables in P.
A *Boolean state* of the program P is an assignment of values to the Boolean variables in P.
The *initial state* of the program P is a state such that

- all the Boolean variables have value $false$;
- all the loop and index variables have value 1;
- all the data variables have the value the same as the first element of A.

A *loop-free* program is a program containing no loops, namely a program formed without using the rules "for $i := 1$ to $length(A)$ do P".

The *Boolean state reachability* problem is defined as follows: Given a program P and a Boolean state m of P, whether there is an array A such that m is reached from the initial state after the execution of P over A.

Restricted ND_2 programs are programs of the following form,

```
for i:=1 to length(A) do
{
   P1;
   for j:=1 to length(A) do
   {
      if A[i].d=A[j].d then
                P2
      else
                P3
   };
   P4
}
```

such that

- $P1, P2, P3, P4$ are loop-free,
- $P1, P2, P3, P4$ do not use index or data variables,
- $P1, P2, P3, P4$ do not refer to the order on indices or data.

Theorem 24 ([1]). *The Boolean state reachability problem is decidable for Restricted ND_2 programs satisfying the following additional condition:*

$P3$ does not refer to $A[j]$, i.e. it does not contain the occurrences of $A[j].s$ or $A[j].d$.

The idea of the proof of Theorem 24 is to reduce the Boolean state reachability problem to the nonemptiness of extended data automata $\mathscr{D} = (\mathscr{A}, \mathscr{B})$ (c.f. Remark 14) such that

- $\mathscr{A}$ guesses an accepting run of the outer-loop of P over an array A,
- $\mathscr{B}$ corresponds to the inner loop and verifies the consistency of the guessed run.

Roughly speaking, $\mathscr{B}$ can be constructed from $P2$ and $P3$ such that

- $P2$ corresponds to the one-transitions in $\mathscr{B}$,
- $P3$ corresponds to the zero-transitions in $\mathscr{B}$.

The restriction that P_3 does not refer to $A[j]$ in Theorem 24 is crucial, because in extended data automata, the labels are omitted in zero-transitions of the class condition $\mathscr{B}$.

On the other hand, as we have shown, PCAs, i.e. class automata with priority class conditions, do not omit the labels in zero-transitions and strictly generalize extended data automata. So naturally, by using PCAs, we should be able to show that the Boolean state reachability problem is decidable for a larger class of programs than those in Theorem 24.

Similar to the construction of extended data automata from Restricted-ND_2 programs satisfying the additional condition in Theorem 24, we have the following result.

Lemma 25. *For a Restricted-ND_2 program P and a Boolean state m, a class automaton $\mathscr{D} = (\mathscr{A}, \mathscr{B})$ can be constructed such that m is reached from the initial state after the run of P over an array A iff the array (data word) A is accepted by $\mathscr{D}$.*

In principle, the Boolean reachability problem is decidable for Restricted-ND_2 programs P satisfying the additional condition that the class automaton $\mathscr{D} = (\mathscr{A}, \mathscr{B})$ constructed from P in Lemma 25 is a class automaton with priority class condition. However, this condition is in some sense a semantical condition, since the construction of the automaton $\mathscr{D}$ from P has an exponential blow-up. In the following, we demonstrate how to define a simple syntactic condition for $P3$ which guarantees that $\mathscr{D}$ constructed from P is a PCA.

The 0-*priority restricted-ND_2* program is a Restricted-ND_2 program satisfying the following condition:

> Either $P3$ does not refer to $A[j]$, i.e. it does not contain the occurrences of $A[j].s$ or $A[j].d$, or there are a set of constants $s1, \ldots, sr \in \Sigma$ such that $P3$ is a program of the following form,
>
> ```
>  if BB then
>    if A[j].s =s1 then
>       PA1
>    else if A[j].s=s2 then
>       PA2
>    ...
>    else if A[j].s=sr then
>       PAr
>    else skip
> else skip
> ```
>
> such that
>
> - BB is a conjunction of literals, i.e. b or *not* b for Boolean variables b,
> - $PA1, PA2, \ldots, PAr$ are compositions of the assignments $b := true$ or $b := false$ for Boolean variables b,

- Each *PAi* for $1 \leq i \leq r$ is *nontrivial* in the sense that there is a Boolean variable b such that either b is a conjunct of *BB* and the assignment $b := false$ is in *PAi*, or *not* b is a conjunct of *BB* and the assignment $b := true$ occurs in *PAi*.

Remark 26. *The* 0*-priority restricted-ND*$_2$ *programs subsume the Restricted-ND*$_2$ *programs satisfying that P*3 *does not refer to* $A[j]$*. A slightly more general syntactic condition than the above can be defined, which we choose not to present here, since the condition is rather tedious, and we believe that the simple condition presented above already sheds some light on the usefulness of PCAs.*

Example 27. *The following program to describe the property "for any two occurrences of the letter a with the same data value in A, there is an occurrence of the letter b between them with a different data value" (c.f. Example 13) is an example of* 0*-priority restricted-ND*$_2$ *programs. Intuitively,*

- *the Boolean state* $b1 = true, b2 = false, b3 = false$ *corresponds to the state* q_0 *in Figure 1(a), the Boolean state* $b1 = false, b2 = true, b3 = false$ *corresponds to the state* q_1*, and the Boolean state* $b1 = false, b2 = false, b3 = true$ *correspond to the Boolean state* q_2*;*
- *the outer loop selects a position i and the inner loop verifies that the class string corresponding to the data value* $A[i].d$ *satisfies the class condition.*

```
for i:=1 to length(A) do
{
  if not b3 then    %the sink state q2 is not reached yet
    b1: = true; b2:=false
  else
    skip
  for j:=1 to length(A) do
  { if A[i].d = A[j].d then
     { if A[j].s=a then
           if b1 and not b2 and not b3 then
              b1:=false; b2:=true
           else if not b1 and b2 and not b3 then
              b2:=false; b3:=true
           else skip
        else skip
     }
     else
     { if not b1 and b2 and not b3 then
           if A[j].s = b then
              b2:=false; b1:= true
           else skip
       else skip
     }
  }
}
```

An array A satisfies the property iff the Boolean state $b1 = true, b2 = false, b3 = false$ *or the state* $b1 = false, b2 = true, b3 = false$ *is reached from the initial state after the run of the above program over the array A.*

Theorem 28. *The Boolean state reachability problem is decidable for* 0*-priority restricted-*ND_2 *programs.*

Acknowledgement. The author thanks Anca Muscholl for introducing him to this field. The author also thanks Luc Segoufin, Stéphane Demri, and Mikołaj Bojańczyk for the discussions and suggestions. Last but not the least, the author thanks anonymous referees for the valuable suggestions and comments.

References

[1] Rajeev Alur, Pavol Cerný & Scott Weinstein (2009): *Algorithmic Analysis of Array-Accessing Programs*. In: *CSL'09*, *LNCS 5771*, pp. 86–101, doi:10.1007/978-3-642-04027-6_9. Also available as a technical report, `http://repository.upenn.edu/cis_reports/894/`.

[2] Henrik Björklund & Mikłoj Bojanczyk (2007): *Bounded depth data trees*. In: *In ICALP' 07*, pp. 862–874, doi:10.1007/978-3-540-73420-8_74.

[3] Mikołaj Bojańczyk & Sławomir Lasota (2010): *An extension of data automata that captures XPath*. In: *LICS '10*, pp. 243–252, doi:10.1109/LICS.2010.33.

[4] Mikołaj Bojanczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin & Claire David (2006): *Two-Variable Logic on Words with Data*. In: *LICS '06*, pp. 7–16, doi:10.1109/LICS.2006.51.

