Monadic Quantifiers Recognized by Deterministic Pushdown Automata

Makoto Kanazawa

National Institute of Informatics, Tokyo, Japan
kanazawa@nii.ac.jp

Abstract

I characterize the class of type $\langle 1\rangle$ quantifiers (or, equivalently, type $\langle 1,1\rangle$ quantifiers satisfying Conservativity and Extension) that are recognized by deterministic pushdown automata in terms of the associated semilinear sets of vectors in $\mathbb{N}^2$. These semilinear sets are finite unions of linear sets with at most two generators each, which are taken from a common three-element set of the form $\{(k,0),(0,l),(m,n)\}$. This answers a question that was left open by Mostowski (1998). A consequence of my characterization is that the type $\langle 1\rangle$ quantifiers recognized by deterministic pushdown automata are already recognized by deterministic *one-counter* machines with zero tests, i.e., deterministic pushdown automata whose stack alphabet contains just one symbol (besides the bottom-of-stack symbol).

1 Introduction

A *type* $\langle 1\rangle$ *quantifier* is a class of finite first-order structures of the form (U,P), with $P \subseteq U$, which is closed under isomorphism. (We allow $U = \varnothing$.) Some examples of type $\langle 1\rangle$ quantifiers are[1]

$$\begin{aligned}\exists &= \{\,(U,P) \mid P \neq \varnothing\,\},\\ \forall &= \{\,(U,P) \mid U = P\,\},\\ \mathbf{D}_n &= \{\,(U,P) \mid n \text{ divides } |P|\,\} \quad (n = 1,2,\dots),\\ Q^R &= \{\,(U,P) \mid |U-P| < |P|\,\}.\end{aligned}$$

Linguistically, the interest of type $\langle 1\rangle$ quantifiers mostly owes to their correspondence with a subclass of the *type* $\langle 1,1\rangle$ *quantifiers*, which are isomorphism-closed classes of first-order structures of the form (U,P_1,P_2) (with $P_1,P_2 \subseteq U$). The correspondence is through the operation of *relativization*:

$$Q^{\mathrm{rel}} = \{\,(U,P_1,P_2) \mid (P_1, P_1 \cap P_2) \in Q\,\}.$$

Relativizations of type $\langle 1\rangle$ quantifiers are precisely those type $\langle 1,1\rangle$ quantifiers satisfying *Conservativity* and *Extension* (see Peters and Westerståhl, 2006):

$$\begin{aligned}&\text{Conservativity:} && (U,P_1,P_2) \in Q \iff (U,P_1,P_1\cap P_2) \in Q.\\ &\text{Extension:} && (U,P_1,P_2) \in Q \iff (P_1\cup P_2,P_1,P_2) \in Q.\end{aligned}$$

Many natural language determiners apparently express type $\langle 1,1\rangle$ quantifiers satisfying Conservativity and Extension:

$$\begin{aligned}\textit{some} &= \{\,(U,P_1,P_2) \mid P_1\cap P_2 \neq \varnothing\,\},\\ \textit{every} &= \{\,(U,P_1,P_2) \mid P_1 \subseteq P_2\,\},\\ \textit{an-even-number-of} &= \{\,(U,P_1,P_2) \mid |P_1\cap P_2| \text{ is even}\,\},\\ \textit{more-than-half-of} &= \{\,(U,P_1,P_2) \mid |P_1 - P_2| < |P_1 \cap P_2|\,\}.\end{aligned}$$

[1]Here, $|P|$ denotes the cardinality of the set P. Elsewhere, we sometimes write $|w|$, where w is a string, to denote the length of w. Context should make it clear which is intended.

These are relativizations of $\exists, \forall, \mathbf{D}_2, Q^R$, respectively.

Because of the correspondence via relativization, classifications of type $\langle 1 \rangle$ quantifiers in terms of their complexity translate into classifications of type $\langle 1, 1 \rangle$ quantifiers satisfying Conservativity and Extension, and vice versa. In this paper, I mostly speak of type $\langle 1 \rangle$ quantifiers, but all the results equally pertain to type $\langle 1, 1 \rangle$ quantifiers satisfying Conservativity and Extension.