[5] K. Reinhardt (2005): *Counting as Method, Model and Task in Theoretical Computer Science*. Habilitation thesis, Universität Tübingen.

[6] Luc Segoufin (2006): *Automata and Logics for Words and Trees over an Infinite Alphabet*. In: *CSL, LNCS 4207*, pp. 41–57, doi:10.1007/11874683_3.

A Appendix

A.1 Proofs in Section 3

Proposition 6. *Let $\mathscr{B} = (Q, \Gamma \times \{0,1\}, \delta, q_0, F)$ be a deterministic complete finite automaton. Then $\mathscr{B}$ is a 0-priority finite automaton iff $\mathscr{B}$ satisfies the following two conditions,*

1. *for any $\gamma \in \Gamma$, there are no $((\gamma,0),(\gamma,0))$-patterns in $\mathscr{B}$;*
2. *for any $\gamma_1, \gamma_2 \in \Gamma$ such that $\gamma_1 \neq \gamma_2$, if there is a $((\gamma_1,0),(\gamma_2,0))$-pattern in $\mathscr{B}$, then there do not exist $((\gamma_2,0),(\gamma_1,0))$-patterns in $\mathscr{B}$.*

Proof. The "only if" direction is obvious.

Now consider the "if" direction.

Suppose that $\mathscr{B}$ satisfies the two conditions.

Define the alphabet dependency graph (Γ, E) as follows: $(\gamma_1, \gamma_2) \in E$ iff there is a $((\gamma_1,0),(\gamma_2,0))$-pattern in $\mathscr{B}$.

If (Γ, E) is an acyclic directed graph, let $\gamma_1 \dots \gamma_k$ be a topological ordering of (Γ, E), then $\mathscr{B}$ is a 0-priority finite automaton under this ordering.

In the following, we show that (Γ, E) is indeed an acyclic directed graph.

To the contrary, suppose that (Γ, E) is not acyclic, i.e. it contains at least one cycle, say C.

Because there are no self-loops (the 1st condition) and no cycles of length two (the 2nd condition), the length of C (number of vertices) must be at least three.

We demonstrate the contradiction by considering the special case that C is of length exactly three, i.e. $C = \gamma_{i_1}\gamma_{i_2}\gamma_{i_3}$. The argument for the more general case is similar.

Then there is a $((\gamma_{i_j},0),(\gamma_{i_{j+1}},0))$-pattern in $\mathscr{B}$ for each $1 \leq j \leq 3$ (where $\gamma_{i_4} = \gamma_{i_1}$).

Let (q_1,q_2,q_3,q_4) be a $((\gamma_{i_1},0),(\gamma_{i_2},0))$-pattern, namely, $q_1 \xrightarrow{(\gamma_{i_1},0)} q_2 \xrightarrow[0]{*} q_3 \xrightarrow{(\gamma_{i_2},0)} q_4$, q_1 is 0-cyclic and q_3 is $(\gamma_{i_2},0)$-acyclic.

If q_2 is $(\gamma_{i_3},0)$-cyclic, then there will be a state q such that $q \xrightarrow{(\gamma_{i_3},0)} q_2 \xrightarrow[0]{*} q_3 \xrightarrow{(\gamma_{i_2},0)} q_4$, q is 0-cyclic and q_3 is $(\gamma_{i_2},0)$-acyclic, so (q,q_2,q_3,q_4) is a $((\gamma_{i_3},0),(\gamma_{i_2},0))$-pattern in $\mathscr{B}$. On the other hand, since $(\gamma_{i_2},\gamma_{i_3}) \in E$, there is also a $((\gamma_{i_2},0),(\gamma_{i_3},0))$-pattern in $\mathscr{B}$, contradicting to the 2nd condition.

If q_2 is $(\gamma_{i_3},0)$-acyclic, then there will be a state q such that $q_1 \xrightarrow{(\gamma_{i_1},0)} q_2 \xrightarrow[0]{*} q_2 \xrightarrow{(\gamma_{i_3},0)} q$, q_1 is 0-cyclic and q_2 is $(\gamma_{i_3},0)$-acyclic, so (q_1,q_2,q_2,q) is a $((\gamma_{i_1},0),(\gamma_{i_3},0))$-pattern in $\mathscr{B}$. On the other hand, since $(\gamma_{i_3},\gamma_{i_1}) \in E$, there is also a $((\gamma_{i_3},0),(\gamma_{i_1},0))$-pattern in $\mathscr{B}$, contradicting to the 2nd condition as well. □

Proposition 8. *If $\mathscr{B}$ is a 0-priority finite automaton, then G_0 enjoys the following two properties.*

1. *Suppose that $q_1 \xrightarrow{(\gamma,0)} q_2$ such that q_1 is 0-cyclic, then q_2 is $(\gamma,0)$-cyclic.*
2. *For each nontrivial SCC C of G_0 and each $(\gamma,0) \in L_C$, every state in C is $(\gamma,0)$-cyclic.*

Proof. 1. To the contrary, suppose that $q_1 \xrightarrow{(\gamma,0)} q_2$ such that q_1 is 0-cyclic, but q_2 is $(\gamma,0)$-acyclic.

Because q_2 is $(\gamma,0)$-acyclic, there is q_3 such that $q_1 \xrightarrow{(\gamma,0)} q_2 \xrightarrow[0]{*} q_2 \xrightarrow{(\gamma,0)} q_3$, q_1 is 0-cyclic and q_2 is $(\gamma,0)$-acyclic, so (q_1,q_2,q_2,q_3) is a $((\gamma,0),(\gamma,0))$-pattern in $\mathscr{B}$, contradicting to the assumption that $\mathscr{B}$ is a 0-priority finite automaton.

2. To the contrary, suppose that C is a nontrivial SCC of G_0, $(\gamma,0) \in L_C$, i.e. $q_1 \xrightarrow{(\gamma,0)} q_2$ for some $q_1, q_2 \in C$, and there is $q_3 \in C$ such that q_3 is $(\gamma,0)$-acyclic.

Then there is q_4 such that $q_3 \xrightarrow{(\gamma,0)} q_4$. Therefore, we have $q_1 \xrightarrow{(\gamma,0)} q_2 \xrightarrow[0]{*} q_3 \xrightarrow{(\gamma,0)} q_4$ such that q_1 is 0-cyclic and q_3 is $(\gamma,0)$-acyclic. So (q_1,q_2,q_3,q_4) is a $((\gamma,0),(\gamma,0))$-pattern in $\mathscr{B}$, which contradicts to the assumption that $\mathscr{B}$ is a 0-priority finite automaton. $\square$

Proposition 10. *Let $L \subseteq (\Gamma \times \{0,1\})^*$ be a regular language. Then L is a 0-priority regular language iff the unique minimal deterministic complete finite automaton $\mathscr{B}$ accepting L is a 0-priority finite automaton.*

Proof. The "if" direction is trivial.

For the "only if" direction, suppose that L is accepted by a 0-priority finite automaton $\mathscr{B}$ under the ordering $\gamma_1 \dots \gamma_k$.

The unique minimal deterministic complete automaton $\mathscr{B}'$ is obtained from $\mathscr{B}$ by merging equivalent states. The states of $\mathscr{B}'$ are subsets of states of $\mathscr{B}$.

To the contrary, suppose that $\mathscr{B}'$ is not a 0-priority finite automaton under the ordering $\gamma_1 \dots \gamma_k$. Then there is a $((\gamma_i,0),(\gamma_j,0))$-pattern with $i \geq j$ in $\mathscr{B}'$. Let (S_1,S_2,S_3,S_4) be such a $((\gamma_i,0),(\gamma_j,0))$-pattern, namely, $S_1 \xrightarrow{(\gamma_i,0)} S_2 \xrightarrow[0]{*} S_3 \xrightarrow{(\gamma_j,0)} S_4$, S_1 is 0-cyclic and S_3 is $(\gamma_j,0)$-acyclic.