Because of isomorphism closure, each type $\langle 1 \rangle$ quantifier Q has two alternative presentations: (i) a set V_Q of vectors in $\mathbb{N}^2$, and (ii) a commutative (i.e., permutation-closed) set W_Q of strings over a two-letter alphabet, say, $\{a, b\}$:

$$V_Q = \{\, (|U - P|, |P|) \mid (U, P) \in Q \,\}, \qquad W_Q = \{\, w \in \{a, b\}^* \mid \#(w) \in V_Q \,\}.$$

Here, $\#(w)$ is the Parikh vector associated with w, defined by

$$\#(w) = (\#_a(w), \#_b(w)),$$

where $\#_c(w)$ denotes the number of occurrences of symbol c in w. Conversely, any subset of $\mathbb{N}^2$ (and any commutative subset of $\{a, b\}^*$) determines a type $\langle 1 \rangle$ quantifier (and via relativization, a type $\langle 1, 1 \rangle$ quantifier satisfying Conservativity and Extension). These correspondences are bijections.

Van Benthem (1986) studied the relationship among the three presentations of quantifiers, proving, among other things, that W_Q is accepted by a nondeterministic pushdown automaton (PDA) if and only if V_Q is a *semilinear* subset of $\mathbb{N}^2$. Since one of the motivations for using automata to classify quantifiers was to bring a procedural perspective to natural language semantics, it makes sense, as noted by van Benthem (1986), to investigate the effect of imposing *determinism* on automata, since nondeterministic automata do not correspond to well-defined algorithms (on a sequential model of computation).[2] It is known that deterministic PDAs accept a proper subclass of the context-free languages, known as the *deterministic context-free languages* (DCFL), which is closed under complementation, but not union. It is certainly interesting to obtain a van Benthem-style characterization of type $\langle 1 \rangle$ quantifiers Q such that W_Q is recognized by a deterministic PDA, in terms of the associated set V_Q of vectors in $\mathbb{N}^2$.

A partial result in this direction was obtained by Mostowski (1998), who gave a characterization of the class of quantifiers that are accepted by deterministic PDAs *by empty stack* using a restricted class of semilinear sets which he called *almost linear*.[3] This result does not cover all quantifiers that are accepted by deterministic PDAs by final state (and arbitrary stack), including such mundane quantifiers as Q^R (*more-than-half-of-all-things*).

I this paper, I give a complete characterization of the semilinear sets of vectors in $\mathbb{N}^2$ that correspond to type $\langle 1 \rangle$ quantifiers recognized by deterministic PDAs (by final state and arbitrary stack). These semilinear sets are finite unions of linear sets with at most two generators each, which are taken from a common three-element set of the form $\{(k, 0), (0, l), (m, n)\}$. One

[2]It would be unrealistic to assume that the human cognitive process of evaluating quantified sentences under normal circumstances even remotely resembles computation on finite or pushdown automata, whose access to the input (the string representation of the described situation) is limited to one-time, one-way scan. Nevertheless, these automata can make finer distinctions than standard models of computation that are used to define computational complexity classes, and are sometimes useful to classify problems that are already of very low computational complexity. For experimental studies of the difficulty of evaluating sentences with different quantifiers, see Szymanik and Zajenkowski 2011, Zajenkowski and Szymanik 2013, and references cited therein.

[3]Mostowski's definition of acceptance by empty stack is not altogether clear, and seems to be different from the standard one given in Section 2.2 below. Presumably, he works with a model that allows both pushing onto empty stack and testing stack for emptiness. At any rate, not all DCFLs are accepted by empty stack under Mostowski's model, even when restricted to commutative languages over $\{a, b\}$, as Mostowski notes.

consequence of this characterization is that a type $\langle 1 \rangle$ quantifier Q is recognized by a deterministic PDA if and only if W_Q is *real-time* recognizable by a (deterministic) *one-counter machine* (Fischer et al., 1968).