Since S_1 is 0-cyclic in $\mathscr{B}'$ and $\mathscr{B}'$ is obtained from $\mathscr{B}$ by merging equivalent states, it follows that each state $q_1 \in S_1$ is 0-cyclic in $\mathscr{B}$.

We claim that there is a state $q_3 \in S_3$ such that q_3 is $(\gamma_j,0)$-acyclic in $\mathscr{B}$.

Otherwise, each $q_3 \in S_3$ is $(\gamma_j,0)$-cyclic in $\mathscr{B}$. Let N be the least common multiple of the length of all these $(\gamma_j,0)$ cycles. Then S_3 can be reached from itself by a $(\gamma_j,0)$-cycle of length N in $\mathscr{B}'$, contradicting to the fact that S_3 is $(\gamma_j,0)$-acyclic in $\mathscr{B}'$.

Let $q_3 \in S_3$ be a $(\gamma_j,0)$-acyclic state in $\mathscr{B}$. Then there are $q_1 \in S_1$, $q_2 \in S_2$, $q_4 \in S_4$ such that $q_1 \xrightarrow{(\gamma_i,0)} q_2 \xrightarrow[0]{*} q_3 \xrightarrow{(\gamma_j,0)} q_4$ in $\mathscr{B}$. Since q_1 is 0-cyclic and q_3 is $(\gamma_j,0)$-acyclic, it follows that (q_1,q_2,q_3,q_4) is a $((\gamma_i,0),(\gamma_j,0))$-pattern in $\mathscr{B}$, contradicting to the assumption that $\mathscr{B}$ is a 0-priority finite automaton under the ordering $\gamma_1 \dots \gamma_k$.

So We conclude that $\mathscr{B}'$ is a 0-priority finite automaton under the ordering $\gamma_1 \dots \gamma_k$. $\square$

Proposition 15. *The class of languages of data words accepted by PCAs are closed under letter projection and union, but not under intersection nor complementation.*

Proof.

1. *Letter projection*.

The closure under letter projection follows from the nondeterminism of the transducer in the definition of PCAs.

2. *Union*.

Suppose $\mathscr{D}_1 = (\mathscr{A}_1, \mathscr{B}_1)$ and $\mathscr{D}_2 = (\mathscr{A}_2, \mathscr{B}_2)$ are two PCAs such that $\mathscr{A}_i = (Q^g_i, \Sigma, \Gamma_i, \delta_{g,i}, q_0^{g,i}, F_{g,i})$, Γ_i is a disjoint union of $\Gamma_{i,1}, \dots, \Gamma_{i,k_i}$, and $\mathscr{L}(\mathscr{B}_i)$ is a disjoint union of 0-priority regular languages $L_{i,1} \subseteq \Gamma^*_{i,1}, \dots, L_{i,k_i} \subseteq \Gamma^*_{i,k_i}$, where $i = 1,2$.

Without loss of generality, we can assume that $\Gamma_1 \cap \Gamma_2 = \emptyset$.

Then the automaton $\mathscr{D} = (\mathscr{A}, \mathscr{B})$ defined in the following accepts the union of $\mathscr{L}(\mathscr{A})$ and $\mathscr{L}(\mathscr{B})$:

- Let $\Gamma = \Gamma_1 \cup \Gamma_2$. Then Γ is a disjoint union of $\Gamma_{1,1}, \dots, \Gamma_{1,k_1}, \Gamma_{2,1}, \dots, \Gamma_{2,k_2}$.

- $\mathscr{A}$ is a transducer with the input alphabet Σ and the output alphabet Γ. $\mathscr{A}$ nondeterministically chooses $\mathscr{A}_1$ or $\mathscr{A}_2$, then uses $\mathscr{A}_1$ or $\mathscr{A}_2$ to guess a string belonging to Γ^*_{1,j_1} or Γ^*_{2,j_2}, where $j_1 : 1 \leq j_1 \leq k_1$ and $j_2 : 1 \leq j_2 \leq k_2$.
- $\mathscr{L}(\mathscr{B})$ is a disjoint union of $L_{1,1}, \dots, L_{1,k_1}, L_{2,1}, \dots, L_{2,k_2}$.

3. *Intersection.*

If PCAs were closed under intersection, then we demonstrate that the two-counter machines can be simulated by PCAs, thus the nonemptiness of PCAs becomes undecidable, contradicting to the main result of this paper, Theorem 18.

We first illustrate how to reduce the nonemptiness of a two-counter machine $\mathscr{C}$ to that of a class automaton $\mathscr{D} = (\mathscr{A}, \mathscr{B})$. Then we show that the class condition $\mathscr{L}(\mathscr{B})$ is the intersection of the regular languages accepted by two 0-priority finite automata $\mathscr{B}_1$ and $\mathscr{B}_2$. As a result, $\mathscr{L}(\mathscr{D})$ is the intersection of the data languages accepted by two PCAs $\mathscr{D}_1 = (\mathscr{A}, \mathscr{B}_1)$ and $\mathscr{D}_2 = (\mathscr{A}, \mathscr{B}_2)$, and get the desired contradiction.

The intuition of $\mathscr{D}$ is that the transducer $\mathscr{A}$ guesses a run of $\mathscr{C}$ over a word, and $\mathscr{B}$ checks the validity of the guessed run.

A successful run of $\mathscr{C}$ over a word $w \in \Sigma^*$ can be seen as a sequence $(p_1, \sigma_1, l_1, p'_1) \dots (p_n, \sigma_n, l_n, p'_n)$ satisfying the following conditions,

- $\sigma_i \in \Sigma \cup \{\varepsilon\}$, $l_i \in \{inc_1, dec_1, ifz_1, inc_2, dec_2, ifz_2\}$ for each $1 \leq i \leq n$,
- $w = \sigma_1 \dots \sigma_n$,
- $p_1 = q_0$, $p'_n \in F$, $p'_i = p_{i+1}$ for each $1 \leq i < n$,
- $(p_i, \sigma_i, l_i, p'_i) \in \delta$ for each $1 \leq i \leq n$.
- For each $r = 1, 2$, there is an injective mapping φ from the set of positions corresponding to the occurrences of dec_r to the set of positions corresponding to the occurrences of inc_r such that
  - each position i corresponding to an occurrence of dec_r is mapped to some position $j < i$ corresponding to an occurrence of inc_r,
  - for each position i corresponding to an occurrence of ifz_r and for each position $j < i$ corresponding to an occurrence of inc_r, there is a position j' corresponding to an occurrence of dec_r such that $j < j' < i$ and $\varphi(j') = j$.

The input and output alphabet of the transducer $\mathscr{A}$ is $\hat{\Sigma} = Q \times (\Sigma \cup \{\varepsilon\}) \times L \times Q$, where $L = \{inc_1, dec_1, ifz_1, \dots, inc_k, dec_k, ifz_k\}$. $\mathscr{A}$ is the identity transducer, it verifies the following condition over a word $(p_1, \sigma_1, l_1, p'_1) \dots (p_n, \sigma_n, l_n, p'_n) \in \hat{\Sigma}^*$: $p_1 = q_0$, $p'_n \in F$, $p'_i = p_{i+1}$ for each $1 \leq i < n$ and $(p_i, \sigma_i, l_i, p'_i) \in \delta$ for each $1 \leq i \leq n$.

The deterministic complete finite automaton $\mathscr{B}$ verifies the following conditions:

- there are no i, j, r such that $i \neq j$, inc_r occurs in the position i and j, and i, j are in the same class,
- there are no i, j, r such that $i \neq j$, dec_r occurs in the position i and j, and i, j are in the same class,
- for each $r : 1 \leq r \leq k$ and each occurrence of dec_r in the position i, there is an occurrence of inc_r in some position $j < i$, and j is in the same class as i,
- each occurrence of ifz_r is not in the same class as any other position,
- if ifz_r occurs in a position i, then inc_r and dec_r are matched before the position i, i.e. for each inc_r (resp. dec_r) occurring in a position $j < i$, there is a position j' such that j, j' are in the same class, $j < j' < i$ (resp. $j' < j < i$), and dec_r (resp. inc_r) occurs in the position j'.

The automaton $\mathscr{B}$ is illustrated in Figure 2, where

- the double cycles denote the accepting states,
- $\hat{\Sigma}_{inc_1}$ denotes the set of letters in $\hat{\Sigma}$ whose L component is inc_1, similarly for $\hat{\Sigma}_{inc_2}$, $\hat{\Sigma}_{dec_1}$, $\hat{\Sigma}_{dec_2}$, $\hat{\Sigma}_{ifz_1}$, and $\hat{\Sigma}_{ifz_2}$.
- $\hat{\Sigma}_{dec}$ denotes the union of $\hat{\Sigma}_{dec_1}$ and $\hat{\Sigma}_{dec_2}$, and $\hat{\Sigma}_{ifz}$ denotes the union of $\hat{\Sigma}_{ifz_1}$ and $\hat{\Sigma}_{ifz_2}$.

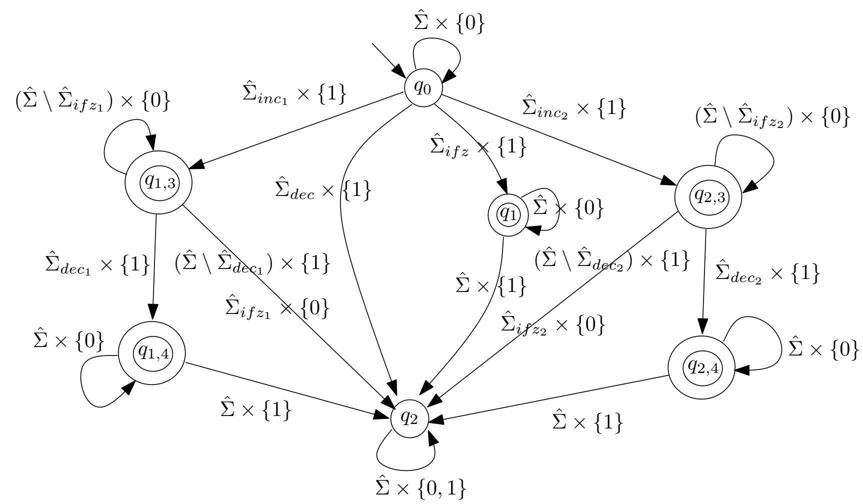

Figure 2: Multicounter automata to class automata: Class condition $\mathscr{B}$

There are the following three situations for a class string to be accepted by $\mathscr{B}$:

- There exists one position in the class string labeled by a letter $\hat{\Sigma}_{ifz} \times \{1\}$, and all the other positions are labeled by the letters from $\hat{\Sigma} \times \{0\}$.
- There exists one position in the class string labeled by a letter from $\hat{\Sigma}_{inc_r} \times \{1\}$, and all the other positions are labeled by the letters from $(\hat{\Sigma} \setminus \hat{\Sigma}_{ifz_r}) \times \{0\}$, where $r = 1,2$.
- There exist exactly two positions in the class string labeled by the letters from $\hat{\Sigma} \times \{1\}$: One position i labeled by a letter from $\hat{\Sigma}_{inc_r} \times \{1\}$ (where $r = 1,2$), and another position j such that $j > i$ labeled by a a letter from $\hat{\Sigma}_{dec_r} \times \{1\}$. Moreover, the letters from $\hat{\Sigma}_{ifz_r} \times \{0\}$ do not occur between any position between i and j (This implies that the zero tests of the counter r are valid).

Since for any $\gamma_1 \in \hat{\Sigma}_{ifz_1}$ and $\gamma_2 \in \hat{\Sigma}_{ifz_2}$, there are both a $((\gamma_1,0),(\gamma_2,0))$-pattern and a $((\gamma_2,0),(\gamma_1,0))$-pattern in $\mathscr{B}$, it follows from Proposition 6 that $\mathscr{B}$ is not a 0-priority finite automaton.

Let $\mathscr{B}_1$ and $\mathscr{B}_2$ be the two automata illustrated in respectively Figure 3 and Figure 4.

The intuition of the automaton $\mathscr{B}_1$ (resp. $\mathscr{B}_2$) is to guarantee the validity of the zero-tests for the first (resp. second) counter, but not the second (resp. the first). The only difference between $\mathscr{B}$ and $\mathscr{B}_1$ (resp. $\mathscr{B}_2$) is the set of transitions out of $q_{2,3}$ (resp. $q_{1,3}$). It is not hard to verify that $\mathscr{L}(\mathscr{B}) = \mathscr{L}(\mathscr{B}_1) \cap \mathscr{L}(\mathscr{B}_2)$.

Moreover, $\mathscr{B}_1$ (resp. $\mathscr{B}_2$) is a 0-priority finite automaton under the ordering $(\hat{\Sigma}_{inc} \cup \hat{\Sigma}_{dec})\hat{\Sigma}_{ifz_2}\hat{\Sigma}_{ifz_1}$ (resp. $(\hat{\Sigma}_{inc} \cup \hat{\Sigma}_{dec})\hat{\Sigma}_{ifz_1}\hat{\Sigma}_{ifz_2}$).

4. *Complementation*.

From the fact that PCAs are closed under union, but not closed under intersection, it is deduced that PCAs are not closed under complementation.

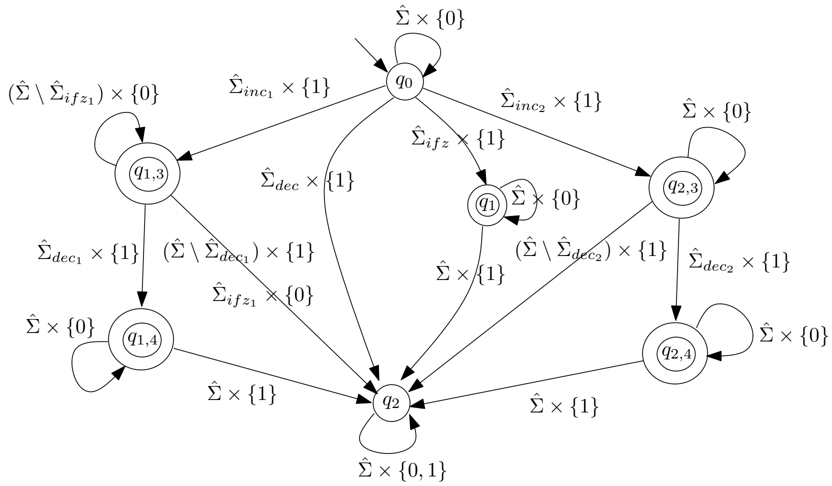

Figure 3: Class condition $\mathscr{B}_1$

This is also witnessed by the language $L_\#$ satisfying the following properties[3],

1. $str(w) \in a^*\$a^*$,
2. the data value of the \$-position occurs exactly once, and each other value occurs exactly twice - once before and once after \$,
3. the order of data values in the first a-block of w is different from the order of data values in the second a-block.