2 Preliminaries

2.1 Semilinear Sets

A subset of $\mathbb{N}^n$ is *linear* if it is of the form

$$L(\vec{v}_0; \{\vec{v}_1, \ldots, \vec{v}_r\}) = \{\, \vec{v}_0 + k_1\vec{v}_1 + \cdots + k_r\vec{v}_r \mid k_i \in \mathbb{N}\ (1 \leq i \leq r) \,\}, \tag{†}$$

where $\vec{v}_0, \vec{v}_1, \ldots, \vec{v}_r$ are elements of $\mathbb{N}^n$. The vectors $\vec{v}_1, \ldots, \vec{v}_r$ are called the *generators* of this set, and the vector $\vec{v}_0$ is called its *offset*.[4]

A *semilinear* set is a finite union of linear sets. It is well known (Ginsburg and Spanier, 1966) that the semilinear subsets of $\mathbb{N}^n$ are precisely those definable by formulas of *Presburger arithmetic*, the first-order language with addition as its only function symbol. It is known that every semilinear set $S \subseteq \mathbb{N}^2$ can be expressed as a finite union of linear sets $S_1 \cup \cdots \cup S_q$ each of which has at most two generators (Abe, 1995).

Let $\Sigma = \{a_1, \ldots, a_n\}$. If $L \subseteq \Sigma^*$, then the *Parikh image* of L is $\#(L) = \{\, \#(w) \mid w \in L \,\}$, where $\#(w) = (\#_{a_1}(w), \ldots, \#_{a_n}(w))$. Parikh's theorem (Parikh, 1966) states that every context-free language has a semilinear Parikh image.

2.2 Pushdown Automata

We adopt a fairly standard definition of the pushdown automaton given by Hopcroft and Ullman (1979). A *pushdown automaton* (PDA) is a system $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, where Q is a finite set of *states*, Σ is finite set called the *input alphabet*, Γ is a finite set called the *stack alphabet*, $q_0 \in Q$ is the *initial state*, $Z_0 \in \Gamma$ is the *start symbol*, $F \subseteq Q$ is the set of *final states*, and δ is a mapping from $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ to finite subsets of $Q \times \Gamma^*$.[5] A *configuration* of M is a triple $(q, w, \gamma) \in Q \times \Sigma^* \times \Gamma^*$. An initial configuration is (q_0, w, Z_0), and the *transition relation* $\vdash_M$ between configurations is defined by

$$\vdash_M = \{\, ((q, aw, Z\alpha), (p, w, \beta\alpha)) \mid (p, \beta) \in \delta(q, a, Z) \,\},$$

where a ranges over $\Sigma \cup \{\varepsilon\}$. We write $\vdash_M^*$ for the reflexive transitive closure of $\vdash_M$. We say that M *accepts* a language $L \subseteq \Sigma^*$ *by final state* when

$$L = \{\, w \in \Sigma^* \mid (q_0, w, Z_0) \vdash_M^* (p, \varepsilon, \gamma) \text{ for some } p \in F \text{ and } \gamma \in \Gamma^* \,\}.$$

The PDA M *accepts* L *by empty stack* if

$$L = \{\, w \in \Sigma^* \mid (q_0, w, Z_0) \vdash_M^* (p, \varepsilon, \varepsilon) \text{ for some } p \in Q \,\}.$$

A PDA M is *deterministic* if every configuration allows transition to at most one configuration. Formally, $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is deterministic if (i) $\delta(q, \varepsilon, Z) \neq \varnothing$ implies $\delta(q, a, Z) = \varnothing$ for all $a \in \Sigma$; and (ii) $\delta(q, a, Z)$ contains at most one element for all $a \in \Sigma \cup \{\varepsilon\}$.

[4] Note that the generators and the offset of a linear set depend on its representation in the form of (†).

[5] We write ε for the empty string and A^* for the set of strings over the alphabet A.

A language is a *deterministic context-free language* (DCFL) if some deterministic PDA accepts it by final state.

With nondeterministic PDAs, acceptance by empty stack is equivalent to acceptance by final state. This equivalence does not hold for deterministic PDAs, however. If some deterministic PDA accepts L by empty stack, then there is a deterministic PDA that accepts L by final state, but the converse does not hold. Since no transition to a new configuration is possible once the stack becomes empty, determinism means that whenever a string w is accepted by empty stack, no string of the form wv with $v \neq \varepsilon$ can be so accepted. Thus, any language L that some deterministic PDA accepts by empty stack is *prefix-free* in the sense that no proper prefix of an element of L belongs to L.[6]

It is known that infinite loops and blocking can be eliminated from deterministic PDAs, which makes it possible to use computation on a deterministic PDA as a recognition algorithm for the language it accepts (either by final state or by empty stack).[7] For this reason, when a deterministic PDA M accepts a language L by final state, we say that M *recognizes* L. (When the acceptance is by empty stack, we say "recognizes by empty stack".)