$L_\#$ can be recognized by a data automaton, but it can be shown that its complement is not recognized even by a class automaton. □

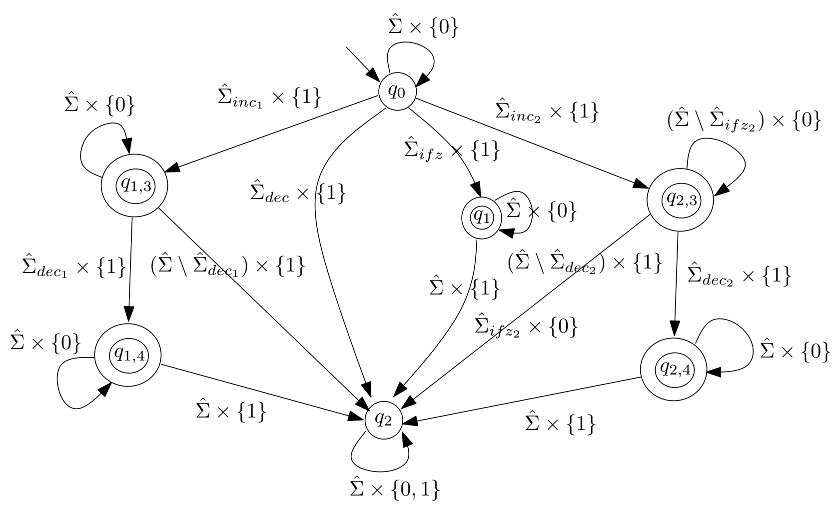

Figure 4: Class condition $\mathscr{B}_2$

[3] This example is from [4]

A.2 Proofs in Section 4

Lemma 21 *For a given PMA $\mathscr{C}$, a PCA $\mathscr{D}$ can be constructed such that $\mathscr{L}(\mathscr{C})$ is a projection of $\mathscr{L}(\mathscr{D})$.*

Proof. Suppose $\mathscr{C} = (Q, \Sigma, k, \delta, q_0, F)$ is a PMA under the ordering of counters $C_1, \dots, C_k$. We show how to construct a PCA $\mathscr{D} = (\mathscr{A}, \mathscr{B})$ and a projection prj such that $\mathscr{L}(\mathscr{C}) = prj(\mathscr{L}(\mathscr{D}))$.

Similar to the simulation of the two-counter machine by a class automaton in the proof of Proposition 15, the idea is that the transducer $\mathscr{A}$ guesses a run of $\mathscr{C}$ over a word, and $\mathscr{B}$ checks the validity of the guessed run.

A run of $\mathscr{C}$ over a word $w \in \Sigma^*$ is a sequence $(p_1, \sigma_1, l_1, p'_1) \dots (p_n, \sigma_n, l_n, p'_n)$ such that

- $\sigma_i \in \Sigma \cup \{\varepsilon\}$, $l_i \in \{inc_1, dec_1, ifz_{\leq 1}, \dots, inc_k, dec_k, ifz_{\leq k}\}$ for each $1 \leq i \leq n$,
- $w = \sigma_1 \dots \sigma_n$,
- $p_1 = q_0$, $p'_n \in F$, $p'_i = p_{i+1}$ for each $1 \leq i < n$,
- $(p_i, \sigma_i, l_i, p'_i) \in \delta$ for each $1 \leq i \leq n$.

The input and output alphabet of the transducer $\mathscr{A}$ is $\hat{\Sigma} = Q \times (\Sigma \cup \{\varepsilon\}) \times L \times Q$, where $L = \{inc_1, dec_1, ifz_{\leq 1}, \dots, inc_k, dec_k, ifz_{\leq k}\}$. $\mathscr{A}$ is the identity transducer, it verifies the following condition over a word $(p_1, \sigma_1, l_1, p'_1) \dots (p_n, \sigma_n, l_n, p'_n) \in \hat{\Sigma}^*$: $p_1 = q_0$, $p'_n \in F$, $p'_i = p_{i+1}$ for each $1 \leq i < n$ and $(p_i, \sigma_i, l_i, p'_i) \in \delta$ for each $1 \leq i \leq n$.

The deterministic complete finite automaton $\mathscr{B}$ verifies the following conditions:

- there are no i, j, r such that $i \neq j$, inc_r occurs in the position i and j, and i, j are in the same class,
- there are no i, j, r such that $i \neq j$, dec_r occurs in the position i and j, and i, j are in the same class,
- for each $r : 1 \leq r \leq k$ and each occurrence of dec_r in the position i, there is an occurrence of inc_r in some position $j < i$, and j is in the same class as i,
- each occurrence of $ifz_{\leq r}$ is not in the same class as any other position,
- if $ifz_{\leq r}$ occurs in a position i, then for each $s \leq r$, inc_s and dec_s are matched before the position i, i.e. for each inc_s (resp. dec_s) occurring in a position $j < i$, there is a position j' such that j, j' are in the same class, $j < j' < i$ (resp. $j' < j < i$), and dec_s (resp. inc_s) occurs in the position j'.

The projection $prj : \hat{\Sigma} \to \Sigma \cup \{\varepsilon\}$ is defined by $prj((q, \sigma, l, q')) = \sigma$ for each $(q, \sigma, l, q') \in \hat{\Sigma}$.

The transition subgraph of the automaton $\mathscr{B}$ is illustrated in Figure 5, where

- the double circles denote the accepting states;
- for each $r : 1 \leq r \leq k$, $\hat{\Sigma}_{inc_r}$ (resp. $\hat{\Sigma}_{dec_r}$, $\hat{\Sigma}_{ifz_{\leq r}}$) denotes the set of letters in $\hat{\Sigma}$ whose L-component is inc_r (resp. dec_r, $ifz_{\leq r}$);
- moreover, $\hat{\Sigma}_{inc}$ (resp. $\hat{\Sigma}_{dec}$, $\hat{\Sigma}_{ifz}$) denotes $\bigcup_r \hat{\Sigma}_{inc_r}$ (resp. $\bigcup_r \hat{\Sigma}_{dec_r}$, $\bigcup_r \hat{\Sigma}_{ifz_{\leq r}}$).

The automaton $\mathscr{B}$ contains the states q_0, q_1, q_2 and the two states $q_{r,3}, q_{r,4}$ for each $r : 1 \leq r \leq k$. There are the following three situations for a class string to be accepted by $\mathscr{B}$:

- There exists one position in the class string labeled by a letter $\hat{\Sigma}_{ifz} \times \{1\}$, and all the other positions are labeled by the letters from $\hat{\Sigma} \times \{0\}$.
- There exists one position in the class string labeled by a letter from $\hat{\Sigma}_{inc_r} \times \{1\}$, and all the other positions are labeled by the letters from $(\hat{\Sigma} \setminus \bigcup\limits_{s \geq r} \hat{\Sigma}_{ifz_{\leq s}}) \times \{0\}$.

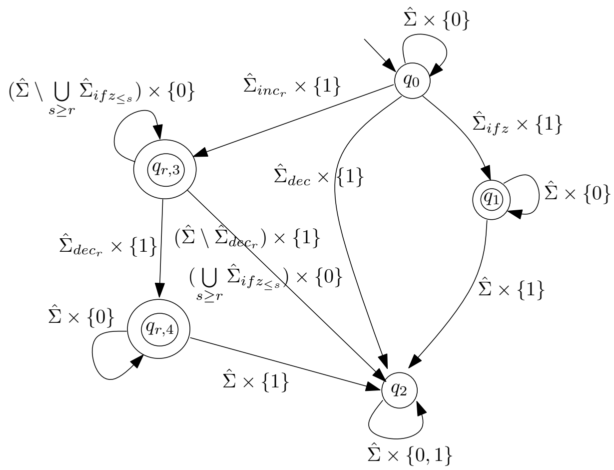

Figure 5: Class condition $\mathscr{B}$: Reduction from PMA to PCA

- There exist exactly two positions in the class string labeled by the letters from $\hat{\Sigma} \times \{1\}$: One position i labeled by a letter from $\hat{\Sigma}_{inc_r} \times \{1\}$, and another position j such that $j > i$ labeled by a a letter from $\hat{\Sigma}_{dec_r} \times \{1\}$. Moreover, the letters from $(\bigcup_{s \geq r} \hat{\Sigma}_{ifz_{\leq s}}) \times \{0\}$ do not occur between any position between i and j.

We claim that $\mathscr{B}$ is a 0-priority finite automaton under the ordering $(\hat{\Sigma}_{inc} \cup \hat{\Sigma}_{dec})\hat{\Sigma}_{ifz_{\leq 1}}\hat{\Sigma}_{ifz_{\leq 2}} \ldots \hat{\Sigma}_{ifz_{\leq r}}$.

We show the claim for the special case that there are only two counters in $\mathscr{C}$. The argument for the more general case that there are more than two counters is similar.

The complete transition graph of $\mathscr{B}$ for the case that there are two counters in $\mathscr{C}$ is illustrated in Figure 6 and its corresponding G_0 is illustrated in Figure 7.

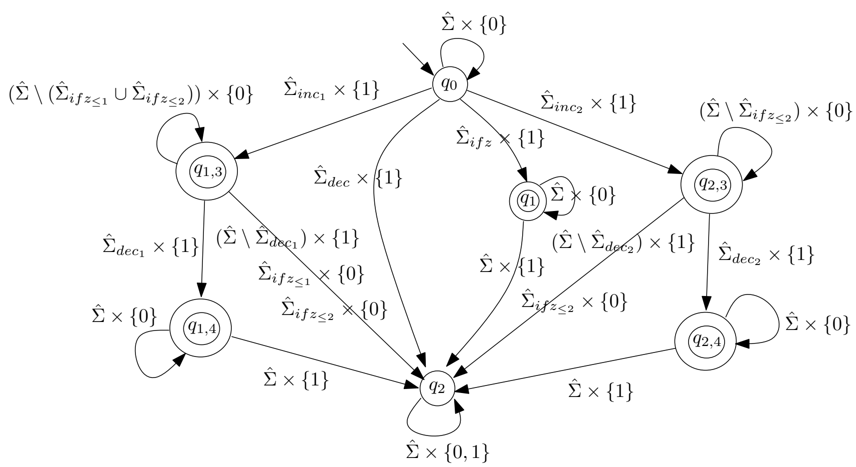

Figure 6: The class condition $\mathscr{B}$: The two counters

If $\hat{\Sigma}$ is ordered as $(\hat{\Sigma}_{inc} \cup \hat{\Sigma}_{dec})\hat{\Sigma}_{ifz_{\leq 1}}\hat{\Sigma}_{ifz_{\leq 2}}$, then from G_0 in Figure 7, we know that

- $q_{1,3}$ is $(\hat{\sigma}, 0)$-acyclic for any $\hat{\sigma} \in \hat{\Sigma}_{ifz_{\leq 1}} \cup \hat{\Sigma}_{ifz_{\leq 2}}$, and $(\hat{\sigma}, 0)$-cyclic for any $\hat{\sigma} \in \hat{\Sigma} \setminus (\hat{\Sigma}_{ifz_{\leq 1}} \cup \hat{\Sigma}_{ifz_{\leq 2}})$.

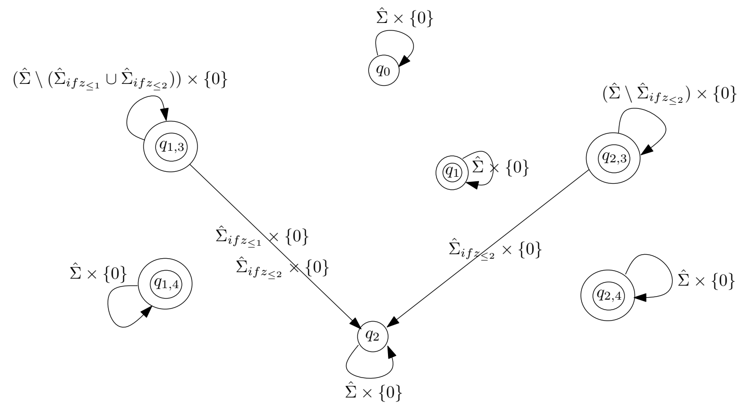

Figure 7: G_0 of $\mathscr{B}$: The two counters

- $q_{2,3}$ is $(\hat{\sigma}, 0)$-acyclic for any $\hat{\sigma} \in \hat{\Sigma}_{ifz_{\leq 2}}$, and $(\hat{\sigma}, 0)$-cyclic for any $\hat{\sigma} \in \hat{\Sigma} \setminus \hat{\Sigma}_{ifz_{\leq 2}}$.
- all the other states are $(\hat{\sigma}, 0)$-cyclic for any $\hat{\sigma} \in \hat{\Sigma}$.

So

- $(q_{1,3}, q_{1,3}, q_{1,3}, q_2)$ is a $((\hat{\sigma}_1, 0), (\hat{\sigma}_2, 0))$-pattern for any $\hat{\sigma}_1 \in \hat{\Sigma} \setminus (\hat{\Sigma}_{ifz_{\leq 1}} \cup \hat{\Sigma}_{ifz_{\leq 2}}) = \hat{\Sigma}_{inc} \cup \hat{\Sigma}_{dec}$ and $\hat{\sigma}_2 \in \hat{\Sigma}_{ifz_{\leq 1}} \cup \hat{\Sigma}_{ifz_{\leq 2}}$ in $\mathscr{B}$.
- $(q_{2,3}, q_{2,3}, q_{2,3}, q_2)$ is a $((\hat{\sigma}_1, 0), (\hat{\sigma}_2, 0))$-pattern for any $\hat{\sigma}_1 \in \hat{\Sigma} \setminus \hat{\Sigma}_{ifz_{\leq 2}} = \hat{\Sigma}_{inc} \cup \hat{\Sigma}_{dec} \cup \hat{\Sigma}_{ifz_{\leq 1}}$ and $\hat{\sigma}_2 \in \hat{\Sigma}_{ifz_{\leq 2}}$ in $\mathscr{B}$.
- There are no other $((\hat{\sigma}_1, 0), (\hat{\sigma}_2, 0))$ patterns for $\hat{\sigma}_1, \hat{\sigma}_2 \in \hat{\Sigma}$ in $\mathscr{B}$.

Therefore, we conclude that $\mathscr{B}$ is a 0-priority finite automaton under the ordering $(\hat{\Sigma}_{inc} \cup \hat{\Sigma}_{dec})\hat{\Sigma}_{ifz_{\leq 1}}\hat{\Sigma}_{ifz_{\leq 2}}$. □

Lemma 22. *Let $\mathscr{D} = (\mathscr{A}, \mathscr{B})$ be a PCA such that $\mathscr{B}$ is a 0-priority finite automaton. Then any abstract run of $\mathscr{D}$ over a data word (w, π), say $(q^g_1, q^c_1, \gamma_1, C_1) \ldots (q^g_{|w|}, q^c_{|w|}, \gamma_{|w|}, C_{|w|})$, enjoys the following property:*

For each $i : 1 \leq i \leq |w|$, the sum of $C_i(q')$'s such that q' is 0-acyclic is bounded by $\#_{scc}(G_0)$.

The proof of Lemma 22 is essentially the same as that of Lemma 1 in [1]. The following proof is given for reader's convenience.