3 Main Result

We say that a deterministic PDA M *recognizes* a type $\langle 1 \rangle$ quantifier Q if M recognizes W_Q.

Theorem 1. *A type $\langle 1 \rangle$ quantifier Q is recognized by a deterministic PDA if and only if there exist natural numbers k, l, m, n such that V_Q is a finite union of linear sets each of which has one of the following as its set of generators:*

$$\varnothing, \quad \{(k,0)\}, \quad \{(0,l)\}, \quad \{(m,n)\}, \quad \{(k,0),(m,n)\}, \quad \{(0,l),(m,n)\}.$$

Example 2. Examples of quantifiers that satisfy the condition in Theorem 1 are

- *more than two thirds*, with the associated set of vectors $L((0,1);\{(0,1),(1,2)\})$;
- *there are an odd number more Ps than non-Ps* (i.e., *the Ps outnumber the non-Ps by an odd number*), with the associated set of vectors $L((0,1);\{(0,2),(1,1)\})$;
- *either there are three more than twice as many Ps as non-Ps or there are less than twice as many Ps as non-Ps*, with the associated set of vectors $L((0,3);\{(1,2)\}) \cup L((1,0);\{(1,0),(1,2)\}) \cup L((1,1);\{(1,0),(1,2)\})$.

In contrast, a semilinear quantifier like *more than one third but less than two thirds*, whose associated set of vectors $L((1,1);\{(2,1),(1,2)\}) \cup L((2,2);\{(2,1),(1,2)\})$ involves two non-trivial ratios, is excluded by the theorem.

The proof of the theorem in one direction relies on the following corollary of the pumping lemma for DCFLs:

Lemma 3 (Harrison 1978)**.** *Let $L \subseteq \Sigma^*$ be a DCFL. There exists a positive integer p satisfying the following property: for every $w \in L$ with $|w| \geq p$, there exist x_1, x_2, x_3, x_4, x_5 such that*

(i) $w = x_1 x_2 x_3 x_4 x_5$;

[6] The languages that some deterministic PDA accepts by empty stack coincide with the languages generated by *LR(0) grammars* (see Hopcroft and Ullman, 1979). Mostowski (1998), who gave a characterization of type $\langle 1 \rangle$ quantifiers recognized by deterministic PDAs by empty stack, seems to have a different conception of PDA, since he allows transition from a configuration with empty stack. See footnote 3.

[7] This also implies that the class of DCFLs is closed under complementation.

(ii) *$x_2x_4 \neq \varepsilon$;*

(iii) *for every $z \in \Sigma^*$ and $n \in \mathbb{N}$, $x_1x_2x_3x_4z \in L$ if and only if $x_1x_2^nx_3x_4^nz \in L$.*

Another simple but useful lemma is the following:

Lemma 4. *Suppose that a semilinear set $S \subseteq \mathbb{N}^2$ is bounded in the first component, i.e., there is a p such that $S \subseteq [0,p] \times \mathbb{N}$. Then there is an l such that S is a finite union of linear sets each of which has $\varnothing$ or $\{(0,l)\}$ as its set of generators. (Analogously for the second component.)*

To prove the "only if" direction of Theorem 1, suppose that W_Q is recognized by a deterministic PDA. By Parikh's theorem, V_Q is semilinear. If W_Q is finite, then V_Q is a finite union of singletons, which are linear sets with $\varnothing$ as the set of generators. If W_Q is infinite, then there must be $w = x_1x_2x_3x_4x_5 \in W_Q$ that satisfy the conditions (i)–(iii) of Lemma 3. Let $\vec{u}_0 = \#(x_1x_3)$ and $\vec{u}_1 = \#(x_2x_4)$. Define[8]

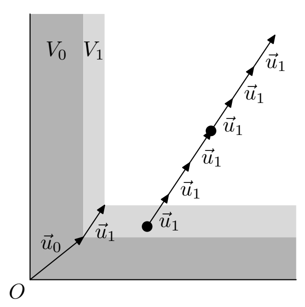
$$V_0 = \{\, \vec{v} \in V_Q \mid \vec{u}_0 \not\leq \vec{v} \,\}, \quad V_1 = \{\, \vec{v} \in V_Q \mid \vec{u}_0 < \vec{v}, \vec{u}_0 + \vec{u}_1 \not\leq \vec{v} \,\}.$$

Since the semilinear sets are closed under intersection, it is not difficult to see using Lemma 4 that there are $k, l \geq 1$ such that V_0 and V_1 are both finite unions of linear sets whose set of generators is one of $\varnothing$, $\{(k,0)\}$, and $\{(0,l)\}$.

Now we claim

$$V_Q - V_0 = \bigcup \{\, L(\vec{t}, \{\vec{u}_1\}) \mid \vec{t} \in V_1 \,\}. \qquad (\ddagger)$$

To see the $\subseteq$ direction of ($\ddagger$), suppose $\vec{v} \in V_Q$ and $\vec{u}_0 \leq \vec{v}$. Then there must be an $n \geq 0$ and $\vec{t} \in \mathbb{N}^2$ such that $\vec{u}_0 \leq \vec{t}$, $\vec{u}_0 + \vec{u}_1 \not\leq \vec{t}$, and $\vec{v} = \vec{t} + n \cdot \vec{u}_1$. Since $\vec{u}_0 \leq \vec{t}$, there is a $z \in \{a,b\}^*$ such that $\vec{t} = \#(x_1x_3z)$. Since $\vec{v} = \#(x_1x_2^nx_3x_4^nz)$, we get $x_1x_2^nx_3x_4^nz \in W_Q$, which implies $x_1x_3z \in W_Q$, by Lemma 3. So $\vec{t} \in V_1$ and $\vec{v} \in L(\vec{t}, \{\vec{u}_1\})$.



To see the $\supseteq$ direction of ($\ddagger$), let $\vec{t} \in V_1$. Then there is a $z \in \{a,b\}^*$ such that $\vec{t} = \#(x_1x_3z)$. Since $\vec{t} \in V_1$, $x_1x_3z \in W_Q$, and this implies $x_1x_2^nx_3x_4^2z$ for all $n \geq 0$, by Lemma 3. So $\vec{t} + n \cdot \vec{u}_1 \in V_Q$ for all $n \geq 0$, i.e., $L(\vec{t}, \{\vec{u}_1\}) \subseteq V_Q$. Since $\vec{t} \notin V_0$, it is clear that $L(\vec{t}, \{\vec{u}_1\}) \subseteq V_Q - V_0$.

Let $(m,n) = \vec{u}_1$. Now we can show that $V_Q - V_0$ is a finite union of linear sets whose set of generators is one of $\{(m,n)\}$, $\{(k,0),(m,n)\}$, and $\{(0,l),(m,n)\}$. Suppose $V_1 = L(\vec{t}_1, G_1) \cup \cdots \cup L(\vec{t}_q, G_q)$, where each G_i is one of $\varnothing$, $\{(k,0)\}$, and $\{(0,l)\}$. Then $V_Q - V_0 = L(\vec{t}_1, G_1') \cup \cdots \cup L(\vec{t}_q, G_q')$, where $G_i' = G_i \cup \{(m,n)\}$. Clearly, G_i' is among $\{(m,n)\}$, $\{(k,0),(m,n)\}$, $\{(0,l),(m,n)\}$. Since V_0 is a finite union of linear sets whose set of generators is one of $\varnothing$, $\{(k,0)\}$, and $\{(0,l)\}$, we have shown that V_Q is as specified in Theorem 1.

For the "if" direction of Theorem 1, assume that V_Q is as specified in the theorem. We can construct a deterministic PDA recognizing W_Q as follows. We use part of the finite control as buffers to store bounded numbers of as and bs, where the bound for the a-buffer is the maximal a-component (i.e., first component) of the offsets plus m, and likewise for the b-buffer. When scanning an a when the a-buffer is full, we push an a onto the stack, and likewise when scanning a b when the b-buffer is full. When both buffers become full, we take out m as and n bs from their respective buffer, and move the symbols on the stack to the appropriate buffer until

[8]The inequality $\leq$ between vectors in $\mathbb{N}^2$ is defined by: $(x,y) \leq (x',y')$ iff $x \leq x'$ and $y \leq y'$. The strict inequality $(x,y) < (x',y')$ holds iff $(x,y) \leq (x',y')$ and $(x,y) \neq (x',y')$.

the buffer becomes full or the stack becomes empty (using bottom-of-stack symbol to test for emptiness), whichever comes first. We also count the number of as (or bs) on the stack modulo k (or l) and keep this information in the finite control. Given this much information in the finite control, inspection of a bounded portion of the stack near the top suffices to determine whether the part of the input scanned so far is in W_Q.