Proof. Suppose (w, π) is a data word, $(q^g_1, q^c_1, \gamma_1, C_1) \ldots (q^g_{|w|}, q^c_{|w|}, \gamma_{|w|}, C_{|w|})$ is an abstract run $\mathscr{D}$ over w, and $(q^g_1, q^c_1, \gamma_1, R_1) \ldots (q^g_{|w|}, q^c_{|w|}, \gamma_{|w|}, R_{|w|})$ is the corresponding (concrete) run.

Let $i : 1 \leq i \leq |w|$.

If $i \leq \#_{scc}(G_0)$, then obviously the sum of $C_i(q')$'s such that q' is 0-acyclic is bounded by $\#_{scc}(G_0)$, since there are at most $\#_{scc}(G_0)$ data values occurring before the position i, and each data value can only increase the sum by one.

Now suppose that $i > \#_{scc}(G_0)$.

Let $C_i(q') > 0$ such that q' is 0-acyclic and $R_i(d) = q'$ for some data value d occurring before the position i (including i).

Then we claim that the data value d must occur at least once between the position $i - \#_{scc}(G_0) + 1$ and the position i.

To the contrary, suppose that d does not occur between the position $i - \#_{scc}(G_0) + 1$ and the position i. Then d occurs in some position j such that $j \leq i - \#_{scc}(G_0)$.

Let $R_j(d) = q''$.

Since d does not occur between the position $i - \#_{scc}(G_0) + 1$ and the position i, q' is obtained from q'' by $i - j \geq \#_{scc}(G_0)$ number of 0-transitions in the copy of $\mathscr{B}$ corresponding to the data value d.

If q'' is 0-cyclic, then according to Corollary 9, in a 0-priority finite automaton $\mathscr{B}$, only 0-cyclic states can be reached from 0-cyclic states, so q' is 0-cyclic, contradicting to the assumption that q' is 0-acyclic.

If q'' is 0-acyclic, because $\mathscr{B}$ is a 0-priority finite automaton, only 0-cyclic states can be reached from 0-cyclic states, it follows that any state reachable from a 0-acyclic state by a path of length at least $\#_{scc}(G_0)$ in G_0 must be 0-cyclic, so q' must be 0-cyclic, contradicting to the assumption as well.

Therefore, the claim holds.

Since there are at most $\#_{scc}(G_0)$ distinct data values between the position $i - \#_{scc}(G_0) + 1$ and the position i, it follows that the number of data values d such that $R_i(d) = q'$ and q' is 0-acyclic is at most $\#_{scc}(G_0)$, namely, the sum of $C_i(q')$'s such that q' is 0-acyclic is at most $\#_{scc}(G_0)$. □

Proposition 23. *Let $\mathscr{D} = (\mathscr{A}, \mathscr{B})$ be a PCA such that $\mathscr{B}$ is a 0-priority finite automaton under the ordering $\gamma_1 \dots \gamma_l$. Then $Acyc_1, \dots, Acyc_{l+1}$ satisfy the following two properties:*

1. *$Acyc_i \subseteq Acyc_{i+1}$ for each $i < l$.*
2. *For each $i : 1 \leq i \leq l$, if $q \in Acyc_i$ and $q \xrightarrow{(\gamma_i,0)} q'$, then $q' \notin Acyc_i$ and $q' \in Acyc_j$ for some $j > i$. In particular, if $q \in Acycl_l$ and $q \xrightarrow{(\gamma_l,0)} q'$, then $q' \notin Acyc_l$ and $q' \in Acyc_{l+1}$.*

Proof.

1. To the contrary, suppose that there is a state q such that $q \in Acyc_i \setminus Acyc_{i+1}$ for some $i < l$.

Then q is $(\gamma_i, 0)$-acyclic, so there exists $q' \in Q_c$ such that $q \xrightarrow{(\gamma_i,0)} q'$. Moreover, $q \notin Acyc_{i+1}$, thus q is $(\gamma_{i+1}, 0)$-cyclic, so there exists $q'' \in Q_c$ such that q'' is 0-cyclic and $q'' \xrightarrow{(\gamma_{i+1},0)} q$. It follows that $q'' \xrightarrow{(\gamma_{i+1},0)} q \xrightarrow[0]{*} q \xrightarrow{(\gamma_i,0)} q'$, so (q'', q, q, q') is a $((\gamma_{i+1}, 0), (\gamma_i, 0))$-pattern in $\mathscr{B}$, contradicting to the fact that $\mathscr{B}$ is a 0-priority finite automaton under the ordering $\gamma_1 \dots \gamma_l$.

2. Suppose that $q \in Acyc_i$, and $q \xrightarrow{(\gamma_i,0)} q'$ for some $i : 1 \leq i \leq l$.

From Corollary 9, it follows that q' is also 0-cyclic, so $q' \in Acyc_j$ for some $j : 1 \leq j \leq l+1$. Let j be such a minimal number.

We claim that $j > i$. To the contrary, suppose that $j \leq i$.

Since q' is $(\gamma_j, 0)$-acyclic, there exists $q'' \in Q_c$ such that $q \xrightarrow{(\gamma_i,0)} q' \xrightarrow[0]{*} q' \xrightarrow{(\gamma_j,0)} q''$, $i \geq j$, q is 0-cyclic and q' is $(\gamma_j, 0)$-acyclic, so (q, q', q', q'') is a $((\gamma_i, 0), (\gamma_j, 0))$-pattern in $\mathscr{B}$, contradicting to the fact that $\mathscr{B}$ is a 0-priority finite automaton under the ordering $\gamma_1 \dots \gamma_l$.

Therefore $q' \notin Acyc_i$ and $q' \in Acyc_j$ for some $j > i$. □

A.3 Proofs in Section 5

Lemma 25. *For a Restricted-ND_2 program P and a Boolean state m, a class automaton $\mathscr{D} = (\mathscr{A}, \mathscr{B})$ can be constructed such that m is reached from the initial state after the run of P over an array A iff the array (data word) A is accepted by $\mathscr{D}$.*

Though the proof of Lemma 25 in the following is essentially the same as in [1], it contains more details and is presented here for reader's convenience.

Proof. Let P be a Restricted-ND_2 program of the following form,

```
for i:=1 to length(A) do
{
  P1;
  for j:=1 to length(A) do
  {
    if A[i].d=A[j].d then
             P2
    else
             P3
   };
   P4
}
```

Let m be a Boolean state of P.

For simplicity, we first assume that no constants $c, c_1, \cdots \in \mathbb{D}$ are used in the Restricted ND_2 program P and no Boolean expressions $i = j$ occur in the inner loop of P.

Let $Bol(P)$ denote the set of all the Boolean states of P, namely, the set of all assignments to the Boolean variables occurring in P.

The general idea is that $\mathscr{A}$ guesses a run of the outer loop over an array (data word) A, and $\mathscr{B}$ corresponds to the inner loop and verifies the consistency of the guessed run.

Specifically, over an array $(A[1].s, A[1].d)(A[2].s, A[2].d)\dots(A[n].s, A[n].d)$, $\mathscr{A}$ generates a string $w' = (A[1].s, m_1, m'_1)(A[2].s, m_2, m'_2)\dots(A[n].s, m_n, m'_n)$ such that for each $i : 1 \leq i \leq n$, $m_i, m'_i \in Bol(P)$. In addition, $\mathscr{A}$ verifies that

- for each $i : 1 \leq i < n$, m_{i+1} is obtained from m'_i by running P_4 over the position i of A, and P_1 over the position $i+1$ of A;
- m_1 is the initial Boolean state (Namely, all the Boolean variables have the value $false$), and $m'_n = m$.