This deterministic PDA always keeps its stack uniform—the stack always contains just one kind of symbol (besides the bottom-of-stack symbol). Using a flag in the finite control to indicate which symbol is in the stack, we can easily turn it into a *one-counter machine* (1-CM) (Fischer et al., 1968), which is like a deterministic PDA but has a counter instead of a pushdown stack, which can hold any natural number and can be tested for zero.

Let us see the working of our 1-CM in more detail. If either $m = 0$ or $n = 0$, it is easy to see that W_Q is regular, so we may assume $m > 0$ and $n > 0$. Since a regular set can be recognized using just the finite control, we may also discard linear sets of the form $L(\vec{u}, \varnothing)$, $L(\vec{u}, \{(k, 0)\})$, or $L(\vec{u}, \{(0, l)\})$, which correspond to regular subsets of W_Q. Writing $L(O; G)$ for $\bigcup\{ L(\vec{u}; G) \mid u \in O \}$, we assume

$$V_Q = L(O_1; \{(m, n)\}) \cup L(O_2; \{(k, 0), (m, n)\}) \cup L(O_3; \{(0, l), (m, n)\}),$$

where O_1, O_2, O_3 are finite subsets of $\mathbb{N}^2$. Let[9]

$$\begin{aligned} \mathit{max\_offset\_a} &= \max\{ x \mid (x, y) \in O_1 \cup O_2 \cup O_3 \}, \\ \mathit{max\_offset\_b} &= \max\{ y \mid (x, y) \in O_1 \cup O_2 \cup O_3 \}, \\ C &= \max(n \cdot \lfloor \mathit{max\_offset\_a}/m \rfloor, m \cdot \lfloor \mathit{max\_offset\_b}/n \rfloor). \end{aligned}$$

Our 1-CM is an implementation of the following pseudocode:

```
$a\_buffer \leftarrow 0$; $b\_buffer \leftarrow 0$; $count \leftarrow 0$; $rem \leftarrow 0$; $counted \leftarrow a$
loop
    $accept \leftarrow$ CheckForAcceptance()
    if EndOfInput() then
        return $accept$
    else
        $c \leftarrow$ GetNextSymbol()
        if $c = a$ then
            if $a\_buffer = max\_offset\_a + m$ then
                $counted \leftarrow a$; $count \leftarrow count + 1$; $rem \leftarrow (rem + 1) \bmod k$
            else
                $a\_buffer \leftarrow a\_buffer + 1$
            end if
        end if
        if $c = b$ then
            if $b\_buffer = max\_offset\_b + n$ then
                $counted \leftarrow b$; $count \leftarrow count + 1$; $rem \leftarrow (rem + 1) \bmod l$
            else
                $b\_buffer \leftarrow b\_buffer + 1$
            end if
        end if
        if $(a\_buffer, b\_buffer) = (max\_offset\_a + m, max\_offset\_b + n)$ then
            $(a\_buffer, b\_buffer) \leftarrow (a\_buffer, b\_buffer) - (m, n)$
            if $counted = a$ then
```

[9]If x is a real number, $\lfloor x \rfloor$ is the integer part of x.

```
                while $count > 0 \land a\_buffer < max\_offset\_a + m$ do
                    $count \leftarrow count - 1$; $rem \leftarrow (rem - 1) \bmod k$; $a\_buffer \leftarrow a\_buffer + 1$
                end while
            end if
            if $counted = b$ then
                while $count > 0 \land b\_buffer < max\_offset\_b + n$ do
                    $count \leftarrow count - 1$; $rem \leftarrow (rem - 1) \bmod l$; $b\_buffer \leftarrow b\_buffer + 1$
                end while
            end if
        end if
    end if
end loop

procedure CheckForAcceptance()
    if $count \leq C$ then
        if $(counted = a \land (a\_buffer + count, b\_buffer) \in V_Q) \lor (counted = b \land (a\_buffer, b\_buffer + count) \in V_Q)$ then
            return true
        end if
    else
        for all $(x, y) \in O_2$ do
            if $(x, y) \leq (a\_buffer, b\_buffer)$ then
                $(x_1, y_1) \leftarrow (a\_buffer - x, b\_buffer - y)$
                if $counted = a \land y_1 \equiv 0 \pmod{n} \land x_1 - m \cdot (y_1/n) + rem \equiv 0 \pmod{k}$ then
                    return true
                end if
            end if
        end for
        for all $(x, y) \in O_3$ do
            if $(x, y) \leq (a\_buffer, b\_buffer)$ then
                $(x_1, y_1) \leftarrow (a\_buffer - x, b\_buffer - y)$
                if $counted = b \land x_1 \equiv 0 \pmod{m} \land y_1 - n \cdot (x_1/m) + rem \equiv 0 \pmod{l}$ then
                    return true
                end if
            end if
        end for
    end if
    return false
end procedure
```