The goal of $\mathscr{B}$ is to check whether for each $i : 1 \leq i \leq n$, m'_i is obtained from m_i by running the inner loop of P over A, when the outer loop is in the position i. Specifically,

over a class string $w' \otimes X$, say

$$(A[1].s, m_1, m'_1, 0)\dots(A[i_1].s, m_{i_1}, m'_{i_1}, 1)\dots(A[i_2].s, m_{i_2}, m'_{i_2}, 1)\dots(A[n].s, m_n, m'_n, 0),$$

for each position $i \in X$, e.g. the position i_1, $\mathscr{B}$ checks whether m'_i is obtained from m_i by running the inner loop of P over A when the outer loop is in the position i.

Note that if $A[i_1].s = A[i_2].s$ and $A[i_1].d = A[i_2].d$, then the two runnings of the inner loop of P over A when the outer loop is in the position i_1 and when the outer loop is in the position i_2 are the same.

So for each $\sigma \in \Sigma$, an automaton $\mathscr{B}_\sigma$ is constructed such that

over a class string $w' \otimes X$, say

$$(A[1].s, m_1, m'_1, 0)\dots(A[i_1].s, m_{i_1}, m'_{i_1}, 1)\dots(A[i_2].s, m_{i_2}, m'_{i_2}, 1)\dots(A[n].s, m_n, m'_n, 0),$$

for each position $i \in X$ satisfying that $A[i].s = \sigma$, $\mathscr{B}_\sigma$ checks whether m'_i is obtained from m_i by running the inner loop of P over A when the outer loop is in the position i.

The desired automaton $\mathscr{B}$ is the product of $\mathscr{B}_\sigma$'s for $\sigma \in \Sigma$.

The automaton $\mathscr{B}_\sigma$ is constructed from the inner loop of P, namely P_2 and P_3, as follows:

- Replace each occurrence of $A[i].s$ in $P2$ and $P3$ by σ. After this, there are no occurrences of $A[i].s$ in $P2, P3$.
- A Boolean state reachability graph $H_1 = (Bol(P), \delta_1)$ with the arcs labeled by letters from $\Sigma \times \{1\}$ is constructed from $P2$, where the letters $(\sigma', 1)$ correspond to the possible values of $A[j].s$.
- Similarly, a Boolean state reachability graph $H_0 = (Bol(P), \delta_0)$ with the arcs labeled by letters from $\Sigma \times \{0\}$ is constructed from $P3$.
- For each Boolean state m, $\mathscr{B}_\sigma$ records the state pairs (m, m') such that m' is reached from m after the running of the inner loop over A when the outer loop is in the position $i \in X$ such that $A[i].s = \sigma$. Let *Reach* denote the set of these recorded state pairs (m, m'). Moreover, $\mathscr{B}_\sigma$ records all the pairs (m_j, m'_j) occurring in the letters $(A[j].s, m_j, m'_j, 1)$ that have been met when reading $w' \otimes X$. Let *Acc* denote the set of these recorded state pairs (m_j, m'_j).
- While reading a class string $w' \otimes X$,
  - if a letter $(A[j].s, m_j, m'_j, 0)$ is met, then $\mathscr{B}_\sigma$ replaces each recorded state pair $(m, m') \in Reach$ with the set $\{(m, m'') \mid m'' \in \delta_0(m', (A[j].s, 0))\}$,
  - if a letter $(A[j].s, m_j, m'_j, 1)$ is met, then $\mathscr{B}_\sigma$ replaces each recorded state pair $(m, m') \in Reach$ with the set $\{(m, m'') \mid m'' \in \delta_1(m', (A[j].s, 1))\}$; moreover, if $A[j].s = \sigma$, then it puts the state pair (m_j, m'_j) into *Acc*.
- $\mathscr{B}_\sigma$ accepts a class string $w' \otimes X$ if $Acc \subseteq Reach$ after reading $w' \otimes X$ (This means that all the guessed runs (m_j, m'_j)'s of the outer loop such that $j \in X$ and $A[j].s = \sigma$ are successful).

The situation that the constants $c, c_1 \cdots \in \mathbb{D}$ are used in P can be handled as follows:

Let $c_1, \ldots, c_k$ be the constants occurring in P.

- At each position i, $\mathscr{A}$ guesses either $(A[i].s, m_i, m'_i, c_l)$ or $(A[i].s, m_i, m'_i, c'_l)$ or $(A[i].s, m_i, m'_i, e)$ for some $l : 1 \leq l \leq k$; in addition, $\mathscr{A}$ guarantees that for each $l : 1 \leq l \leq k$, at most one c'_l is guessed when reading A, and once c'_l is guessed in some position, c_l cannot be guessed after. Intuitively, c'_l is the rightmost guessing of the constants c_l, and e denotes the data values not equal to these constants.
- $\mathscr{B}_\sigma$ checks in addition whether for each class string $w' \otimes X$, the projection of the fourth component of the substring of $w' \otimes X$ restricted to the positions in X, is either e^* or $c_l^* c'_l$ for some $l : 1 \leq l \leq k$. By doing this, each guessed constant c_l cannot occur in positions belonging to different classes. Notice that this checking only concerns the one-transitions of $\mathscr{B}_\sigma$.

A slight modification of the one-transitions in $\mathscr{B}_\sigma$ can also be done to handle the situation that the expression $i = j$ occurs in P, more precisely in $P2$, since $i = j$ is trivially false in $P3$. □

Theorem 28. *The Boolean state reachability problem is decidable for* 0*-priority restricted-ND_2 programs.*

Proof. It is sufficient to show that the class automata $\mathscr{D} = (\mathscr{A}, \mathscr{B})$ constructed from 0-priority restricted-ND_2 programs are PCAs.

Let P be a 0-priority restricted-ND_2 program.

If $P3$ in P does not refer to $A[j]$, then $\mathscr{D}$ constructed from P is an extended data automaton, thus a PCA, as shown in [1].

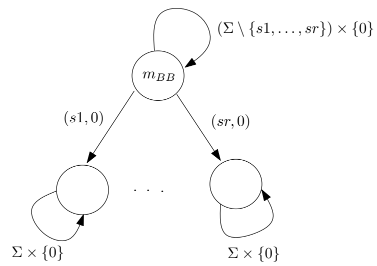

Figure 8: The structure of H_0 constructed from $P3$

Now consider the situation where $P3$ refers to $A[j].s$ and is of the form specified in the definition of 0-priority restricted-ND_2 programs.

The Boolean state reachability graph H_0 constructed from $P3$ (c.f. the proof of Lemma 25) has the structure illustrated in Figure 8, where m_{BB} denotes any Boolean state satisfying the condition BB, and any $(sl, 0)$-successor $(1 \leq l \leq r)$ of m_{BB} does not satisfy the condition BB since the assignments $PA1, \ldots, PAr$ are nontrivial in $P3$.

So under the ordering $(\Sigma \setminus \{s1, \ldots, sr\})s1, \ldots, sr$, H_0 satisfies the restriction of 0-priority finite automata. From this fact and the construction of $\mathscr{B}_\sigma$ from H_1 and H_0, it is not hard to show that the zero-transition graph of $\mathscr{B}_\sigma$ also satisfies the restriction of the 0-priority finite automata. Moreover, because $A[i].s$ does not occur in $P3$ from the definition of 0-priority restricted-ND_2 programs, it follows that the zero-transition graphs of the different $\mathscr{B}_\sigma$'s are in fact the same. Thus the zero-transition graph of $\mathscr{B}$, as a product of the automata $\mathscr{B}_\sigma$'s, has the same structure as that of $\mathscr{B}_\sigma$, so satisfies the restriction of the 0-priority finite automata as well. $\square$