The length of any legal sequence of ε-transitions of this 1-CM is bounded by a constant. Using the compression technique of Fischer et al. (1968, proof of Theorem 1.1), we can eliminate ε-transitions from the machine to make it operate in *real time* (Fischer et al., 1968).

Corollary 5. *A type $\langle 1 \rangle$ quantifier is recognized by a deterministic PDA if and only if it is real-time recognized by a (deterministic) 1-CM.*

4 Conclusion

I have characterized the type $\langle 1 \rangle$ quantifiers recognized by deterministic PDAs in terms of the associated semilinear sets of vectors and showed that exactly the same type $\langle 1 \rangle$ quantifiers are recognized by (deterministic) 1-CMs (in real time). Let me make a few more related observations.

A theorem of Fischer et al. (1968, Theorem 2.2) says that a language of the form $\{ w \in \Sigma^* \mid \#(w) \in S \}$ is a finite Boolean combination of languages real-time recognizable by 1-CMs if and only if S is semilinear. So we have

Corollary 6. *A type $\langle 1 \rangle$ quantifier is recognized by a nondeterministic PDA if and only if it is a finite Boolean combination of type $\langle 1 \rangle$ quantifiers recognized by deterministic PDAs.*

It is relatively straightforward to show that whenever $S \subseteq \mathbb{N}^2$ is semilinear, $\{ w \in \{a, b\}^* \mid \#(w) \in S \}$ is accepted by a *nondeterministic one-counter machine*, which gives us

Proposition 7. *A type $\langle 1 \rangle$ quantifier is accepted by a nondeterministic PDA if and only if it is accepted by a nondeterministic one-counter machine.*

By a similar pumping argument, we can also give a simple characterization of the semilinear sets associated with type $\langle 1 \rangle$ quantifiers recognized by finite automata:

Proposition 8. *A type $\langle 1 \rangle$ quantifier Q is recognized by a finite automaton if and only if there exist natural numbers k, l such that V_Q is a finite union of linear sets each of which has one of the following as its set of generators:*

$$\varnothing, \quad \{(k, 0)\}, \quad \{(0, l)\}, \quad \{(k, 0), (0, l)\}.$$

References

Naoki Abe. Characterizing PAC-learnability of semilinear sets. *Informaiton and Computation*, 116:81–102, 1995.

Patrick C. Fischer, Albert R. Meyer, and Arnold L. Rosenberg. Counter machines and counter languages. *Mathematical Systems Theory*, 2:265–283, 1968.

Seymour Ginsburg and Edwin H. Spanier. Semigroups, Presburger formulas, and languages. *Pacific Journal of Mathematics*, 16:285–296, 1966.

Michael A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, Reading, MA, 1978.

John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA., 1979.

Marcin Mostowski. Computational semantics for monadic quantifiers. *Journal of Applied Non-Classical Logics*, 8:107–201, 1998.

Rohit J. Parikh. On context-free languages. *Journal of the ACM*, 13:570–581, 1966.

Stanley Peters and Dag Westerståhl. *Quantifiers in Language and Logic.* Oxford University Press, Oxford, 2006.

Jakub Szymanik and Marcin Zajenkowski. Contribution of working memory in parity and proportional judgments. *Belgian Journal of Linguistics*, 25:176–194, 2011.

Johan van Benthem. *Essays in Logical Semantics.* Reidel, Dordrecht, 1986.

Marcin Zajenkowski and Jakub Szymanik. MOST intelligent pepole are accurate and SOME fast people are intelligent. Intelligence, working memory, and semantic processing of quantifiers from a computational perspective. *Intelligence*, 41:456–466, 2013.